Lösungen zu ausgewählten Aufgaben

> Man lernt das Klavierspielen nicht
> durch den Besuch von Konzerten.
>
> Carl Runge

Lösungen zur **1. Auflage**. *Letzte Aktualisierung:* **26. Oktober 2024**.

Für Aufgaben, bei denen es einfach und offensichtlich ist, wie man die Ergebnisse selbst überprüfen kann, werden keine Lösungen angegeben. (Ohnehin sollten Sie *immer* Ihre Resultate selbst prüfen und nicht blind mir oder jemand anderem trauen.) Außerdem gibt es meistens mehr als einen Weg, zur Lösung zu kommen. Wenn mein Vorschlag von Ihrem Lösungsweg abweicht, heißt das nicht, dass Ihrer falsch ist.[1] In diesem Fall sollten Sie aber vielleicht Ihre Lösung erneut überprüfen.

Manchmal habe ich auch deshalb keine Lösung explizit aufgeschrieben, weil die Lösung im Text direkt danach besprochen wird. Dann war die Aufgabe so intendiert, dass Sie vor dem Weiterlesen erst einmal selbst nachdenken sollten.

Bei Programmieraufgaben habe ich mich bemüht, nur PYTHON-Konstrukte zu benutzen, die im Buch schon vor der Aufgabe erklärt wurden. Ab und zu werden aber auch in den Lösungen neue Dinge eingeführt, die dann im weiteren Verlauf des Buches ohne weiteren Kommentar verwendet werden.

Lösungen zu Programmieraufgaben sollten Sie in stilistischer Hinsicht *nicht* als exemplarisch betrachten; es wird üblicherweise elegantere, effizientere oder idiomatischere Wege geben, denselben Sachverhalt auszudrücken. Vergessen Sie nicht, dass dies in erster Linie ein Buch über Mathematik und kein Programmierlehrbuch ist!

[1] Es sei denn, die Lösung ist einfach „ja“ oder „nein“ oder eine Zahl. Wenn wir mal davon ausgehen, dass *meine* Lösung richtig ist…

E. Weitz, *Konkrete Mathematik (nicht nur) für Informatiker*, https://doi.org/10.1007/978-3-658-21565-1_1

Lösung 5: `a += 10` ist eine Abkürzung für `a = a + 10`, weil sowas oft gebraucht wird. Ähnliche Abkürzungen gibt es für andere zweistellige Operatoren, z.B. `-=`, `*=`, und so weiter.

Es kann allerdings auch subtile Unterschiede zwischen der „Abkürzung“ und der langen Schreibweise geben. Siehe z.B. Aufgabe 241.

Lösung 9: Hier ist eine Version:

```
def sumFn2 (n):
    s = 0
    while n != 0:
        s += n
        if n > 0:
            n -= 1
        else:
            n += 1
    return s
```

Die zweite Version setzt voraus, dass `sumFn` schon existiert:

```
def sumFn3 (n):
    if n >= 0:
        return sumFn(n)
    else:
        return -sumFn(-n)
```

Das Wiederverwenden bestehender Lösungen wie im Falle von `sumFn3` ist üblicherweise dem „Neuerfinden des Rades“ wie bei `sumFn2` vorzuziehen. Es ist eleganter und weniger fehlerträchtig.

Lösung 10: Siehe Seite 15.

Lösung 11: Die Fakultät von null, also 0!, hat nach Definition den Wert 1. Das ist der einzig sinnvolle Wert, weil 1 das neutrale Element der Multiplikation ist. Wenn man mit 1 anfängt, ist es so, als hätte man noch gar nicht multipliziert. Siehe dazu auch Aufgabe 285.

Lösung 12: Vielleicht so:

```
def myAbs (x):
    if x >= 0:
        return x
    else:
        return -x
```

Und hier eine deutlich kürzere Version:

```
def myAbs (x):
    return x if x >= 0 else -x
```

Das ist der sogenannte *ternäre Operator*, den Sie auch in Sprachen wie Java oder C finden werden.[2] Mit dieser Abkürzung kann man sich oft viel Schreibarbeit sparen.

Die Python-Funktion für den Absolutbetrag heißt übrigens `abs`.

Lösung 13: `abs` und der ternäre Operator wurden in der vorherigen Lösung eingeführt:

```
def farther (x, y):
    return x if abs(x) > abs(y) else y
```

Lösung 14: In der Aufgabe wurde nicht spezifiziert, was getan werden sollte, wenn beide Argumente gleich weit von null entfernt sind. Soll dann das erste (oder das zweite) Argument zurückgegeben werden? Oder das größere von beiden (falls sie verschieden sind)? Oder die Spezifikation könnte auch einfach sagen, dass das keine Rolle spielt. Aber dieser Fall sollte zumindest erwähnt werden.

Lösung 15: Dies ähnelt sehr einigen Funktionen, die wir schon gesehen haben:

```
def power(a, b):
    p = 1
    while b > 0:
        p *= a
        b -= 1
    return p
```

Die Reihenfolge spielt natürlich eine Rolle. Z.B. sind 2^3 und 3^2 verschieden.

Lösung 16: Die erste Funktion, die im Buch vorkommt, ist `print`. Sie gibt keinen Wert zurück.[3] Die Funktion wird nur wegen des sogenannten *Nebeneffekts* aufgerufen.

Lösung 17: Diese Funktion zeigt zwar das richtige Ergebnis an, sie gibt es aber nicht zurück. Daher können andere Funktionen sie nicht sinnvoll verwenden. Sie sollte stattdessen so aussehen:

```
def Max (a, b):
    if a > b:
        return a
    else:
        return b
```

Nun können Sie z.B. ganz einfach eine Funktion schreiben, die das Maximum von drei Zahlen berechnet:

[2] Dort allerdings mit einer kryptischeren Syntax mit Fragezeichen und Doppelpunkt.

[3] Führen Sie `a = print(42)` aus und schauen Sie sich dann den Wert von `a` an.

```
def Max3 (a, b, c):
    return Max(Max(a, b), c)
```

Mit der ursprünglichen Version von `Max` hätte das nicht funktioniert.

Python hat übrigens schon eine eingebaute Funktion `max`, die mit beliebig vielen Argumenten zurechtkommt und die auch mit *Listen* arbeiten kann, die im nächsten Kapitel vorgestellt werden.

Lösung 18: Die *meisten* Python-Funktionen verhalten sich wie mathematische Funktionen: sie akzeptieren Argumente und geben Werte zurück, die von diesen Argumenten abhängen. (In der Mathematik würde man sagen, dass die Funktion die Argumente auf die Werte *abbildet*.) Eine weitere Gemeinsamkeit ist, dass bei manchen Funktionen manche Argumente nicht „erlaubt“ sind. In der Mathematik nennt man die Menge der „erlaubten“ Argumente den *Definitionsbereich* der Funktion.

Und hier ein paar Unterschiede. (Machen Sie sich keine Gedanken, falls Ihnen das nicht eingefallen ist. Insbesondere dann, wenn Sie noch nie programmiert haben.)

- Mathematische Funktionen sind immer für eine feste Zahl von Argumenten definiert. In Python können Funktionen *optionale* Argumente haben.
- Python-Funktionen können *Nebeneffekte* haben; siehe Aufgabe 16.
- Manche Python-Funktionen geben keinen Wert zurück; siehe Aufgabe 17.
- Der wohl wichtigste Unterschied: Mathematische Funktionen bilden immer dieselben Argumente auf dieselben Werte ab.[4] In Computerprogrammen ist das nicht notwendigerweise der Fall. (Denken Sie an eine Funktion, die die Tageszeit oder eine Zufallszahl zurückgibt.)

In Kapitel 19 werden wir uns noch einmal ausführlich mit (mathematischen) Funktionen beschäftigen.

Lösung 19: Das Produkt $30!$ enthält den Faktor 10, also muss am Ende der Zahl mindestens eine Null stehen. (Tatsächlich stehen sogar sieben Nullen am Ende. Wo kommen die her?)

Lösung 20: Jeder Quotient ist entweder größer als 1 oder er *ist* 1. Im ersten Fall ist der *nächste* Quotient nach der Division durch 2 auf keinen Fall kleiner als 1 und der Algorithmus läuft daher weiter. Im zweiten Fall handelt es sich um den letzten Schritt des Algorithmus. Dies bedeutet, dass das Ergebnis immer mit einer Eins anfängt und niemals mit überflüssigen führenden Nullen.

Lösung 21: Dezimal ist die größte Zahl 9999_{10}, binär $1111_2 = 15_{10}$.

Lösung 22: Die größte Zahl, die man mit n binären Ziffern darstellen kann, ist $2^n - 1$.

Lösung 23: Nach der letzten Aufgabe benötigt man n binäre Ziffern, um alle Zahlen, die kleiner als 2^n sind, darzustellen. Mit anderen Worten, wenn k eine Zweierpotenz

[4] Sogenannte *funktionale* Programmiersprachen übernehmen dieses Verhalten.

ist, braucht man $\log_2 k$ Ziffern für alle Zahlen, die kleiner als k sind. Wenn k *keine* Zweierpotenz ist, braucht man offenbar so viele Ziffern wie man bis zur *nächsten* Zweierpotenz gebraucht hätte. Die gesuchte Zahl ist in diesem Fall also die erste ganze Zahl, die größer als $\log_2 k$ ist.

Der mathematische Ausdruck, der beide Fälle (ob k Zweierpotenz ist oder nicht) zusammenfasst, ist $\lceil \log_2 k \rceil$. Dabei werden sogenannte *Gaußklammern* benutzt.[5] (Im Englischen nennt man das die *ceiling function* und so oder so ähnlich heißt sie oft auch in Programmiersprachen.)

Lösung 26: 8 und 16 sind beides Zweierpotenzen. Eine Oktalziffer entspricht exakt drei Binärziffern, eine Hexadezimalziffer entspricht genau vier Binärziffern. Dadurch wird die Konvertierung zwischen einem dieser Systeme und dem Binärsystem sehr einfach. Binärzahlen werden schnell sehr lang und unhandlich; durch das Oktal- oder das Hexadezimalsystem kann man die Binärziffern in Gruppen zusammenfassen und übersichtlicher darstellen.

(Direkte Konvertierung zwischen Oktal- und Hexadezimalsystem ist nicht so einfach. Und die Konvertierung zwischen diesen beiden und dem Dezimalsystem macht noch weniger Spaß.)

Lösung 27: Die „Bauernmultiplikation“ ist eine Art Zusammenfassung all der Dinge, die wir bisher über das Binärsystem gelernt haben. Wenn die Gruben auf der linken Seite gefüllt werden, entspricht das der wiederholten Division durch 2 beim Konvertieren von dezimal nach binär.[6] Wir entfernen die Gruben mit geraden Zahlen, weil das den Nullen im ersten Faktor entspricht.

Was dann auf der rechten Seite passiert, entspricht der schriftlichen binären Multiplikation. Jede Verdoppelung ist ein Verschieben der Zahl um eine Stelle nach links.

Lösung 28: Because *Oct 31* equals *Dec 25*.[7]

Lösung 29: Es ist wie bei den Binärzahlen, nur mit anderen Präfixen: `0o` und `0x`.

Die Buchstaben braucht man, weil man für ein System mit der Basis n in der Lage sein muss, die Zahlen von 0 bis $n-1$ (also n verschiedene) durch jeweils eine Ziffer darzustellen. Für das Hexadezimalsystem braucht man daher sechzehn Ziffern, aber es gibt nur zehn Dezimalziffern. Für die restlichen sechs nimmt man Buchstaben.

Lösung 30: Erinnern Sie sich, dass eine Zahl nur ein abstraktes Konzept ist. Eine Zahl ist einfach eine Zahl. Nur ihre *Darstellung* kann binär oder dezimal oder was auch immer sein. Korrekter wäre es also, von einer „Zahl in binärer Repräsentation“ zu sprechen. Aber „Binärzahl“ ist, wenn auch nicht ganz korrekt, kürzer und knackiger.

Lösung 31: Als wir den Algorithmus auf Seite 19 entwickelt haben, haben wir in jedem Schritt verdoppelt *bis auf den letzten*. Die PYTHON-Funktion verdoppelt in jedem

[5] Macht nichts, wenn Sie das nicht wussten. Es reicht, wenn Sie die prinzipielle Idee mit Ihren eigenen Worten beschreiben konnten.

[6] Beachten Sie, dass es dem Algorithmus „egal“ ist, ob die zu konvertierende Zahl im Dezimalsystem vorliegt. Er konvertiert *jede* Zahl.

[7] Und es gibt sogar eine Wikipedia-Seite, die den Witz erklärt.

Schritt, also müssen wir die letzte Verdopplung durch Division wieder rückgängig machen.

Alternativ können wir aber auch auf das Dividieren am Ende verzichten, wenn wir die Reihenfolge der beiden Operationen in der Schleife vertauschen:

```
def convBinToDec (binList):
    binList = list(reversed(binList))
    result = 0
    while len(binList) > 0:
        result = result * 2
        result = result + binList.pop()
    return result
```

Siehe auch Aufgabe 114.

Lösung 32: Das haben Sie hoffentlich selbst ausprobiert! Das Resultat *als Zahl* ist identisch, aber die *Repräsentation* ist eine andere. Die Originalfunktion gab eine ganze Zahl zurück, die mit `/` statt `//` eine Fließkommazahl.

Lösung 33: Hier eine Beispiellösung:

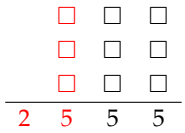
```
def convDecToBin (n):
    if n == 0:
        return [0]
    result = []
    while n > 0:
        result.append(n % 2)
        n = n // 2
    return list(reversed(result))
```

Lösung 35: `a % b` ist dasselbe wie `a - (a // b) * b`.

Lösung 37: Die Funktion `convBinToDec` modifiziert den ersten Parameter `binList` in der ersten Zeile. Hätte man tatsächlich jedes Auftreten von `binList` durch das Argument *ersetzt*, würde der daraus resultierende Code keinen Sinn mehr ergeben.

Lösung 38: Die Ziffer 1 wird nicht verwendet.

Wir fangen mit der linken, hier rot markierten, Spalte von Ziffern an:



Die Summe dieser drei Ziffern muss, zusammen mit einem eventuellen Übertrag aus der mittleren Spalte, 25 sein. Wählt man die größten drei Dezimalziffern 7, 8 und 9, so ergibt sich als Summe 24; man würde also den Übertrag 1 benötigen. Wählt

man andere Ziffern, so ergibt sich immer eine kleinere Summe. Die zweitgrößte Möglichkeit wäre $6+8+9 = 23$. Dann bräuchte man schon den Übertrag 2. Mit den verbleibenden Ziffern (7 und 0 bis 5) kann man in den beiden rechten Spalten aber maximal die Summe $(70+50+40)+(3+2+1) = 166$ erreichen. Das reicht nur für den Übertrag 1. Die erste Spalte muss also so aussehen:[8]

$$\begin{array}{cccc} & 7 & \square & \square \\ & 8 & \square & \square \\ & 9 & \square & \square \\ & {\scriptstyle 1} & & \\ \hline 2 & 5 & 5 & 5 \end{array}$$

Nun muss sich in der mittleren Spalte 15 ergeben. Mit demselben Argument wie eben folgt, dass das nur möglich ist, wenn in der mittleren Spalte 4, 5 und 6 stehen.

$$\begin{array}{cccc} & 7 & 4 & \square \\ & 8 & 5 & \square \\ & 9 & 6 & \square \\ & {\scriptstyle 1} & & \\ \hline 2 & 5 & 5 & 5 \end{array}$$

In der letzten Spalte muss sich nun 5 ergeben und wir haben nur noch die vier Ziffern 0 bis 3 zur Verfügung. Die einzige Möglichkeit, aus diesen Ziffern drei verschiedene auszuwählen, deren Summe 5 ist, ist $0+2+3$.

$$\begin{array}{cccc} & 7 & 4 & 0 \\ & 8 & 5 & 2 \\ & 9 & 6 & 3 \\ & {\scriptstyle 1} & & \\ \hline 2 & 5 & 5 & 5 \end{array}$$

Übrigens gibt es noch eine schnellere Methode, auf die Antwort zu kommen, wenn man sich mit modularer Arithmetik (Kapitel 3) auskennt. Die Summe aller zehn Dezimalziffern von 0 bis 9 ist 45 und damit durch 9 teilbar. Andererseits ist 8 die iterierte Quersumme von 2555. Es „fehlt“ also gerade die Eins.

Lösung 39: Der Anfang ist offensichtlich. Zum Beispiel muss links und rechts neben den beiden Nullen in der zweiten Zeile jeweils eine Eins stehen, damit keine drei Nullen in einer Reihe stehen. Ebenso muss zwischen den beiden Einsen in der sechsten Zeile eine Null stehen. Wenn Sie nach diesem System (natürlich auch für Spalten) fortfahren, ist irgendwann mal eine Zeile komplett gefüllt. Daran erkennen Sie, wie viele Einsen und Nullen in dieser Zeile stehen. Nach den Spielregeln muss das für alle Zeilen gelten. Dann wissen Sie aber auch, wie viele Einsen und Nullen es insgesamt geben muss, und können daraus schließen, wie viele Einsen und Nullen in jeder Spalte stehen müssen. Mit diesem Wissen können Sie nun weiter auffüllen. Wenn beispielsweise in einer Spalte nur noch eine Null fehlt, dann wird es Positionen geben,

[8] Die Reihenfolge spielt für die Lösung offenbar keine Rolle. Es gibt verschiedene Möglichkeiten, die Kästchen zu füllen, aber es ändert sich jeweils nur die Reihenfolge innerhalb der Spalten. Siehe dazu auch Aufgabe 330.

an denen diese *nicht* stehen kann, weil es sonst an einer anderen Stelle zu viele Einsen gäbe. Dort, wo keine Null stehen kann, muss aber eine Eins stehen. Und so weiter. Man kann so nach und nach alle Felder füllen und benötigt dafür nicht mal die letzten beiden Regeln.

Falls ich mich nicht vertippt habe, sieht die Lösung so aus:

0	**1**	**1**	**0**	**1**	**0**	**0**	**1**	**1**	**0**	**1**	**0**
1	**0**	**0**	**1**	**0**	**1**	**0**	**0**	**1**	**0**	**1**	**1**
1	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**0**	**0**
0	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**
1	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**0**	**1**	**0**
0	**1**	**0**	**1**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**
0	**1**	**0**	**0**	**1**	**1**	**0**	**1**	**0**	**1**	**0**	**1**
1	**0**	**1**	**0**	**0**	**1**	**1**	**0**	**1**	**1**	**0**	**0**
0	**0**	**1**	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**1**
1	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**0**	**1**
0	**0**	**1**	**1**	**0**	**1**	**1**	**0**	**0**	**1**	**1**	**0**
1	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**0**	**1**

Und wenn Sie ein Programm geschrieben haben, dann hoffentlich nicht eins, das einfach alle möglichen Lösungen „blind“ durchprobiert. Dafür gibt es nämlich 2^{114} Möglichkeiten (siehe Kapitel 17) und Ihre Lebenszeit würde bei weitem nicht ausreichen, um auf das Ergebnis zu warten...

Lösung 40: Nach einigem Überlegen kann man auf die Idee kommen, sich das zu wiegende Gewicht als Binärzahl vorzustellen. Zum Beispiel ist $19_{10} = 10011_2$, d.h. 19 ist die Summe von 16, 2 und 1. 19 kg ließen sich also mit den drei Gewichtsstücken 1 kg, 2 kg und 16 kg auswiegen. Für Zahlen, die kleiner als $128 = 2^7$ sind, braucht man maximal sieben Binärziffern, d.h. jedes Gewicht bis einschließlich 127 (und damit „erst recht“ bis 80) lässt sich mit den sieben Kilogewichten 1, 2, 4, 8, 16, 32 und 64 kombinieren.

Man kann das aber noch verbessern: Zunächst wird man vielleicht merken, dass man eigentlich nur geradzahlige Gewichte braucht. Ist z.B. der zu wiegende Gegenstand leichter als 14 kg und schwerer als 12 kg, so kann er nur 13 kg wiegen. Man kann daher auf das leichteste Gewichtsstück (1 kg) verzichten, weil man mit den anderen sechs alle geraden Gewichte durch Addition erhalten kann.

Tatsächlich geht es aber noch besser: Legt man z.B. den zu wiegenden Gegenstand zusammen mit 2 kg in eine Waagschale und 6 kg in die andere, so ermittelt man, ob der Gegenstand genau 4 kg (oder mehr oder weniger) wiegt. Man arbeitet also mit der *Differenz* $6 - 2$. Betrachtet man die Dreierpotenzen 1, 3, 9 und 27, so kann man alle Zahlen von 1 bis 40 als Summen bzw. Differenzen dieser vier Zahlen erhalten:

$$1 = 1 \qquad 2 = 3 - 1 \qquad 3 = 3$$
$$4 = 3 + 1 \qquad 5 = 9 - 3 - 1 \qquad 6 = 9 - 3$$
$$7 = 9 - 3 + 1 \qquad 8 = 9 - 1 \qquad 9 = 9$$

$10 = 9 + 1$ $11 = 9 + 3 - 1$ $12 = 9 + 3$
$13 = 9 + 3 + 1$ $14 = 27 - 9 - 3 - 1$ $15 = 27 - 9 - 3$
$16 = 27 - 9 - 3 + 1$ $17 = 27 - 9 - 1$ $18 = 27 - 9$
$19 = 27 - 9 + 1$ $20 = 27 - 9 + 3 - 1$ $21 = 27 - 9 + 3$
$22 = 27 - 9 + 3 + 1$ $23 = 27 - 3 - 1$ $24 = 27 - 3$
$25 = 27 - 3 + 1$ $26 = 27 - 1$ $27 = 27$
$28 = 27 + 1$ $29 = 27 + 3 - 1$ $30 = 27 + 3$
$31 = 27 + 3 + 1$ $32 = 27 + 9 - 3 - 1$ $33 = 27 + 9 - 3$
$34 = 27 + 9 - 3 + 1$ $35 = 27 + 9 - 1$ $36 = 27 + 9$
$37 = 27 + 9 + 1$ $38 = 27 + 9 + 3 - 1$ $39 = 27 + 9 + 3$
$40 = 27 + 9 + 3 + 1$

Entsprechend kann man alle *geraden* Zahlen von 2 bis 80 aus den vier Zahlen 2, 6, 18 und 54 kombinieren. Man kommt also mit *vier* Gewichtsstücken aus.

Der Zusammenhang mit den Zweierpotenzen am Anfang ist relativ einleuchtend, aber was hat diese verbesserte Lösung mit den Dreierpotenzen zu tun? Man kann jede Zahl natürlich auch zur Basis 3 darstellen, wobei man als Ziffern 0, 1 und 2 zur Verfügung hat. Z.B. ist $19_{10} = 201_3$, d.h. die Zahl 19 lässt sich als Summe in der Form $9 + 9 + 1$ darstellen. Diese Darstellung ist aber auf den ersten Blick nur dann hilfreich, wenn man von jedem Gewichtsstück *zwei* Exemplare hat. Allerdings kann man sich folgendermaßen an das Ziel herantasten: Statt $9 + 9$ addiert man noch mehr, nämlich $9 + 9 + 9$; das entspricht dann genau dem nächsten Gewichtsstück 27. Den „Überschuss“ von 9 kann man nun wieder abziehen, weil man dieses Stück für die Addition ja gar nicht benutzt hat. So kann man prinzipiell mit jeder 2 in der ternären Darstellung eines Gewichts vorgehen. (Man spricht auch vom *balancierten Ternärsystem*.)

Nun sollte man sich noch überlegen, warum drei Stücke *nicht* reichen. Das sieht man aber sofort an der obigen Übersicht: Für die Zahlen 1 bis 13 wurden tatsächlich *alle* Möglichkeiten ausgenutzt, die drei Zahlen 1, 3 und 9 so zu kombinieren, dass durch Additionen und Subtraktionen eine positive Zahl herauskommt. Man kann also offensichtlich mit drei Gewichtsstücken nicht mehr als 13 verschiedene Messungen durchführen.

Diese Aufgabe ist auch als *Wiegeproblem von Bachet* bekannt und wird einem französischen Mathematiker aus dem 17. Jahrhundert zugeschrieben. Inzwischen glaubt man aber, dass das Problem sogar schon im Mittelalter bekannt war.

Lösung 41: Das C-Programm verwendete nicht alle 64 Bits pro Ganzzahl, sondern eine kleinere Zahl. Der C-Standard schreibt Compilern nicht genau vor, wie viele Bits eine Zahl vom Typ `int` haben muss. Siehe auch Aufgabe 58.

Lösung 43: Der Wert von `1 << n` ist 2^n. Merken Sie sich das als einen effizienten Weg, Zweierpotenzen zu berechnen.

Lösung 44: 14 ist 1110 in binärer Notation (oder $14 = 8 + 4 + 2$) und daher gilt

$$14a = 8a + 4a + 2a.$$

Die drei Summanden kann man durch Verschiebungen berechnen, `4 * a` ist z.B. dasselbe wie `a << 2`:

```
def multWith14 (a):
    return (a << 3) + (a << 2) + (a << 1)
```

Beachten Sie, dass dieses Vorgehen im Prinzip der binären Multiplikation mit Bleistift und Papier entspricht.

Lösung 45: 1000_{10} ist durch $8_{10} = 1000_2$ teilbar, daher müssen die drei Ziffern ganz rechts drei Nullen sein.

Lösung 47: Das hier wollen Sie berechnen:

```
(186 * 25) % 256
```

Das ergibt `42`. Um zu überprüfen, dass das korrekt war, können Sie das hier eingeben:

```
"{:b}".format(186 * 25)
```

Nun sollten Sie sehen, dass die acht Ziffern ganz rechts die Zahl 42 repräsentieren.

Lösung 48: Die größte Zahl, die man mit acht Bits darstellen kann, ist 255; siehe Aufgabe 22. Also ist $255 \cdot 255 = 65025$ das größtmögliche Produkt. Nach Aufgabe 23 brauchen wir dafür 16 Bits, also doppelt so viele wie für einen Faktor.

Lösung 50: Die Skizze zeigt nur 2, 7, 12, 17 und 22. Aber natürlich sind z.B. 27, 102 und 1027 auch kongruent zu 2 modulo 5. Es gibt unendlich viele Zahlen mit dieser Eigenschaft. (Und später werden wir sehen, dass dazu auch negative Zahlen gehören.)

Lösung 51: Die andere Interpretation war, dass a und b kongruent modulo n sind, wenn ihr Abstand durch n teilbar ist:

```
def congruentModulo (a, b, n):
    return dist(a, b) % n == 0
```

Wir benutzen hier die Funktion `dist` von Seite 10.

Lösung 52: Bei der modularen Arithmetik geht es zentral um Teilbarkeit. Aber man kann nicht durch null teilen, also ergibt es keinen Sinn, modulo null zu rechnen.

Wenn man hingegen modulo eins rechnet, dann sind offenbar *alle* Zahlen kongruent zueinander. Man könnte das also rein technisch machen, es wäre aber nicht sehr spannend.

Lösung 53: Das Symbol „mod“ in einem Ausdruck wie „a mod b“ ist ein *binärer* bzw. *zweistelliger Operator* wie $\cdot$ oder $+$; etwas, das zwei Zahlen auf eine bestimmte Art kombiniert, so dass eine dritte dabei herauskommt.

Das Symbol „mod“ in „$a \equiv b \pmod{n}$“ ist hingegen Teil einer *Aussage*, die entweder wahr oder falsch ist. Ohne die anderen Symbole (wie das $\equiv$) drumherum ergibt das Ganze keinen Sinn.

Lösung 54: Siehe Aufgabe 67.

Lösung 55: 7! ist 5040 und 5040 mod 256 ist 176. Sie sollten natürlich bei beiden Varianten dasselbe Ergebnis erhalten haben!

Lösung 56: Wir wissen, dass sich das Ergebnis modulo 3 nicht ändert, wenn wir die einzelnen Operanden durch andere Zahlen ersetzen, die kongruent zu ihnen sind. 210 ist z.B. offenbar durch 3 teilbar, also kann 211 durch 1 ersetzt werden. Ebenso ist 300 durch 3 teilbar; 302 kann somit durch 2 ersetzt werden. Und 17 ist schließlich kongruent zu 2. Statt $211 \cdot 302 + 17$ können wir also $1 \cdot 2 + 2 = 4$ berechnen, was deutlicher einfacher ist. Der Rest, den wir gesucht haben, ist somit 1, weil es derselbe Rest sein muss, den man erhält, wenn man 4 durch 3 teilt.

Lösung 57: So sollte es aussehen:

$$\begin{aligned} 24 \cdot 12 &= (3 \cdot 7 + 3) \cdot (1 \cdot 7 + 5) = 3 \cdot 7 \cdot 1 \cdot 7 + 3 \cdot 1 \cdot 7 + 3 \cdot 7 \cdot 5 + 3 \cdot 5 \\ &= (3 \cdot 7 \cdot 1 + 3 \cdot 1 + 3 \cdot 5) \cdot 7 + 3 \cdot 5 = 39 \cdot 7 + 3 \cdot 5 \end{aligned}$$

$39 \cdot 7$ ist irrelevant, wenn wir modulo 7 arbeiten, es verbleibt also $3 \cdot 5$. (Die 39 hätten wir natürlich gar nicht ausrechnen müssen. Wir können aufhören zu rechnen, sobald wir sehen, welche Summanden durch 7 teilbar sind.)

Lösung 58: Hier eine PYTHON-Funktion, die diese Arbeit für uns erledigt:

```
def howManyBits ():
  correct = fact(30)
  wrong = 1409286144
  n = 64
  while n > 0:
      if wrong == correct % (1 << n):
          return n
      n -= 1
```

Sie wird uns sagen, dass das C-Programm 32 (statt 64) Bits benutzt hat, um ganze Zahlen (vom Typ `int`) darzustellen. (Siehe Aufgabe 43, falls Ihnen `1 << n` nichts sagt.)

Lösung 59: 2^3 ist 8, modulo 7 ist das 1. Wir haben gerade gelernt, dass $2^6 = (2^3) \cdot (2^3)$ modulo 7 dann $1 \cdot 1 = 1$ ist. Und das gilt natürlich auch für höhere Potenzen:

$$\begin{aligned} a \bmod 7 &= (2^{3000} + 41) \bmod 7 = ((2^3)^{1000} + 41) \bmod 7 \\ &= (1^{1000} + 41) \bmod 7 = 42 \bmod 7 = 0 \end{aligned}$$

Also ist a durch 7 teilbar.

Lösung 60: Der rechten Tabelle entnehmen wir $3 \cdot 4 = 0$ und $5 \cdot 2 = 4$. Aus der linken lesen wir dann $0 + 4 = 4$ ab.

Lösung 62: Wenn Sie immer nur mit den acht Bits ganz rechts arbeiten, arbeiten Sie in $\mathbb{Z}/256\mathbb{Z}$.

Lösung 63: Additionstabellen kann man offenbar ganz einfach dadurch erzeugen, dass man jede Zeile gegenüber der vorherigen zyklisch um eine Spalte nach links schiebt.

Allen Tabellen gemein ist, dass sie bzgl. der Hauptdiagonalen (der von links oben nach rechts unten) symmetrisch sind, d.h. man muss eigentlich nur etwas mehr als die Hälfte der Tabelleneinträge kennen. Der Fachbegriff dafür ist, dass die Operationen *kommutativ* sind. Mit anderen Worten, in $\mathbb{Z}/n\mathbb{Z}$ gilt $a + b = b + a$ und $ab = ba$ wie in $\mathbb{N}$ oder in $\mathbb{Z}$.

Lösung 64: Die „2“ in $\mathbb{Z}/6\mathbb{Z}$ repräsentiert alle Zahlen, die kongruent zu 2 modulo 6 sind. Dazu gehört z.B. 8, aber *nicht* 7. Die „2“ in $\mathbb{Z}/5\mathbb{Z}$ repräsentiert alle Zahlen, die kongruent zu 2 modulo 5 sind. Dazu gehört 7, aber *nicht* 8.

Lösung 65: Das war einfach, oder?

```
def sumMod (a, b, n):
    return (a + b) % n

def prodMod (a, b, n):
    return (a * b) % n
```

Es sei denn, Sie haben die Klammern vergessen! Versuchen Sie Argumente für diese Funktionen zu finden, die zu falschen Ergebnissen führen, falls die Funktionen ohne Klammern in der jeweils letzten Zeile geschrieben werden.

Lösung 66: Man kann leicht nachrechnen, dass 2^3 und 2^{10} *nicht* kongruent modulo 7 sind. (Haben Sie das gemacht?) Falls Sie das überrascht: Warum *sollten* sie kongruent sein? Wir wissen nur, dass Addition und Multiplikation „kompatibel“ mit dem Hin- und Herspringen zwischen normaler und modularer Arithmetik sind. Über Exponentiation wurde noch nichts gesagt. (Und mit gutem Grund, wie das Beispiel zeigt.)

Wenn wir $2 + 3$ in $\mathbb{Z}/7\mathbb{Z}$ berechnen, dann sind 2 und 3 als Elemente von $\mathbb{Z}/7\mathbb{Z}$ jeweils Repräsentanten für unendlich viele andere Zahlen. Berechnen wir jedoch 2^3 in $\mathbb{Z}/7\mathbb{Z}$, dann ist 2 zwar ein Element von $\mathbb{Z}/7\mathbb{Z}$, 3 ist aber einfach die gute alte natürliche Zahl 3, die wir schon aus dem Kindergarten kennen.

Lösung 67: 9, 99, 999 und so weiter sind alle durch 9 teilbar und damit „erst recht“ durch 3, weil $9 = 3 \cdot 3$ gilt. Wir können also genauso argumentieren wie im Falle der 9.

Lösung 68: Es könnte so aussehen:

```
def digSum (n):
    r = n
    s = 0
    while r > 0:
        s = s + r % 10
        r = r // 10
    return s

def digRoot (n):
    r = digSum(n)
    while r >= 10:
        r = digSum(r)
    return r
```

Lösung 70: Ein einfaches Beispiel ist „$11 \cdot 33 = 3\,333$“. Das ist *falsch*, denn $11 \cdot 33$ ist eigentlich 363. Aber die Quersummen von 11 und 33 sind 2 und 6 mit dem Produkt 12. Und die Quersumme von 3 333 ist auch 12.

Lösung 71: Man kann die oktale Quersumme verwenden, um Teilbarkeit durch 7 zu überprüfen:

$$\begin{aligned} 458_{10} = 712_8 &= 7 \cdot 8^2 + 1 \cdot 8^1 + 2 \cdot 8^0 \\ &= 7 \cdot (8 \cdot 7 + 8) + 1 \cdot (7 + 1) + 2 \\ &= 7 \cdot (8 \cdot 7 + 7 + 1) + 1 \cdot (7 + 1) + 2 \end{aligned}$$

Wenn wir modulo 7 rechnen, fallen alle Vielfachen von 7 (die grauen Terme oben) weg. 458 mod 7 ist 3 und die (oktale!) iterierte Quersumme von 712_8 ist auch 3.

Man muss nur aufpassen, dass man nicht durcheinanderkommt. $7 + 1 + 2$ ist 10_{10} und die *dezimale* Quersumme davon ist 1. Aber wir wollen ja die *oktale* Quersumme von $10_{10} = 12_8$, und die ist 3.

Haben Sie das Prinzip erkannt? Allgemein eignet sich die Quersumme zur Basis n zum Überprüfen der Teilbarkeit durch m, wenn $n - 1$ ein Vielfaches von m ist.

Lösung 72: Ihnen sollte natürlich auffallen, dass es aussieht, als wäre die letzte Ziffer der fünften Potenz einer Zahl a immer identisch mit der letzten Ziffer von a selbst.

Ein Test könnte z.B. so aussehen:

```
def test (a):
    if a % 10 != (a * a * a * a * a) % 10:
        print(a)
a = 1
while a <= 1000000:
    test(a)
    a = a + 1
```

Danach wissen Sie, dass Ihre Vermutung zumindest für eine Million Zahlen stimmt.[9] Aber stimmt sie auch für *alle* Zahlen?

Wenn Ihre Testfunktion so ähnlich aussieht wie meine, dann haben Sie auch schon die Lösung. Man rechnet in $\mathbb{Z}/10\mathbb{Z}$ und muss daher nur zehn Werte überprüfen. Und es stimmt tatsächlich:

a	a^5	$a^5 \bmod 10$
0	0	0
1	1	1
2	32	2
3	243	3
4	1 024	4
5	3 125	5
6	7 776	6
7	16 807	7
8	32 768	8
9	59 049	9

Nebenbei sei übrigens bemerkt, dass wir hier kein tiefes Mysterium entdeckt haben. Dieses zugegebenermaßen interessante Muster ist, wie wir gesehen haben, eine Folge der Dezimaldarstellung und keine Eigenschaft der Zahlen selbst…

Lösung 73: Die Binärdarstellung von 862 929 ist natürlich

$$\texttt{00001101001010101101000 1.}$$

Wenn Sie das nicht sofort und ohne Rechnung gesehen haben, sollten Sie sich das vorherige Kapitel noch mal genau durchlesen!

Lösung 74: 481 und 882 sind `1E1` und `372` im Hexadezimalsystem.[10] Hier die einzelnen Produkte:

$$\begin{aligned} \texttt{E1} \cdot \texttt{72} &= \texttt{6432} \\ \texttt{1} \cdot \texttt{72} &= \texttt{\ \ 72} \\ \texttt{E1} \cdot \texttt{\ 3} &= \texttt{\ 2A3} \\ \texttt{1} \cdot \texttt{\ 3} &= \texttt{\ \ \ 3} \end{aligned}$$

Der Übertrag (wenn es einen gibt) ist jeweils orange dargestellt, das (modulare) Produkt blau. Zum korrekten Addieren müssen wir teilweise noch nach links schieben:

$$\begin{array}{rrr} & \texttt{64} & \texttt{32} \\ & \texttt{72} & \\ \texttt{2} & \texttt{A3} & \\ \texttt{3} & & \\ \hline {\scriptstyle 1} & & \\ \texttt{6} & \texttt{79} & \texttt{32} \end{array}$$

[9] Ersetzen Sie fünf durch eine andere Zahl. Sie werden sehen, dass es dann nicht mehr klappt.

[10] Wir werden in dieser Aufgabe zur Basis 16 rechnen, weil wir dann immer eine 8-Bit-Zahl durch zwei Hexadezimalziffern darstellen können.

Das Produkt ist `67932` im Hexadezimalsystem bzw. 424 242 dezimal.

Lösung 75: Das inverse Element zu null bzgl. der Addition ist null. Es spricht nichts dagegen, dass eine Zahl zu sich selbst invers ist. Wir werden später sehen, dass z.B. in $\mathbb{Z}/6\mathbb{Z}$ die Zahl 3 invers zu 3 ist.

Lösung 76: Nein. In der Gleichung $7 + x = 3$ kommen z.B. nur natürliche Zahlen vor, aber es gibt keine natürliche Zahl, mit der man sie lösen könnte. (Die Lösung ist -4, aber das ist keine natürliche Zahl.)

Lösung 77: Eine Bedeutung des Zeichens ist die eines *binären Operators* (siehe Aufgabe 53) wie in $a - b$. Die zweite Bedeutung ist die des Minuszeichens als Teil der Darstellung einer Zahl. Wenn man z.B. -42 schreibt, meint man damit eine andere Zahl, als wenn man 42 schreibt. Man kann das Minuszeichen nicht einfach weglassen, weil es ein Teil der Zahl ist.[11] Die dritte Bedeutung ist die des Minuszeichens als *unärer* bzw. *einstelliger Operator* wie in $-a$. Das ist die Operation, die das additiv Inverse der Zahl a berechnet. (Oder man könnte es auch als Abkürzung für $0 - a$ betrachten.)

In einem Ausdruck wie -42 könnte man das Minuszeichen also sowohl als unären Operator (angewandt auf die positive Zahl 42) oder als Bestandteil einer negativen Zahl interpretieren. Glücklicherweise wäre in beiden Fällen das *Ergebnis* identisch...

Lösung 78: Wenn Sie Aufgabe 77 bearbeitet haben, war diese einfach. -7 ist eine negative Zahl und $-(-7)$ ist das additiv Inverse dieser Zahl.

Lösung 81: Das einzige kleine Problem hier ist, den Fall $a = 0$ korrekt zu behandeln:

```
def addInv (a, n):
    return a if a == 0 else n - a
```

Lösung 82: 7 teilt 0, weil $0 = 0 \cdot 7$ gilt. *Jede* Zahl teilt offenbar null. Andererseits gibt es keine Möglichkeit, eine Zahl k zu finden, die die Gleichung $7 = k \cdot 0$ erfüllt (weil $k \cdot 0$ immer 0 ist). 0 teilt 7 also *nicht*. Null teilt überhaupt keine Zahl (außer sich selbst).

Lösung 83: Wenn wir mit ganzen Zahlen rechnen, dann können wir manchmal teilen und manchmal nicht. Wenn wir mit Brüchen arbeiten, können wir *immer* teilen (außer natürlich durch null). Daher gibt's da nichts zu untersuchen, es wäre langweilig.

Lösung 84: Wenn der Rest a mod b *nicht* kleiner als b wäre, dann könnten wir b abziehen und einen kleineren, immer noch nichtnegativen, Rest erhalten. Zum Beispiel gilt $25 = 2 \cdot 7 + 11$, aber 11 ist *nicht* 25 mod 7, weil $11 = 7 + 4$ gilt, wir also

$$25 = 2 \cdot 7 + 11 = 2 \cdot 7 + 7 + 4 = 3 \cdot 7 + 4$$

schreiben können. 4 ist der Rest.

Lösung 86: Wenn r der Rest a mod b ist, dann gilt $a = k \cdot b + r$ für eine Zahl k. Aber dann muss auch $a = (-k) \cdot (-b) + r$ gelten, d.h. wir können den Divisor b durch $-b$ ersetzen und erhalten denselben Rest.

[11] Man könnte aber, zur Vermeidung von Mehrdeutigkeit, negative Zahlen anders auszeichnen, z.B. indem man ihnen eine andere Farbe gibt.

Lösung 87: In Python hat `a % b` immer dasselbe Vorzeichen wie `b`. Hingegen hat in Java `a % b` dasselbe Vorzeichen wie `a`. In C darf sich jeder Compiler selbst aussuchen, wie er es gerne hätte. Scheme hält sich an die mathematische Konvention, dass der Rest nie negativ ist. Und schließlich gibt es Sprachen wie Common Lisp oder Julia, die mehr als einen Rest-Operator haben, damit man sich den aussuchen kann, der am besten zur Anwendung passt.

Die ganze Vielfalt findet man auf der entsprechenden Wikipedia-Seite.

Lösung 88: Das kann man in einer Zeile machen:

```
def mathMod(a, b):
    return a % abs(b)
```

Lösung 89: Das ist fast identisch zu Aufgabe 57. Es geht nur darum, zu zeigen, dass es auch mit negativen Zahlen funktioniert.

$$\begin{aligned} -11 \cdot 12 &= (-2 \cdot 7 + 3) \cdot (1 \cdot 7 + 5) \\ &= -2 \cdot 7 \cdot 1 \cdot 7 + 3 \cdot 1 \cdot 7 + (-2) \cdot 7 \cdot 5 + 3 \cdot 5 \\ &= (-2 \cdot 7 \cdot 1 + 3 \cdot 1 + (-2) \cdot 5) \cdot 7 + 3 \cdot 5 = -21 \cdot 7 + 3 \cdot 5 \end{aligned}$$

Lösung 90: Beide sind falsch. Der Abstand von -16 und 11 ist 27, der von 16 und -11 auch. Und 27 ist nicht durch 5 teilbar.

Lösung 91: Ja. Wenn $a \geq b$ gilt, *ist* die Differenz der Abstand. Gilt $a < b$, dann ist die Differenz einfach das additiv Inverse des Abstands. Und wir wissen schon, dass das Vorzeichen bzgl. der Teilbarkeit keine Rolle spielt.

Lösung 92: Wenn man sich die geraden und die ungeraden Potenzen von zehn separat anschaut, so ergibt sich modulo 11 ein einfaches Muster, weil $10^2 = 100 = 9 \cdot 11 + 1$ gilt:

$$\begin{aligned} 10^2 \bmod 11 &= 1 \\ 10^{2n} \bmod 11 &= (10^2)^n \bmod 11 = 1^n \bmod 11 = 1 \\ 10^{2n+1} \bmod 11 &= ((10^2)^n \cdot 10) \bmod 11 = (1^n \cdot 10) \bmod 11 = 10 \end{aligned}$$

Modulo 11 sind also die geraden Zehnerpotenzen kongruent zu 1 und die ungeraden kongruent zu 10 und damit zu -1.

Man kann nun die *alternierende Quersumme* bilden, um Teilbarkeit durch elf zu überprüfen: man geht von rechts nach links durch die Zahl und addiert die erste Ziffer (die zur geraden Potenz 10^0 gehört), subtrahiert die zweite (die zur ungeraden Potenz 10^1 gehört), addiert die dritte, und so weiter.

Für z.B. 157 619 hätte man $9 - 1 + 6 - 7 + 5 - 1 = 11$. Weil 11 durch 11 teilbar ist, gilt das auch für 157 619.[12]

[12] Man kann natürlich auch erneut die alternierende Quersumme von 11 bilden. Dann erhält man die ebenfalls durch elf teilbare Zahl 0.

Lösung 93: Nein, kann es nicht. Gerade Palindrome sind immer durch elf teilbar, wie man sich mithilfe von Aufgabe 92 leicht überlegen kann. Bildet man z.B. die alternierende Quersumme von 329 923, so ergibt sich:

$$3-2+9-9+2-3=(3-3)+(2-2)+(9-9)=0$$

Offenbar taucht jede Ziffer doppelt auf, einmal positiv und einmal negativ, so dass sich immer null ergibt. (Und null ist durch elf teilbar.)

Lösung 95: Um eine negative Zahl a zu konvertieren (wenn man sie konvertieren *kann*, wenn sie also nicht zu klein ist), addiere man einfach 256 und konvertiere wie üblich ins Binärsystem. Für $a=-100$ erhält man z.B. 156 und dann 10011100_2.

Lösung 96: Hier müssen wir 2^{16} statt $256=2^8$ addieren. Das Ergebnis ist $2^{16}-1$, und wir wissen schon, dass das binär die Zahl 1111111111111111_2 ist, sechzehn Einsen. Allgemein wird aus -1 im Zweierkomplement die längstmögliche Folge von Einsen.

Lösung 97: Die größte Zahl ist 01111111_2, also 127_{10}. Die kleinste ist 10000000_2, also -128_{10}.

Lösung 99: Wenn wir zwei Binärzahlen addieren, von denen eine jeweils das Einerkomplement der anderen ist, besteht das Ergebnis aus lauter Einsen, entspricht auf einer 8-Bit-Maschine also der Zahl 255. Mit anderen Worten, wenn wir das Einerkomplement einer Zahl a berechnen, berechnen wir $255-a$. Wenn wir dann 1 addieren, haben wir $256-a$, und das müssen wir nach Aufgabe 95 ja auch ausrechnen.

Und warum können wir den Übertrag ganz links ignorieren? Weil das einfach bedeutet, dass wir innerhalb des „grünen Fensters“ von Seite 49 bleiben. Ein Übertrag würde bedeuten, dass wird das Fenster nach rechts verlassen. Das Ignorieren bedeutet, dass wir stattdessen von links wieder eintreten.

Lösung 100: Der korrekte Wert ist $20!=2\,432\,902\,008\,176\,640\,000$. Die rechten 32 Bits der binären Repräsentation sind diese:

$$10000010101101000000000000000000$$

Weil diese Zahl mit einer Eins anfängt, wird sie als negativ interpretiert. (Überprüfen Sie, dass das wirklich die Zweierkomplementdarstellung von $-2\,102\,132\,736$ ist.)

Lösung 101: Wenn Sie in Python eine negative Zahl nach rechts schieben, werden von links Einsen nachgeschoben. Das soll dafür sorgen, dass negative Zahlen negativ bleiben.[13] Es hat den Effekt, dass so eine Verschiebung *fast* wie eine Division durch eine Zweierpotenz wirkt, aber man muss vorsichtig sein. Wenn Sie positive Zahlen nach rechts schieben, ist es so, als würden die Ergebnisse der Division „zur Null hin“ gerundet. Wenn Sie z.B. wiederholt die 9 schieben, erhalten Sie nacheinander 4, 2, 1 und dann immer wieder null. Machen Sie das mit -9, dann bekommen Sie -5, -3, -2 und dann immer wieder -1.

[13] Man nennt das *arithmetische Verschiebung*. Manche Sprachen bieten auch eine *logische Verschiebung*. Java hat z.B. >> und >>>. Aber warum gibt es nur eine Möglichkeit, nach links zu schieben?

Eine korrekte Beschreibung ist die, dass arithmetisches Verschieben immer wie eine Division durch eine Zweierpotenz wirkt, bei der die Ergebnisse „nach $-\infty$ hin“ gerundet werden (und nicht zur Null hin, wie man vielleicht erwarten würde).

Lösung 102: So zum Beispiel:

```
def naiveGCD (a, b):
    d = a
    if d > b:
        d = b
    while a % d != 0 or b % d != 0:
        d -= 1
    return d
```

Lösung 103: Da d Teiler von a und b ist, können wir Zahlen m und n mit $a = md$ und $b = nd$ finden. Jetzt können wir d ausklammern:

$$\alpha a + \beta b = \alpha md + \beta nd = (\alpha m + \beta n) \cdot d$$

Damit haben wir wieder ein Vielfaches von d.

Lösung 104: Wenn man für α und β jeweils 1 einsetzt, wird aus dem Ausdruck $\alpha a + \beta b$ die Summe $a + b$. Setzt man stattdessen für β den Wert -1 ein, so erhält man die Differenz $a - b$. Die Aussage sagt also ganz allgemein etwas über beliebige α und β aus, damit aber speziell auch etwas über Summe und Differenz (wenn man für α und β entsprechende Werte wählt).

Lösung 105: Wenn man die Gleichung etwas umstellt, dann sieht sie so aus:

$$x!y! - x! - y! = 2 \tag{A.1}$$

Wenn $x \leq y$ gilt, dann sind alle drei Terme auf der linken Seite von (A.1) durch $x!$ teilbar, also ist die ganze linke Seite durch $x!$ teilbar und damit muss auch 2 durch $x!$ teilbar sein. Das geht aber nur, wenn $x! = 1$ oder $x! = 2$ gilt. Setzt man diese beiden Werte in (A.1) ein, so erhält man $y! - 1 - y! = 2$ (also $-1 = 2$) für $x! = 1$ und $2y! - 2 - y! = 2$ (also $y! = 4$) für $x! = 2$. Beides ist nicht möglich.

Wenn hingegen *nicht* $x \leq y$ gilt, so gilt $y < x$ und man kann genau wie eben argumentieren, wenn man die Rollen von x und y vertauscht.

Es gibt also keine ganzen Zahlen, die die Gleichung erfüllen.

Lösung 107: So könnte man's z.B. machen:

```
def gcdMod (a, b):
    while True:
        if b > a:
            a, b = b, a
        a = a % b
        if a == 0:
```

```
            break
    return b
```

Wir setzen hier eine Technik ein, bei der wir (scheinbar) endlos iterieren. Die Bedingung `True` ist nach Definition immer wahr und es sieht daher so aus, als könne die `while`-Schleife niemals beendet werden. PYTHON hat allerdings einen Befehl namens `break`, mit dem sofort die aktuelle Schleife verlassen wird. Diesen Befehl setzen wir hier ein.

`break`

Stellen Sie sicher, dass Sie verstehen, wann genau wir den Algorithmus abbrechen. In der Originalversion hätten wir aufgehört, wenn `a` und `b` gleich gewesen wären. Hier hören wir auf, wenn `b` ein Teiler von `a` ist.

Lösung 109: Die Idee, den Rest zu berechnen statt zu subtrahieren, soll idealerweise mehrere Schritte durch einen ersetzen. Wir sahen so ein Beispiel auf Seite 53, wo wir es mit den beiden Werten 175 und 50 zu tun hatten und 50 von 175 subtrahiert wurde, 50 aber immer noch kleiner als die Differenz 125 war. 50 wurde dann von 125 subtrahiert und war immer noch kleiner als die Differenz, und so weiter. Man konnte „Zeit gewinnen“, indem man 50 gleich so oft wie möglich von 175 abzog, was der Berechnung des Divisionsrestes entspricht.

Im schlimmsten Fall bringt diese „Verbesserung“ gar nichts. Und zwar dann, wenn in *jedem Schritt* die größere der beiden Zahlen kleiner als das Doppelte der kleineren ist. Dann bräuchte man nämlich keinen Rest zu berechnen, sondern könnte gleich die Differenz bilden.

Versuchen wir, ein Beispiel zu konstruieren, bei dem der Algorithmus immer gezwungen ist, zu subtrahieren. Der letzte Schritt wird immer aus zwei gleichen Zahlen bestehen, die dem größten gemeinsamen Teiler entsprechen. Fangen wir etwa mit $(7,7)$ an. Einer von beiden Werten war im vorherigen Schritt die kleinere Zahl, der andere ist die Differenz der beiden vorherigen Zahlen. Da beide Werte aber gleich sind, *muss* das vorherige Paar $(14,7)$ gewesen sein. Wie eben ist nun einer von beiden Werten die kleinere Zahl aus dem Schritt davor, der andere die Differenz. In diesem Fall gibt es aber zwei Möglichkeiten. Die beiden Werte im Schritt vorher waren entweder 7 und $7+14=21$ oder es handelte sich um 14 und $14+7=21$. Aber wenn es das Paar $(7,21)$ gewesen wäre, dann hätten wir 7 mehr als einmal von 21 subtrahieren können. Da wir ein „schlechtes“ Szenario konstruieren wollen, müssen wir uns für das Paar $(14,21)$ entscheiden.

Mit demselben Argument folgt, dass wir es im Schritt davor mit dem Paar $(21,35)$ und *nicht* mit $(14,35)$ zu tun hatten. Und so weiter.[14] Die besonders „ungünstigen“ Fälle für den euklidischen Algorithmus sind also die, die sich als *Fibonacci-Folgen* (siehe Seite 137) darstellen lassen, die mit dem größten gemeinsamen Teiler als erstem und zweitem Wert beginnen.

[14] Wenn wir zwei *verschiedene* Zahlen haben, dann wird ihre Summe immer größer als das Doppelte der kleineren und kleiner als das Doppelte der größeren Zahl sein. Wenn Sie das nicht offensichtlich finden, probieren Sie ein paar Beispiele durch.

Für unser Beispiel mit der 7 würde eine solche Folge so anfangen:

$$7, 7, 14, 21, 35, 56, 91, 147, 238, \ldots$$

Jedes Paar von aufeinanderfolgenden Zahlen hat 7 als größten gemeinsamen Teiler und ist gleichzeitig ein Beispiel für eine Eingabe, bei der der euklidische Algorithmus keine „Abkürzungen“ nehmen kann.

Lösung 111: Hier die verbesserte Version des euklidischen Algorithmus für unser Beispiel mit 400 und 225. (Der wesentliche Unterschied ist die dritte Reihe, in der wir direkt den Rest 25 berechnen, statt drei Mal 50 zu subtrahieren.)

$$\begin{aligned} 400 &= 1 \cdot 225 + 175 \\ 225 &= 1 \cdot 175 + 50 \\ 175 &= 3 \cdot 50 + 25 \\ 50 &= 2 \cdot \mathbf{25} \end{aligned}$$

Nun der „Rückweg“. Man kann das am einfachsten nachvollziehen, wenn man die obigen Gleichungen so umschreibt, dass der Rest jeweils auf einer Seite isoliert ist. (Die letzte Zeile brauchen wir gar nicht mehr.)

$$\begin{aligned} 175 &= 1 \cdot 400 - 1 \cdot 225 \\ 50 &= 1 \cdot 225 - 1 \cdot 175 \\ 25 &= 1 \cdot 175 - 3 \cdot 50 \end{aligned}$$

Der Rest ist jetzt ein Kinderspiel. Fangen Sie mit der letzten Zeile an und ersetzen Sie jeweils eine Zahl gemäß der Zeile darüber:

$$\begin{aligned} 25 &= 1 \cdot 175 - 3 \cdot 50 \\ &= 1 \cdot 175 - 3 \cdot (1 \cdot 225 - 1 \cdot 175) = 4 \cdot 175 - 3 \cdot 225 \\ &= 4 \cdot (1 \cdot 400 - 1 \cdot 225) - 3 \cdot 225 = 4 \cdot 400 - 7 \cdot 225 \end{aligned}$$

Und da haben wir unsere Linearkombination für 25.

Lösung 112: Die Lösung von Aufgabe 111 zeigt bereits, dass es mindestens eine weitere Linearkombination gibt. Aber es gibt eine ganz allgemeine Methode, noch weitere, und zwar beliebig viele, zu finden.

Wir wissen schon: $25 = 9 \cdot 225 - 5 \cdot 400$. Wenn wir die 9 durch $9 + 400 = 409$ ersetzen (wenn wir also $409 \cdot 225 - 5 \cdot 400$ schreiben), haben wir $400 \cdot 225$ addiert. Dann kommt natürlich nicht mehr 25 als Ergebnis raus. Wir können das korrigieren, indem wir den gleichen Betrag wieder abziehen, nämlich $225 \cdot 400$. Das machen wir, indem wir 5 durch $5 + 225 = 230$ ersetzen:

$$25 = 409 \cdot 225 - 230 \cdot 400$$

Und genau dasselbe Spielchen können wir erneut spielen, und so weiter.

Die allgemeine Antwort ist also, dass der erweiterte euklidische Algorithmus uns *eine* mögliche Darstellung des größten gemeinsamen Teilers als Linearkombination liefert, dass es aber immer unendlich viele weitere Darstellungen gibt.

Lösung 113: Sei p eine Primzahl, die das Produkt ab teilt. Nehmen wir außerdem an, dass $p \nmid a$ gilt. (Anderenfalls wären wir ja schon fertig.) Weil p prim ist, muss natürlich $\mathrm{ggT}(a, p) = 1$ gelten und mit dem erweiterten euklidischen Algorithmus können wir Zahlen m und n finden, so dass wir $1 = mp + na$ schreiben können. Wenn wir diese Gleichung mit b multiplizieren, erhalten wir $b = mpb + nab$. mpb ist offenbar durch p teilbar. nab ist auch durch p teilbar, weil ab durch p teilbar ist. Daraus folgt, dass b, als Summe zweier Zahlen, die durch p teilbar sind, auch durch p teilbar ist.

Lösung 114: Das sollte dann ungefähr so aussehen:

```
def convBinToDec (binList):
    result = 0
    for digit in binList:
        result = result*2 + digit
    return result
```

Lösung 115: Wahrscheinlich unsere längste Python-Funktion bisher, aber immer noch vergleichsweise kurz:[15]

```
def extGCD (a, b):
    diffs = []
    while a != b:
        if b > a:
            a, b = b, a
        diffs.append([a, b])
        a = a - b
    d = a
    m = 1
    n = -1
    a, b = diffs.pop()
    for x, y in reversed(diffs):
        if y == a:
            b = x
            m -= n
        else:
            a = x
            n -= m
    return [d, [m, a, n, b]]
```

Die erste Schleife kennen wir schon. Die einzige Änderung ist, dass wir uns die Operanden jeder Subtraktion merken und dafür die Liste `diffs` benutzen.

[15] Warum nur `reversed(diffs)` und nicht `list(reversed(diffs))`? Beides würde funktionieren, aber diese Version ist aus Gründen der Effizienz vorzuziehen. Wir werden den Unterschied verstehen, wenn wir uns mit *Iteratoren* beschäftigt haben.

Die Idee hinter der zweiten Schleife ist, dass zu Beginn jeder Iteration die folgende Bedingung gilt:[16]

```
d = m * a + n * b
```

Am Anfang stimmt das auf jeden Fall, weil `d` ja die letzte Differenz war, die in `diff` gespeichert wurde. Wir können also `a` und `b` entsprechend initialisieren und `m` und `n` auf 1 und -1 setzen.

In der Schleife wandern wir nun rückwärts durch die Differenzen `x - y`. Weil die Differenz entweder der aktuelle Wert von `a` oder der aktuelle Wert von `b` sein muss, muss der *andere* Wert `y` sein. Wir unterscheiden diese beiden Möglichkeiten und aktualisieren die Variablen jeweils entsprechend.

Es folgt ein exemplarischer Durchlauf für `extGCD(12,34)`. `d`, also der größte gemeinsame Teiler, ist 2. Wenn Sie die Spalten für `x` und `y` von unten nach oben lesen, sehen Sie den normalen Ablauf des euklidischen Algorithmus, der $34-12$, $22-12$, und so weiter berechnet.

	x	y	m	a	n	b	m*a	n*b
0	4	2	1	4	-1	2	4	-2
1	6	2	1	6	-2	2	6	-4
2	8	2	1	8	-3	2	8	-6
3	10	2	1	10	-4	2	10	-8
4	12	10	5	10	-4	12	50	-48
5	22	12	5	22	-9	12	110	-108
6	34	12	5	34	-14	12	170	-168

Wenn Sie sich nun die anderen Spalten anschauen, sehen Sie, dass wir am Anfang `m` und `n` als 1 und -1 initialisieren. Anhand der beiden Spalten ganz rechts können Sie jeweils die Schleifeninvariante überprüfen. Die Zeile, die mit 1 anfängt, entspricht der ersten Iteration. `x` und `y` haben die Werte 6 und 2. Als wir diese beiden Zahlen subtrahierten, wurde `y` zu `b` und die Differenz war das neue `a`; daher aktualisieren wir `b`. So geht's im Prinzip immer weiter.

Interessant ist vielleicht noch Zeile 4, weil hier die Rollen von `x` und `y` vertauscht werden. Darum muss hier `a` und nicht `b` aktualisiert werden.

Lösung 116: Der Code ist nun sogar kürzer als in der vorherigen Lösung. Außerdem ist er schneller. [17]

```
def extGCD2 (a, b):
    m_a, n_a = 1, 0
    m_b, n_b = 0, 1
    while a != b:
```

[16] Man nennt das eine *Schleifeninvariante*. Darüber sollten Sie in Ihren Informatik-Vorlesungen etwas gehört haben.

[17] Für „normale" Eingaben wird das allerdings nicht auffallen. Erst bei sehr großen Werten macht sich das bemerkbar, oder wenn man die Funktion sehr oft aufruft.

```
        if b > a:
            a, b = b, a
            m_a, n_a, m_b, n_b = m_b, n_b, m_a, n_a
        a = a - b
        m_a = m_a - m_b
        n_a = n_a - n_b
    return [a, m_a, n_a]
```

(Die lange Zeile mag etwas furchteinflößend wirken, aber dort wechseln nur die Variablen, die auf `_a` enden, mit den Variablen ihre Werte, die auf `_b` enden.)

Die Schleifeninvariante (siehe letzte Aufgabe) ist in diesem Fall, dass zu Beginn jeder Iteration die aktuellen Werte von `a` bzw. `b` als Linearkombination der Funktionswerte ausgedrückt werden können, wenn man die Faktoren `m_a` und `n_a` bzw. `m_b` und `n_b` benutzt. Schauen wir uns ein Beispiel an. Die Funktion wurde als `extGCD2(12, 34)` aufgerufen:

a	b	m_a	n_a	m_b	n_b	12*m_a	34*n_a	12*n_b	34*n_b
12	34	1	0	0	1	12	0	0	34
22	12	−1	1	1	0	−12	34	12	0
10	12	−2	1	1	0	−24	34	12	0
2	10	3	−1	−2	1	36	−34	−24	34
8	2	−5	2	3	−1	−60	68	36	−34
6	2	−8	3	3	−1	−96	102	36	−34
4	2	−11	4	3	−1	−132	136	36	−34

Die Summe der beiden letzten Spalten ist immer der aktuelle Wert von `b` und das ist am Ende der größte gemeinsame Teiler. (Gleichzeitig ergibt die Summe der siebten und achten Spalte jeweils den momentanen Wert von `a`.)

Lösung 117: Der erste Rückgabewert ist der größte gemeinsame Teiler g von a und b. Er wird im Prinzip wie in Aufgabe 107 berechnet. Dafür kann man alle Zeilen, in denen die Variablen `c`, `d`, `f` oder `p` auftauchen, komplett ignorieren.

Die nächsten beiden Werte können verwendet werden, um g als Linearkombination von a und b darzustellen.[18] Das Programm führt also auch den umgekehrten euklidischen Algorithmus aus. Nebenbei gibt es aber als vierten und fünften Wert noch zwei teilerfremde Zahlen aus, so dass das Produkt des vierten Wertes mit a dem Produkt des fünften Wertes mit b entspricht. Dieses Produkt ist das *kleinste gemeinsame Vielfache* von a und b.

Die Vorgehensweise des Programms wird deutlich, wenn man ein paar Befehle hinzufügt. Man sieht dann die Schleifeninvarianten.

[18] Die Reihenfolge stimmt nur für den Fall $a \geq b$.

```
from math import gcd

def foo (a, b):
    if a < b:
        a, b = b, a
    a0, b0 = a, b                    # hinzugefügt
    c0, c = 1, 0
    d0, d = 0, 1
    p = 1
    print(c0*a0 - d0*b0, p*a)        # hinzugefügt
    print(d*b0 - c*a0, p*b)          # hinzugefügt
    print(gcd(c, d))                 # hinzugefügt
    while b != 0:
        f = a // b
        a, b = b, a % b
        d0, d = d, f * d + d0
        c0, c = c, f * c + c0
        p *= -1
        print(c0*a0 - d0*b0, p*a)    # hinzugefügt
        print(d*b0 - c*a0, p*b)      # hinzugefügt
        print(gcd(c,d))              # hinzugefügt
    return a, p * c0, -p * d0, c, d
```

Lösung 118: Nennen wir die Zahl, die wir untersuchen wollen, m. Durch das Abtrennen der letzten Ziffer zerlegen wir diese Zahl in der Form $m = 10a + b$ und berechnen dann $a - 2b$. Ist nun $a - 2b$ durch 7 teilbar, so findet man ein k mit $a - 2b = 7k$. Multiplikation mit 10 liefert $10a - 20b = 70k$ bzw. $m = 10a + b = 70k + 21b$ nach Addition von $21b$ auf beiden Seiten. Rechts steht jetzt eine Linearkombination der Zahlen 70 und 21, die beide durch 7 teilbar sind. Also ist die linke Seite, m, auch durch 7 teilbar.

Ist umgekehrt m durch 7 teilbar, so findet man ein k mit $7k = m = 10a + b$ bzw. $b = 7k - 10a$. Damit folgt $a - 2b = a - 2 \cdot (7k - 10a) = 21a - 14k$ und wir haben $a - 2b$ als Linearkombination zweier Vielfacher von 7 dargestellt.

Lösung 119: Selbst wenn man nur die Packungsgrößen neun und zwanzig zur Verfügung hätte, könnte man ab einer gewissen Größe jede Zahl erreichen. Das liegt daran, dass 9 und 20 teilerfremd sind und es daher mittels des erweiterten euklidischen Algorithmus möglich ist, die Zahl 1 als Linearkombination von 9 und 20 darzustellen, z.B. als $9 \cdot 9 - 4 \cdot 20 = 1$. Wenn man diese Gleichung mit einer anderen Zahl multipliziert, kann man auf der rechten Seite *jeden* gewünschten Wert hinbekommen: Multiplikation mit 38 etwa liefert die Gleichung $342 \cdot 9 - 152 \cdot 20 = 38$.

Allerdings kann man natürlich nicht 342 Neuner-Packungen kaufen und dann 152 Zwanziger-Packungen zurückgeben.[19] Das Minuszeichen vor der 152 ist also ein kleines Problem. Wir wissen aber auch schon, dass man die 1 auch auf unendlich viele

[19] Oder doch? Probieren Sie's aus. Sagen Sie aber nicht, dass der Vorschlag von mir kam!

andere Arten als Linearkombination von 9 und 20 darstellen kann, und wir wissen auch, wie das geht: Indem man nämlich $9 \cdot 20 = 20 \cdot 9$ auf der linken Seite sowohl addiert als auch subtrahiert:

$$\begin{aligned}
342 \cdot 9 - 152 \cdot 20 &= 38 \\
322 \cdot 9 - 143 \cdot 20 &= 38 \\
302 \cdot 9 - 134 \cdot 20 &= 38 \\
282 \cdot 9 - 125 \cdot 20 &= 38 \\
262 \cdot 9 - 116 \cdot 20 &= 38 \\
&\vdots \\
22 \cdot 9 - \quad 8 \cdot 20 &= 38 \\
2 \cdot 9 + \quad 1 \cdot 20 &= 38
\end{aligned}$$

Man könnte also zwei Neuner- und eine Zwanziger-Packung kaufen, um genau 38 Nuggets zu erhalten.

Klappt das immer? Nein! Man kann z.B. wie oben auf $117 \cdot 9 - 52 \cdot 20 = 13$ kommen, aber wenn man durch „Verschieben“ aus -52 eine größere Zahl macht, wird der Faktor vor der 9 negativ, bevor der Faktor vor der 20 positiv werden kann. Man erhält erst $17 \cdot 9 - 7 \cdot 20 = 13$ und dann $-3 \cdot 9 + 2 \cdot 20 = 13$. Daher kann man 13 nicht aus Neuner- und Zwanziger-Packungen kombinieren.[20]

Wenn die Nugget-Zahl x, die Sie erreichen wollen, aber nicht kleiner als $9 \cdot 20 = 180$ ist, dann klappt es: Zunächst ermitteln Sie mit Euklids Hilfe wie oben ganze Zahlen k und m mit $k \cdot 9 + m \cdot 20 = x$. Wenn k und m beide nicht negativ sind, sind Sie fertig. Sollte k negativ sein, dann addieren Sie so lange 20 dazu (und subtrahieren entsprechend oft 9 vom anderen Faktor m), bis Sie auf $k' \cdot 9 + m' \cdot 20 = x$ mit $k' \geq 0$ kommen. Die erste Zahl k', für die das klappt, ist mindestens 0 und höchstens 20. (Sonst hätten Sie einen Schritt übersprungen.) Daher kann m' auch nicht negativ sein, denn $k' \cdot 9$ ist ja höchstens 180. Sollte m negativ sein, können Sie analog vorgehen.

Die größte Zahl, für die man keine Packungskombination finden kann, kann nach dieser Überlegung also maximal 179 sein. Durch Probieren (schreiben Sie z.B. ein kleines PYTHON-Programm dafür) findet man heraus, dass es sich um die Zahl 43 handelt. Damit ist die Aufgabe vollständig gelöst.

Man kann diese Fragestellung leicht verallgemeinern: Auch für mehr als zwei Zahlen kann man deren größten gemeinsamen Teiler berechnen. Handelt es sich um die Zahl eins, so kann man durch (mehrfache) Anwendung des erweiterten euklidischen Algorithmus die Eins als Linearkombination aller Zahlen darstellen, wobei aber evtl. einige Faktoren negativ sind. Wie eben kann man dann argumentieren, dass man jede Zahl ab einer gewissen Größe als Linearkombination schreiben kann, deren Faktoren alle nicht negativ sind. Die Frage ist: Welches ist jeweils die *größte* Zahl, die man *nicht* so schreiben kann? Mit anderen Worten: Gibt es eine Formel, mit der man auf diese Zahl kommen kann?

[20] Und die Sechser-Packung hilft in diesem Fall auch nicht weiter.

Für zwei Zahlen gibt es eine, das ist der sogenannte *Satz von Sylvester*: Die größte Zahl, die sich nicht als Linearkombination mit nichtnegativen Faktoren der teilerfremden positiven Zahlen a und b darstellen lässt, ist $ab - a - b$. (Im obigen Fall für 9 und 20, also *ohne* Sechser-Packungen, wäre das $180 - 9 - 20 = 151$.)

Für mehr als zwei Zahlen sind bisher nur Abschätzungen, Formeln für Spezialfälle und mehr oder weniger effiziente Algorithmen zum Ausprobieren bekannt. Es handelt sich also um ein *ungelöstes Problem* der Mathematik, auch bekannt als *Münzproblem* oder *Frobenius-Problem*, benannt nach dem deutschen Mathematiker Ferdinand Georg Frobenius.

Lösung 121: $a(b+c) = ab + ac$ für alle a, b und c gilt in $\mathbb{Z}$. Das ist das *Distributivitätsgesetz*, das wir aus der Schule kennen. Dann muss das aber auch in $\mathbb{Z}/n\mathbb{Z}$ gelten. (Wir können so eine Gleichung ja in $\mathbb{Z}/n\mathbb{Z}$ aufstellen und die linke und die rechte Seite dann separat in $\mathbb{Z}$ ausrechnen.)

Lösung 122: Für 4 in $\mathbb{Z}/6\mathbb{Z}$ wäre die Gleichung $4a - 6n = 1$. Da 4 und 6 nicht teilerfremd sind, haben sie einen gemeinsamen Teiler, der größer als 1 ist, in diesem speziellen Fall die 2. Aber nach Aufgabe 103 teilt 2 dann auch jeden Ausdruck der Form $4a - 6n$, also kann da nie 1 rauskommen. Mit anderen Worten, es ist unmöglich, die Gleichung zu lösen.

Lösung 124: Division ist Multiplikation mit dem Kehrwert. Um 4 durch 5 in $\mathbb{Z}_7$ zu teilen, brauchen wir also den Kehrwert von 5, und das ist 3. $4/5$ ist dann $4 \cdot 3$, also 5.

Lösung 125: Das Ergebnis wird *immer* null sein. Wenn Sie die Prüfsumme als letzten (zehnten) Buchstaben des Strings eingeben, wird sie mit dem Faktor 10 multipliziert und addiert. Da wir aber in $\mathbb{Z}_{11}$ rechnen, entspricht das dem *Subtrahieren* der Prüfsumme. Für eine gültige ISBN muss also null herauskommen.

Man kann die Funktion daher auch zum Prüfen von ISBN verwenden. Man muss nur darauf achten, dass man das X korrekt behandelt.

```
def checkISBN (digits):
    factor = 1
    sum = 0
    for digit in digits:
        if digit == "X":
            digit = 10
        sum += factor * int(digit)
        factor += 1
    return sum % 11
```

Lösung 126: Hier sehen Sie, was passiert, wenn eine Ziffer falsch ist (in diesem Fall die 2 an der fünften Stelle) und man wie auf Seite 67 die Differenz berechnet:

$$\begin{aligned}&(1\cdot 1 + 2\cdot 4 + 3\cdot 8 + 4\cdot 4 + 5\cdot 2 + 6\cdot 1 + 7\cdot 1 + 8\cdot 7 + 9\cdot 7)\\ -&(1\cdot 1 + 2\cdot 4 + 3\cdot 8 + 4\cdot 4 + 5\cdot 7 + 6\cdot 1 + 7\cdot 1 + 8\cdot 7 + 9\cdot 7)\end{aligned}$$

Die Differenz wäre $5 \cdot (2 - 7)$ und wie beim Vertauschen ist dies offenbar das Produkt zweier von null verschiedener Faktoren und kann daher nicht null sein.

Lösung 127: Auch solche Fehler werden erkannt. Eine Möglichkeit, das zu verstehen, ist, die Prüfsumme mit in die Berechnung aufzunehmen. Das Resultat für eine korrekte ISBN muss dann immer null sein:

$$1 \cdot 1 + 2 \cdot 4 + 3 \cdot 8 + 4 \cdot 4 + 5 \cdot 2 + 6 \cdot 1 + 7 \cdot 1 + 8 \cdot 7 + 9 \cdot 7 - 1 \cdot 4$$

Weil wir in $\mathbb{Z}_{11}$ rechnen, ist -1 dasselbe wie $+10$ (siehe Aufgabe 125):

$$1 \cdot 1 + 2 \cdot 4 + 3 \cdot 8 + 4 \cdot 4 + 5 \cdot 2 + 6 \cdot 1 + 7 \cdot 1 + 8 \cdot 7 + 9 \cdot 7 + 10 \cdot 4$$

Mit dieser Repräsentation kann man nun dieselbe Argumentation verwenden wie beim Vertauschen zweier Ziffern innerhalb der ersten neun Stellen.

Lösung 128: Wir berechnen $5 \cdot 1 + 5 \cdot 2 + 5 \cdot 3 + 5 \cdot 4 = 5 \cdot 10 = 50$, also ist die Prüfsumme 0 in $\mathbb{Z}/10\mathbb{Z}$. Wenn Sie jetzt z.B. die vierte mit der sechsten Ziffer vertauschen, ersetzen Sie $5 \cdot 4 + 0 \cdot 6 = 20$ in der obigen Summe durch $5 \cdot 6 + 0 \cdot 4 = 30$. Aber 20 und 30 sind kongruent modulo 10, also wird sich die Prüfsumme nicht ändern.

Lösung 129: $1 \cdot 1 + 2 \cdot 2 + \cdots + 9 \cdot 9 = 285 = 40 \cdot 7 + 5$, also ist die Prüfsumme 5. Wenn Sie nun z.B. 1 durch 8 ersetzen, erhalten Sie dieselbe Prüfsumme, weil Sie modulo 7 gar nichts geändert haben.

Lösung 130: Es ist einfach, ein unlösbares Kongruenzsystem anzugeben, wenn die Moduln identisch sind:

$$x \equiv 1 \pmod 3$$
$$x \equiv 2 \pmod 3$$

Aber das wäre geschummelt. Hier ist ein Beispiel mit verschiedenen Moduln:

$$x \equiv 1 \pmod 4$$
$$x \equiv 2 \pmod 6$$

Jedes x, das die erste Kongruenz löst, ist offenbar ungerade. Jedes x, das die zweite Kongruenz löst, ist gerade. Daher kann kein x beide Kongruenzen lösen.

Lösung 132: Hier mein Versuch:

```
def chrem(r1, m1, r2, m2, r3, m3):
    k1 = 0
    while k1 < m1:
        s1 = k1 * m2 * m3
        if s1 % m1 == r1:
            break
        k1 += 1
    k2 = 0
    while k2 < m2:
        s2 = k2 * m1 * m3
```

```
        if s2 % m2 == r2:
            break
        k2 += 1
    k3 = 0
    while k3 < m3:
        s3 = k3 * m1 * m2
        if s3 % m3 == r3:
            break
        k3 += 1
    return (s1 + s2 + s3) % (m1 * m2 * m3)
```

Das ist aber nicht gerade elegant, weil wir im Prinzip drei Kopien desselben Codes verwenden. Eine allgemeine Lösung würde besser aussehen, aber auch PYTHON-Funktionalität erfordern, die wir noch nicht besprochen haben.

Beachten Sie bitte außerdem, dass der Code natürlich nur für den Fall gedacht ist, dass die Moduln teilerfremd sind. Anderenfalls werden irgendwelche sinnlosen Werte ausgegeben.

Lösung 133: Nein, ist es nicht. Wenn n nicht prim ist, kann man n als Produkt $a \cdot b$ zweier Zahlen a und b schreiben, die beide größer als 1 und kleiner als n sind. Wären a und b beide größer als $\sqrt{n}$, dann wäre $a \cdot b$ größer als $\sqrt{n} \cdot \sqrt{n} = n$, also kann nur einer dieser beiden Teiler größer als $\sqrt{n}$ sein. Es reicht daher, bei der Suche nach potentiellen Teilern bei der Wurzel aufzuhören:

```
import math

def isPrime (n):
    if n <= 1:
        return False
    for i in range(2, 1 + int(math.sqrt(n))):
        if n % i == 0:
            return False
    return True
```

`sqrt`

Die Funktion `math.sqrt` berechnet die Quadratwurzel einer Zahl. Das Ergebnis müssen wir mit `int` in eine ganze Zahl umwandeln, weil `range` nur ganze Zahlen akzeptiert und Quadratwurzeln im Allgemeinen keine ganzen Zahlen sind.

Verstehen Sie, warum wir 1 addieren müssen? Anderenfalls würde unsere Funktion z.B. 25 für eine Primzahl halten.

Lösung 134: Wenn die Zahl, die wir testen wollen, etwa hundert Dezimalstellen hat, dann wird ihre Quadratwurzel ca. fünfzig Stellen haben. (Siehe Aufgabe 48.) Wenn Sie durch alle ganzen Zahlen bis zu dieser Zahl iterieren, brauchen Sie etwa 10^{50} Schleifendurchläufe.

Bei 10^{15} Iterationen pro Sekunde sind das $10^{50-15} = 10^{35}$ Sekunden. Ist das viel? Oh ja! Nach dem aktuellen Stand der Kosmologie ist unser Universum etwa vierzehn Milliarden Jahre alt, das sind weniger als $5 \cdot 10^{17}$ Sekunden. Wenn Ihr Supercomputer also seit dem Urknall rechnen würde, hätte er bis heute nur etwa 0.00000000000000005 Prozent seines Jobs erledigt...

Lösung 136: Hier ist eine Möglichkeit, die Funktion zu schreiben:

```
def primeFactors (n):
    result = []
    c = 2
    while n > 1:
        while True:
            if n % c == 0:
                result.append(c)
                n //= c
                break
            c += 1
    return result
```

Wenn Sie es so machen, wird die Liste, die Sie bekommen, automatisch sortiert sein. Beachten Sie außerdem, was das Ergebnis bei der Eingabe 1 ist. (Siehe die Anmerkung über das „leere Produkt“ in Fußnote 6 auf Seite 76.)

Die innere Schleife weist der Variablen c aufeinanderfolgende Werte zu. Dazu gehören auch zusammengesetzte Zahlen wie 4 oder 6. Warum enthält die Liste trotzdem nur Primzahlen?

Lösung 137: Man kann das z.B. so machen:

```
from math import sqrt

def findNonUniqueFactorization ():
    n = 5
    while True:
        # first factor found will be stored here
        F = 0
        for k in range(5, 1 + int(sqrt(n)), 4):
            if n % k == 0:
                d = n // k
                if d % 4 == 1:
                    if F == 0:
                        # remember k
                        F = k
                    else:
                        # k != F and F neither divides k nor n/k
```

```
                    if d % F != 0 and k % F != 0:
                        return n
        n += 4
```

Diese Funktion gibt die Quad-Zahl 441 zurück. Und in der Tat kann man 441 sowohl als $9 \cdot 49$ als auch als $21 \cdot 21$ schreiben. 9, 21 und 49 sind Quad-Primzahlen.

Lösung 138: Auf keinen Fall! Sie müssten alle Zahlen mit weniger als hundert Dezimalstellen in einer Liste speichern. Dafür würden Sie deutlich mehr Materie benötigen, als es im ganzen Universum gibt...

Lösung 139: Es wird nichts gedruckt, weil `5` kein Element der Liste ist. Die Liste hat vier Elemente, von denen eines, das dritte, die Liste `[5, 6]` ist. Das ist aber nicht die dasselbe wie die Zahl `5`.

Lösung 140: Wenn ein Element in einer Liste mehrfach auftritt, wie in diesem Fall, so entfernt die Methode das Element nur an der ersten Stelle und nicht an den weiteren.

Lösung 141: Zum Ausprobieren kann man diesen Code verwenden:[21]

```
p = 1
k = 2
while True:
    if isPrime(k):
        p *= k
        print(k, end="")
        if not isPrime(p + 1):
            print(" + 1 is not prime.")
            break
        print(" * ", end="")
    k += 1
```

Wir sehen, dass die Zahl $2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 + 1 = 30031$ nicht prim ist. (Man kann sie als $59 \cdot 509$ darstellen.) Der entscheidende Fakt aber ist, dass sie nicht durch 2, 3, 5, 7, 11 oder 13 teilbar ist.

Lösung 142: Dies sollte mit akzeptabler Geschwindigkeit ausgeführt werden:

```
[primepi(10 ** n) for n in range(1, 8)]
```

Wenn Sie allerdings 8 durch 9, 10 oder 11 ersetzen, kommen Sie evtl. schon an den Punkt, an dem Sie Ihre CPU ganz schön auslasten.

Lösung 143: So würde man die tausendste Primzahl mit dem Schätzwert vergleichen:

[21] `print` gibt normalerweise einen Zeilenumbruch am Ende aus, so dass jede Ausgabe in einer eigenen Zeile steht. Man kann das vermeiden, indem man wie hier explizit angibt, was am Ende der Ausgabe ausgegeben werden soll.

```
from sympy import prime
from math import log

prime(1000), 1000 * log(1000)
```

Und hier sind ein paar relative Fehler:

```
def relPNTError (n):
    p = prime(n)
    a = n * log(n)
    return abs(a - p) / p

[relPNTError(10 ** n) for n in range(1, 7)]
```

Man sieht, dass der Fehler langsam von etwa zwanzig auf etwa zehn Prozent sinkt, während wir uns der millionsten Primzahl nähern.

Lösung 144: So sieht man, dass die ersten vierzig Werte, $g(0)$ bis $g(39)$, Primzahlen sind, während $g(40)$ nicht prim ist:

```
def g (n):
    return n * n + n + 41

n = 0
while True:
    if not isPrime(g(n)):
        print(n, g(n))
        break
    n += 1
```

Aber das hätte man sich auch ohne Computer überlegen können. Nach der binomischen Formel erhält man:

$$g(40) = 40^2 + 40 + 41 = 40^2 + 2 \cdot 40 \cdot 1 + 1^2 = (40+1)^2 = 41 \cdot 41$$

Und noch einfacher sieht man, dass $g(41)$ nicht prim ist:

$$g(41) = 41^2 + 41 + 41 = (41 + 1 + 1) \cdot 41 = 43 \cdot 41$$

(Auf ähnliche Art und Weise kann man recht einfach zeigen, dass *kein* nicht-konstantes Polynom mit ganzzahligen Koeffizienten ausschließlich Primzahlen als Werte haben kann, wenn man ganze Zahlen einsetzt.)

Lösung 145: Wenn m das folgende Produkt ist,

$$m = 2 \cdot 3 \cdot 4 \cdot \cdots \cdot (n-1) \cdot n \cdot (n+1),$$

dann ist m offenbar durch 2 teilbar und $m+2$ daher auch; $m+2$ ist also keine Primzahl. Ebenso sind m und $m+3$ durch 3 teilbar, $m+3$ ist also auch keine Primzahl.

$m + 4$ ist ebenfalls nicht prim, und so weiter, bis wir bei $m + (n + 1)$ (auch keine Primzahl) ankommen. Das ist eine Folge von n zusammengesetzten Zahlen.

Lösung 146: Ich hab's so gemacht:

```
from sympy import isprime

def nextPrimePair (n):
    if n % 2 == 0:
        n += 1
    while True:
        if isprime(n) and isprime(n + 2):
            return n, n+2
        n += 2
```

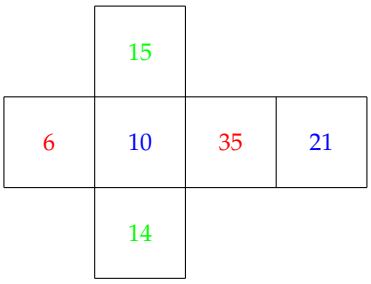
(Beachten Sie, wie diese Funktion zwei Werte zurückgibt.)

Lösung 147: Wenn es so einen Drilling gäbe, müsste er so aussehen:

Die roten Punkte sind dabei die ungerade Primzahlen, die blauen Punkte also gerade Zahlen, die zwischen ihnen bzw. um sie herum liegen. Wenn man aber drei aufeinderfolgende gerade Zahlen hat, dann muss eine von denen offenbar durch 6 teilbar sein. Mit anderen Worten, mindestens einer der blauen Punkte repräsentiert eine Zahl, die durch 3 teilbar ist. Wenn man von dort aus drei Schritte nach links oder rechts geht, trifft man erneut eine Zahl, die durch 3 teilbar ist. Aber eine von denen ist ein roter Punkt. Und die einzige Primzahl, die durch 3 teilbar ist, ist die Drei selbst.

Es gibt also keine anderen Primzahldrillinge.

Lösung 148: Eine mögliche Lösung (und sogar die optimale) könnte so aussehen:

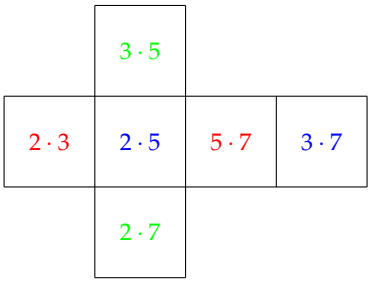


Dabei haben gegenüberliegende Seiten dieselbe Farbe.

Nun zur Begründung: Wir greifen uns eine der sechs Seiten heraus. Damit ihre Zahl nicht teilerfremd mit einem Nachbarn ist, muss sie mindestens einen Primteiler haben, den der Nachbar auch hat. Da zwei der Nachbarn sich gegenüberliegen, muss unsere Zahl mindestens zwei Primteiler haben, denn die sich gegenüberliegenden Nachbarn dürfen nicht denselben Primteiler mit ihr gemeinsam haben. Dieses Argument gilt für jede der sechs Zahlen, also muss jede mindestens zwei verschiedene Primteiler haben.

Gegenüberliegende Zahlen müssen teilerfremd sein. Da jede von ihnen mindestens zwei Primteiler haben muss, braucht man also insgesamt mindestens vier verschiedene Primteiler schon für zwei Seiten des Würfels.

Wenn man mit insgesamt vier Primfaktoren und zwei Primteilern pro Zahl auskommt und dann auch noch die vier kleinsten Primzahlen wählt, hat man also eine minimale Lösung gefunden.



Lösung 149: Mit *list comprehesion* kann man das in einer Zeile machen:

```
[[a**(p-1) % p for a in range(p)] for p in [2,3,5,7,11]]
```

Lösung 150: p muss nicht unbedingt prim sein. Das einfachste Gegenbeispiel ist trivial: nehmen Sie $a = 1$, dann kann p irgendeine zusammengesetzte Zahl sein. Aber selbst wenn wir diesen Fall ausschließen, finden wir genügend Beispiele. So ist etwa $8^{9-1} \bmod 9 = 1$, obwohl 9 keine Primzahl ist.

Lösung 152: Wenn 3^{561-1} modulo 561 den Wert 1 hätte, dann wäre 3^{561-2} der Kehrwert von 3 in $\mathbb{Z}/561\mathbb{Z}$. Weil aber 3 und 561 nicht teilerfremd sind, kann 3 keinen Kehrwert haben. Siehe Aufgabe 122.

Lösung 153: Der folgende Code sollte nichts ausgeben:

```
from sympy import isprime

for p in range(2, 100):
    if isprime(p):
        for a in range(2, p-1):
            if a * a % p == 1:
                print("Whoops!")
```

Beachten Sie, dass $p - 1$ in $\mathbb{Z}_p$ der Zahl -1 entspricht.

Lösung 157: Hier ist ein kleiner Vergleichstest:

```
def foo (n):
    s = 1
    for i in range(n):
```

```
        s = (s + i**10000) % 97
    return s

def bar (n):
    s = 1
    for i in range(n):
        s = (s + pow(i, 10000, 97)) % 97
    return s
```

Beide Funktionen liefern dieselben Ergebnisse. Auf meinem Computer ist `bar` aber mehr als 700mal so schnell wie `foo`! (Das kann man z.B. mit `%timeit` messen. Siehe Kapitel 39.)

Lösung 158: Wenn Sie es möglichst kurz haben wollen, dann wohl so:

```
def caesar (S, K):
    return [chr((ord(c)-ord('A')+K) % 26 + ord('A')) for c in S]
```

Diese Funktion arbeitet allerdings wirklich nur mit den Großbuchstaben von `A` bis `Z`. Vielleicht machen Sie das ja noch ein bisschen schicker. Und wenn es Sie stört, dass das Ergebnis eine Liste und kein String ist, dann können Sie das folgendermaßen ändern:

```
def caesar (S, K):
    return "".join([chr((ord(c)-ord('A')+K)%26+ord('A')) for c in S])
```

Zum Entschlüsseln kann man dann dieselbe Funktion mit dem Schlüssel $26 - K$ verwenden.

Lösung 159: Abgesehen von den bereits erwähnten Argumenten gegen symmetrische Verfahren gibt es nur 26 verschiedene Schlüssel. Man muss den Schlüssel nicht mal kennen, man kann einfach alle möglichen Schlüssel auf einem Computer durchprobieren! Moderne Verfahren sind daher so konzipiert, dass es viel mehr Auswahl gibt. Wenn Sie z.B. 256 Bits für den Schlüssel zur Verfügung haben, dann gibt es für ihn 2^{256} Möglichkeiten. Das sind fast so viele, wie es Atome im Universum gibt! Da hilft der schnellste Computer nicht.

Außerdem sind alle Verfahren, die einfach Buchstaben durch andere ersetzen, immer sehr leicht zu knacken, weil man ihnen mit statistischen Methoden zu Leibe rücken kann. So ist z.B. e der häufigste Buchstabe in deutschen Texten, dann kommt n und so weiter. Sie müssten also einfach die Zeichen in der „verschlüsselten“ Nachricht (wenn sie lang genug ist) nach Häufigkeit sortieren und hätten eine Übersetzungstabelle.

Lösung 160: Man benutzt dafür den erweiterten euklidischen Algorithmus.

Lösung 162: Kennt man p und q, so kennt man sofort $\varphi(n) = (p-1)(q-1)$. Da e öffentlich ist, kann man nun d als Kehrwert von e berechnen.

Lösung 163: Sie kann 42^2 verschicken und als „verschlüsseltes“ Pendant einfach das Quadrat der von Bob erhaltenen verschlüsselten Nachricht:

```
S = crypt(42, e)         # von Bob erhalten
42*42, crypt(S*S, d)     # von Alice verschickt
```

Warum das funktioniert, sollte aus der obigen Herleitung hervorgehen. Sie sehen: Das ist alles ganz schön kompliziert...

Lösung 164: Mathematiker verwenden typischerweise einen horizontalen oder einen schrägen Strich (und nicht : oder ÷) als Divisionszeichen. Aber wenn man dann $2/5$ schreibt, kann das als eine bestimmte Zahl (die man z.B. auch als $4/10$ oder 0.4 hätte schreiben können) verstanden werden, aber auch als Rechenoperation. Im ersten Fall ist der Strich der Bruchstrich, der zur Zahl dazugehört (wie das Minuszeichen zu den negativen Zahlen); im zweiten Fall ist der Strich ein Rechenzeichen (wie $+$). „Glücklicherweise“ stimmt das Ergebnis der zweiten Interpretation mit der ersten Interpretation überein.

Lösung 165: Man könnte das in einer Zeile machen, aber so ist es selbsterklärend:

```
from math import gcd

def lowestTerms (frac):
    num = frac[0]
    den = frac[1]
    g = gcd(num, den)
    return [num // g, den // g]
```

Lösung 166: Ich habe mich entschieden, dass der Nenner immer positiv sein soll. Das Vorzeichen des Bruchs entspricht dann dem Vorzeichen des Zählers:

```
def lowestTerms (frac):
    num = frac[0]
    den = frac[1]
    if den < 0:
        num *= -1
        den *= -1
    g = gcd(num, den)
    return [num // g, den // g]
```

Lösung 167: Sie haben hoffentlich nicht vergessen, das Ergebnis zu kürzen:

```
def add (frac1, frac2):
    num1 = frac1[0]
    den1 = frac1[1]
    num2 = frac2[0]
```

```
    den2 = frac2[1]
    return lowestTerms([num1 * den2 + num2 * den1,
                        den1 * den2])

def mult (frac1, frac2):
    num1 = frac1[0]
    den1 = frac1[1]
    num2 = frac2[0]
    den2 = frac2[1]
    return lowestTerms([num1 * num2, den1 * den2])
```

Lösung 168: Erinnern Sie sich an das, was wir auf Seite 43 über inverse Operationen gesagt haben:

```
def sub (frac1, frac2):
    return add(frac1, [-frac2[0], frac2[1]])

def div (frac1, frac2):
    return mult(frac1, [frac2[1], frac2[0]])
```

Lösung 169: Das sollte für Sie ein Kinderspiel sein. Anderenfalls greifen Sie sich bitte ein Schulbuch und wiederholen Sie diesen Stoff.

$1/2 = 0.5$, $4/25 = 0.16$, $13/8 = 1.625$. Man kann diese Brüche alle so erweitern, dass der Nenner eine Zehnerpotenz wird. Zum Beispiel:

$$\frac{13}{8} = \frac{13 \cdot 125}{8 \cdot 125} = \frac{1625}{1000}$$

Jetzt kann man die Dezimaldarstellung sofort „sehen“.

$0.34 = 17/50$, $1.3 = 13/10$ und $0.012 = 3/250$. Hier kann man die Zahlen erst als Brüche mit Zehnerpotenzen im Nenner schreiben (man zähle einfach die Stellen nach dem Komma) und dann kürzen. Zum Beispiel:

$$0.012 = \frac{12}{1000} = \frac{3 \cdot 4}{250 \cdot 4} = \frac{3}{250}$$

Lösung 170: Um eine Zahl wie $5/6$ in die Dezimaldarstellung zu überführen, wenden Sie die schriftliche Division an[22] und stellen sich vor, dass der Zähler, falls nötig, hinter dem Komma einen unbegrenzten Vorrat an Nullen zur Verfügung hat:

$$\begin{array}{rl} & 5.000\ldots : 6 = 0.833\ldots \\ -\ & \underline{4.8} \\ & 20 \\ -\ & \underline{18} \\ & 20 \\ -\ & 18 \end{array}$$

[22] Das hätte bei der vorherigen Aufgabe übrigens auch geklappt.

$$\overline{}$$
$$2$$

Irgendwann werden Sie sehen, dass der Prozess sich zu wiederholen beginnt. Dann können sie aufhören und die Periode entsprechend markieren. In diesem Fall ist die Antwort $0.8\overline{3}$.

Für $3/7$ dauert es etwas länger:

$$\begin{array}{rl}
 & 3.0000000\ldots : 7 = 0.4285714\ldots \\
-\ & \underline{2.8} \\
 & 20 \\
-\ & \underline{14} \\
 & 60 \\
-\ & \underline{56} \\
 & 40 \\
-\ & \underline{35} \\
 & 50 \\
-\ & \underline{49} \\
 & 10 \\
-\ & \underline{7} \\
 & 30 \\
-\ & \underline{28} \\
 & 2
\end{array}$$

Das Resultat ist $0.\overline{428571}$. Aber wichtiger ist, dass die roten Zahlen, die Sie oben sehen, alle *Divisionsreste* von der Division durch 7 sind. Und es gibt nur sieben mögliche Reste. Entweder kommt irgendwann mal der Rest null, dann ist die Division aufgegangen und der Prozess beendet. (Das wäre in der vorherigen Aufgabe passiert.) Oder man bekommt ständig andere Reste. Aber dann muss der Prozess sich zwangsläufig wiederholen, weil es ja nur endlich viele mögliche Reste gibt! In diesem Beispiel (wenn der Nenner 7 ist) muss nach spätestens sechs Schritten eine Periode erkennbar sein.

Lösung 171: Wie wir in Aufgabe 169 gesehen haben, kommen wir mit endlich vielen Stellen aus, wenn der Bruch so erweitert werden kann, dass der Nenner eine Zehnerpotenz wird. Das kann man dann (und nur dann) machen, wenn die einzigen Primteiler des Nenners 2 und 5 sind, also die Primteiler von 10. (Dabei gehen wir hier natürlich von der *gekürzten* Darstellung des Bruchs aus.)

Lösung 172: Die Resultate sind $3/11$ und $251/1100$.

Lösung 173: 2, 3 und 6: $1/2 + 1/3 + 1/6 = 1$.

Lösung 174: Ist eine natürliche Zahl n größer als drei, so ist $1/n$ kleiner als $1/3$. Daher können die gesuchten drei Zahlen nicht alle größer als drei sein, denn sonst wäre die Summe ihrer Kehrwerte kleiner als $3 \cdot 1/3 = 1$. Eine der drei Zahlen, nennen wir sie n_1, muss also einen der Werte zwei oder drei haben.

Gilt $n_1 = 2$, so muss die Summe der Kehrwerte der anderen beiden Zahlen $1/2$ sein. Wie eben folgt daraus, dass diese beiden Zahlen nicht beide größer als 4 sein können.

Eine der beiden, nennen wir sie n_2, muss also drei oder vier sein. Probieren zeigt, dass es mit vier nicht klappt, mit drei aber schon. Das liefert die uns schon bekannte Lösung.

Es bleibt noch die Möglichkeit, dass n_1 den Wert 3 hat. Dann müssen die beiden anderen Kehrwerte zusammen aber sogar noch eine größere Summe als $1/2$ haben, nämlich $2/3$. Einer von beiden darf daher nicht größer als drei sein. Für diesen Wert bleibt nur die Möglichkeit zwei, womit man auf dieselbe Lösung wie eben kommt.

Lösung 175: Im Text, der auf die Aufgabe folgt, finden Sie Beispiele.

Lösung 176: Wir haben zehn Ziffern zur Verfügung und für jede Ziffer zehn Möglichkeiten. Damit erhalten wir 10^{10}, also zehn Milliarden verschiedene Zahlen. Das sind zwar scheinbar sehr viele, aber der so darstellbare Zahlenbereich ist offenbar für die meisten Anwendungen nicht geeignet.

Lösung 177: Die besten Näherungen für die ersten beiden Zahlen sind `0.3333333` und `0.6666667`. Beachten Sie die `7` am Ende der zweiten Zahl. Für die dritte Zahl könnte man `0.0000000` *oder* `0.0000001` nehmen, beide Näherungen haben denselben absoluten Fehler.

Lösung 179: Man hätte 5 000 auch so darstellen können:

```
+  0500000000  +04
```

Oder so:

```
+  0005000000  +06
```

Offensichtlich gibt es noch weitere Möglichkeiten.

Lösung 180: Die fünf Zahlen würden so dargestellt werden:

```
+  5000000000  -01
-  1666666667  -01
+  7000000000  -04
+  4200000000  +01
-  1234500000  +04
```

Lösung 181: Da die Mantisse nicht mit null anfangen darf, gibt es $9 \cdot 10^9$ verschiedene Mantissen. Zu jeder dieser Mantissen gibt es zwei verschiedene Vorzeichen und 199 verschiedene Exponenten. Damit kommt man auf $2 \cdot 199 \cdot 9 \cdot 10^9$ verschiedene Zahlen. Und die Null darf man natürlich nicht vergessen. Das ergibt 3 582 000 000 001 darstellbare rationale Zahlen, also mehr als drei Billionen.

Lösung 182: Die größte und die kleinste positive Zahl würden intern so repräsentiert werden:

```
+  9999999999  +99
+  1000000000  -99
```

Das sind die Zahlen $9.999999999 \cdot 10^{99}$, also *fast* 10^{100}, und 10^{-99}.

Lösung 183: Zwischen 0.1 und 0.2 liegen in der Festkomma-Darstellung die Zahlen 0.1000001, 0.1000002 und so weiter bis 0.1999999. Das sind insgesamt 999 999 Stück, also eine weniger als eine Million. Zwischen 100.1 und 100.2 liegen genauso viele.

In der Fließkomma-Darstellung sieht der Bereich zwischen 0.1 und 0.2 so aus:

```
+  1000000001  -01
+  1000000002  -01
        ⋮
+  1999999999  -01
```

Das sind 999 999 999 Zahlen, eine weniger als eine Milliarde.

Zwischen 100.1 und 100.2 sieht es aber so aus:

```
+  1001000001  +02
+  1001000002  +02
        ⋮
+  1001999999  +02
```

Das sind 999 999 Zahlen, also nur knapp eine Million.

Lösung 184: $1/3 + 1/3 + 1/3$ wird auch in Fließkomma-Arithmetik nicht 1 ergeben, $10^{-6}/100 \cdot 100$ aber schon.

Lösung 185: Denken Sie daran, diesen Lösungshinweis mit PYTHON zu überprüfen:

- Die korrekte Summe in der ersten Klammer ist 10 000 000 042. Das ist aber keine Maschinenzahl und muss durch

  ```
  +  1000000004  +10
  ```

  angenähert werden. Die gleiche Maschinenzahl ergibt sich in der zweiten Klammer. Die Differenz ist dann natürlich null. Hätte man stattdessen in dieser Reihenfolge

  $$(10\,000\,000\,000 - 10\,000\,000\,000) + (42 - 41)$$

  ausgewertet, wäre das Ergebnis auch mit Maschinenzahlen korrekt gewesen.
- Die korrekte Summe der ersten beiden Terme ist 1.0000000001. Das ist aber keine Maschinenzahl und muss durch

  ```
  +  1000000000  +00
  ```

  approximiert werden, also durch eins, was dem ersten Summanden entspricht. Es ist also so, als hätte gar keine Addition stattgefunden. Die nächsten Additionen verlaufen ebenso. Hätte man stattdessen so

  $$10^{-10} + 10^{-10} + 10^{-10} + 10^{-10} + 10^{-10} + 10^{-10} + 1$$

  gerechnet, wäre zumindest die beste Näherung herausgekommen.

– Das korrekte Produkt der ersten beiden Terme ist 10^{-140}. Das ist keine Maschinenzahl und die beste Näherung dafür ist null. Daher ergibt sich dann auch als Gesamtprodukt null. Hätte man stattdessen so

$$10^{-70} \cdot 10^{70} \cdot 10^{-70}$$

gerechnet, wäre das korrekte Resultat herausgekommen.

Lösung 186: Die folgenden Gleichungen gelten alle nicht mehr, wenn man statt mit rationalen Zahlen mit Maschinenzahlen gemäß unseres Modells arbeitet:

$$\begin{aligned}(1 + 4 \cdot 10^{-10}) + 4 \cdot 10^{-10} &= 1 + (4 \cdot 10^{-10} + 4 \cdot 10^{-10}) \\ (10^{-70} \cdot 10^{-70}) \cdot 10^{70} &= 10^{-70} \cdot (10^{-70} \cdot 10^{70}) \\ 10^{-98} \cdot (0.04 + 0.04) &= 10^{-98} \cdot 0.04 + 10^{-98} \cdot 0.04\end{aligned}$$

Falls Sie nicht selbst Beispiele gefunden haben, sollten Sie die obigen Angaben natürlich überprüfen!

Lösung 188: Man geht wie im Dezimalsystem vor. Allerdings werden die Neunen im Dezimalsystem zu Einsen im Binärsystem. Und natürlich sind Zähler und Nenner auch Binärzahlen.

$0.\overline{01}$ wird (binär) zu $1/11$ und das ist dezimal $\frac{1}{3}$.

$1.10\overline{101}$ wird (binär) zu diesem Bruch:

$$1/100 \cdot (110 + 0.\overline{101}) = 1/100 \cdot (110 + 101/111)$$

Dezimal ist das $1/4 \cdot (6 + 5/7) = 47/28$.

Lösung 189: Wie im Dezimalsystem hängt das von den Teilern des Nenners ab. Eine abbrechende (endliche) Darstellung erhält man binär genau dann, wenn der Nenner des gekürzten Bruchs eine Zweierpotenz ist.

Lösung 191: Man könnte es z.B. so machen:

```
def initialBinPlaces (x, n):
    result = []
    while len(result) < n:
        x *= 2
        if x >= 1:
            result.append(1)
            x -= 1
        else:
            result.append(0)
    return result
```

Beachten Sie, dass diese Funktion nur für positive Zahlen, die kleiner als eins sind, ausgelegt ist.

Lösung 194: Die Antwort wird im nachfolgenden Text gegeben.

Lösung 195: Die Zahl 2 ist binär 10_2 bzw. $1.0_2 \cdot 2^1$. Da die führende Eins weggelassen wird, besteht die Mantisse aus 52 Nullen (und der Exponent stellt die Zahl 1 dar). 1 und 0.5 sind binär $1.0_2 \cdot 2^0$ und $1.0_2 \cdot 2^{-1}$. Die Mantissen sind also in allen drei Fällen identisch, nur die Exponenten unterscheiden sich.

(Falls Sie sich jetzt fragen, wie die Null dargestellt wird: Sie lässt sich nicht in dieser normalisierten Form darstellen. Dafür wird einer der beiden erwähnten speziellen Exponenten verwendet. Siehe die Tabelle auf Seite 129.)

Lösung 196: Für die Mantisse gibt es 2^{52} verschiedene Möglichkeiten. Ferner gibt es 2046 verschiedene Exponenten und zwei verschiedene Vorzeichen. Durch die Normalisierung ergibt jede Kombination aus Mantisse, Vorzeichen und Exponent auch tatsächlich eine andere Zahl. Das macht insgesamt

$$2 \cdot 2^{52} \cdot 2046 = 18\,428\,729\,675\,200\,069\,632$$

verschiedene Zahlen, also knapp 20 Trillionen.

Lösung 201: Das war eine Fangfrage. Diese Schleife wird nie enden. Das wurde im Absatz vor dieser Aufgabe ja gerade erklärt.

Abfragen wie die in dieser Aufgabe oder in den beiden Beispielen davor gehören zu den typischen Fehlern, die Anfänger machen. Sie sollten, wenn Sie mit Fließkommazahlen arbeiten, niemals testen, ob ein bestimmter Wert *exakt* erreicht wird, weil das häufig gar nicht möglich ist.

Stattdessen hätte man die Schleife so schreiben können:

```
c = 0
x = 0.0
while x < 2.0:                  # <--
    x += 0.1
    c += 1
```

Alternativ kann man statt auf Gleichheit auch immer darauf testen, ob die Abweichung vom erwarteten Wert unterhalb einer gewissen Toleranzschwelle liegt:

```
c = 0
x = 0.0
while abs(x - 2.0) > 1E-5:      # <--
    x += 0.1
    c += 1
```

Dafür eignet sich auch die im Text erwähnte Funktion `isclose`:

`isclose`

```
from math import isclose

c = 0
x = 0.0
```

```
while not isclose(x, 2.0):     # <--
    x += 0.1
    c += 1
```

Lösung 202: Auch diese Schleife wird nie enden. Da man (inklusive *hidden bit*) nur 53 Bits für die Mantisse zur Verfügung hat, können nur bis 2^{53} alle ganzen Zahlen ohne Informationsverlust als Fließkommazahlen repräsentiert werden. Danach „fehlt" mindestens ein Bit. Der Wert von `b + 1` ist in der Fließkomma-Arithmetik derselbe wie der von `b`. Erst `b + 2` hat einen anderen Wert.

Hätte man die Zeile `a *= 1.0`, die aus dem Wert von `a` eine Fließkommazahl macht, weggelassen, dann hätte es nur einen Schleifendurchlauf gegeben. Die Moral von der Geschichte: Wenn Sie mit ganzen Zahlen arbeiten, dann stellen Sie diese nicht durch Fließkommazahlen dar.

Lösung 203: Wenn Sie wissen, dass `a == 0.0` gilt und wissen wollen, ob `a` positiv oder negativ ist, können Sie in C `1.0 / a` berechnen und unterscheiden, ob das Ergebnis `inf` oder `-inf` ist. In PYTHON geht das aber nicht. Sie können jedoch die Funktion `copysign` verwenden:

`copysign`

```
import math
math.copysign(42, 0.0), math.copysign(42, -0.0)
```

Lösung 204: Die kleinste normalisierte Zahl steht in der Tabelle direkt vor der Aufgabe. Man erhält sie, wenn die Mantisse (ohne *hidden bit*) nur aus Nullen besteht und der Exponent minimal ist, und sie entspricht der Zahl 2^{-1022}, also ungefähr $2.2 \cdot 10^{-308}$.

Der folgende Code zeigt, wie man der Null noch näher kommen kann. Alle ausgedruckten Zahlen außer der ersten sind denormalisiert:

```
a = 2 ** -1022
for i in range(54):
    print(a)
    a = a / 2
```

Man sieht allerdings auch, wie PYTHON nach und nach immer weniger Ziffern ausgibt. Die normalisierten Zahlen werden umso ungenauer, je kleiner sie werden.

Lösung 205: Mit kaufmännischem Runden ist die Summe

$$2 + 3 + \cdots + 10 = 54.$$

Mit mathematischem Runden ist die Summe

$$2 + 2 + 4 + 4 + 8 + 8 + 10 = 50.$$

Die *richtige* Summe (ohne Runden) ist 49.5.

Lösung 206: Man kann immer die letzte Ziffer, die nicht null ist, durch eine um eins kleinere ersetzen und dahinter unendlich viele Neunen hängen. Statt 1.23 kann man

also zum Beispiel auch $1.22\overline{9}$ schreiben. Jede Dezimalzahl mit endlich vielen Nachkommastellen lässt sich also auf zwei Arten schreiben.

(Falls Sie das nicht sofort einsehen, rechnen Sie eine Zahl wie $1.22\overline{9}$ in einen Bruch um, so wie wir es im Kapitel über rationale Zahlen gelernt haben.)

Lösung 207: Wenn x die Gleichung $x^2 = 2$ löst, dann löst $-x$ sie auch, denn $(-x)^2$ ist ja dasselbe wie x^2. Daher kann man davon ausgehen, dass, wenn es überhaupt einen Bruch gibt, der $x^2 = 2$ löst, es auch einen *positiven* Bruch gibt, der dies tut.

Lösung 208: Wie im Hinweis schon erwähnt, kann man den Beweis für 2 fast wortwörtlich abschreiben. Wir gehen zunächst davon aus, dass es teilerfremde positive Zahlen a und b mit $(a/b)^2 = 3$ gibt. Das führt auf $a^2 = 3b^2$ und daher muss a nach Aufgabe 113 durch 3 teilbar, also von der Form $a = 3m$, sein. Man kann dann $9m^2 = 3b^2$ schreiben und erhält gekürzt $3m^2 = b^2$. Nun folgt, dass b auch durch 3 teilbar sein muss. Dann können a und b aber nicht teilerfremd sein.

Lösung 209: Man kann den obigen Beweis bis zur Stelle $a^2 = 4b^2$ „abschreiben“. Man kann nun aber nicht folgern, dass a durch 4 teilbar ist. Das gibt Aufgabe 113 nicht her, weil es dort ja um *Prim*teiler geht.

Lösung 210: Die Gleichung $x^2 = c$ ist offenbar dann durch eine rationale Zahl lösbar, wenn in der eindeutigen Zerlegung in Primfaktoren von c jeder Primfaktor in gerader Potenz vorkommt. Das gilt z.B. für $c = 4 = 2^2$ oder $c = 36 = 2^2 \cdot 3^2$ oder $c = 40\,000 = 2^6 \cdot 5^4$. Anders ausgedrückt: es klappt, wenn c eine *Quadratzahl* ist.

Anderenfalls ist die Gleichung *nie* durch einen Bruch lösbar und man kann immer den gleichen Beweis in leicht abgewandelter Form benutzen.

Lösung 211: Wir nehmen an, man könne positive ganze Zahlen a und b mit $\log_2 3 = a/b$ finden. Dann müsste $3 = 2^{\log_2 3} = 2^{a/b}$ gelten. Potenziert man beide Seiten mit b, so erhält man $3^b = 2^a$. Dann steht aber links vom Gleichheitszeichen eine ungerade Zahl und rechts eine gerade. Das kann natürlich nicht sein.

Dies dürfte wohl einer der einfachsten Beweise dafür sein, dass eine Zahl nicht rational ist. Beachten Sie allerdings, dass man dafür voraussetzen muss, dass die Gleichung $2^x = 3$ überhaupt eine Lösung hat.

Lösung 212: Zunächst haben wir ein Viereck (den „Rest“ nach Entfernen des kleinen Dreiecks aus dem großen) konstruiert, bei dem zwei nebeneinanderliegende Seiten gleich lang und zwei anliegende Winkel (die rechten) gleich groß sind.[23] Daraus folgt, dass die beiden blauen Seiten auch gleich lang sein müssen.

Die Winkelsumme in jedem Viereck beträgt 360°. In unserem Deltoid kennen wir drei Winkel, deren Summe 225° ist, also muss der vierte Winkel, der zwischen den beiden blauen Seiten, 135° haben. Daraus folgt, dass der eine Winkel des kleinen Dreiecks eine Größe von $180° - 135° = 45°$ hat. Da ein anderer Winkel in diesem Dreieck ein rechter ist, ist es ähnlich zu dem großen.

[23] Es handelt sich also um ein *Drachenviereck* bzw. *Deltoid*.

Da solch ein Dreieck insbesondere gleichschenklig ist, ist die rote Seite so lang wie die blauen.

Die Länge der roten Seite (und damit auch der blauen) ist die Differenz der Längen von Hypotenuse und Kathete des großen Dreiecks und damit als Differenz ganzer Zahlen auch eine ganze Zahl. Die Länge der Hypotenuse des kleinen Dreiecks ergibt sich als Differenz der Länge einer Kathete des großen Dreiecks und einer blauen Seite.

(Siehe dazu auch Kapitel 22, falls Ihnen die geometrischen Grundlagen fehlen.)

Lösung 214: Nach Aufgabe 210 ist $\sqrt{5}$ nicht rational. Dann kann aber $a = 1 + \sqrt{5}$ auch nicht rational sein, denn sonst wäre ja $\sqrt{5} = a - 1$ rational. Ebenso kann $\Phi = a/2$ nicht rational sein, weil sonst $a = 2\Phi$ rational wäre.

Lösung 215: Algebraisch (statt geometrisch) formuliert funktioniert der Algorithmus folgendermaßen: Gegeben ist eine Zahl x, deren Quadratwurzel wir berechnen wollen. Wir starten mit der Approximation $a_0 = x$ und setzen $b_0 = 1$. Dann gilt $a_0 \cdot b_0 = x$. Im nächsten Schritt berechnen wir den Mittelwert $a_1 = 1/2 \cdot (a_0 + b_0)$ und dazu $b_1 = x/a_1$, so dass wieder $a_1 \cdot b_1 = x$ gilt. Allgemein wird immer

$$b_n = x/a_n \quad \text{und} \quad a_{n+1} = 1/2 \cdot (a_n + b_n) = 1/2 \cdot (a_n + x/a_n)$$

berechnet.

Als Abbruchbedingung wurde in diesem Fall gewählt, dass das Quadrat der Näherung um maximal 0.001 vom richtigen Wert abweicht.[24] Die Funktion gibt zwei Werte zurück: die Approximation und die Abweichung von dem Wert, den Pythons Funktion `sqrt` berechnet.[25]

```
import math

def babylon (x):
    a = x
    while abs(a * a - x) > 0.001:
        a = (a + x / a) / 2
    return a, abs(a - math.sqrt(x))
```

Beachten Sie, dass man die Funktion auch mit Python-Objekten vom Typ `Fraction` (siehe Seite 106) aufrufen kann,[26] wodurch man tatsächlich einen Bruch als Ergebnis erhält:

```
from fractions import Fraction
babylon(Fraction(2, 1))
```

Lösung 216: Mit der folgenden Eingabe erhält man 1597/987:

[24] Das ist ein *absoluter Fehler*. Besser wäre es, einen *relativen Fehler* vorzugeben.

[25] Der ist natürlich auch in den meisten Fällen nicht korrekt, aber die beste Näherung im Rahmen der Rechengenauigkeit.

[26] Weil Python das sogenannte *Duck-Typing* unterstützt.

```
(1 + babylon(Fraction(5))[0]) / 2
```

Auf sechs Stellen nach dem Komma gerundet entspricht das dem korrekten Wert.

Lösung 217: Wir wissen (siehe Aufgabe 215), dass der richtige Wert immer zwischen a_n und b_n liegt. Daher ist eine einfache Lösung, die Funktion so zu modifizieren, dass a_n und b_n beide explizit berechnet werden. Ist deren Differenz kleiner als der vorgegebene maximal erlaubte Fehler, können sie auch vom richtigen Wert nicht weiter abweichen.

```
def babylon (x, err):
    a = x
    b = 1
    while abs(a - b) > err:
        a = (a + b) / 2
        b = x / a
    return a, abs(a - math.sqrt(x))
```

Lösung 218: Offenbar kann man hier nacheinander

$$\frac{1}{10},$$
$$\frac{1}{10}+\frac{1}{1000}=\frac{101}{1000},$$
$$\frac{1}{10}+\frac{1}{1000}+\frac{1}{1000000}=\frac{101001}{1000000},$$
$$\frac{1}{10}+\frac{1}{1000}+\frac{1}{1000000}+\frac{1}{10000000000}=\frac{10100100001}{100000000000}$$

und so weiter als Näherungen wählen. Wenn Sie mit der Summenschreibweise und der Gaußschen Summenformel schon vertraut sind, können Sie die n-te Approximation kurz auch so

$$\sum_{k=1}^{n} 10^{-k(k+1)/2}$$

schreiben. (Ansonsten müssen Sie sich etwas gedulden; das werden wir uns später noch mal genauer anschauen.)

Aber auch ohne die og. Kenntnisse kann man eine PYTHON-Funktion schreiben, die die n-te Approximation liefert:

```
from fractions import Fraction

def approx (n):
    sum = 0
    for k in range(1, n+1):
        exp = Fraction(0)
        for i in range(1, k+1):
            exp += i
```

```
        sum += 10 ** -exp
    return sum
```

Beachten Sie, dass wir durch die Zeile `exp = Fraction(0)` PYTHON dazu zwingen, einen Bruch auszugeben.

Lösung 219: Man könnte z.B. -42, $2/3$ und $\sqrt{13}$ nehmen. Natürlich sind alle diese Zahlen reell.

Lösung 220: Wir wissen bereits, dass es unendlich viele Primzahlen gibt. Nach Aufgabe 210 wissen wir, dass die Wurzeln von Primzahlen alle irrational sind. Damit haben wir schon unendlich viele irrationale Zahlen.

Lösung 221: Summen und Produkte von Brüchen sind natürlich wieder Brüche.[27] Wenn die Zahl $a = 3 + \sqrt{2}$ rational wäre, dann wäre auch die Summe der beiden rationalen Zahlen a und -3 rational. Diese Summe ist aber die Zahl $\sqrt{2}$, von der wir schon wissen, dass sie *nicht* rational ist. Daher kann a nicht rational sein. Ebenso kann $2 \cdot \sqrt{3}$ nicht rational sein, denn sonst wäre auch das Produkt $(2 \cdot \sqrt{3}) \cdot 1/2 = \sqrt{3}$ rational.

Lösung 222: Summen und Produkte von irrationalen Zahlen können auch rational sein. Zum Beispiel ist $3 + \sqrt{2}$ irrational (siehe vorherige Aufgabe) und natürlich ist auch $-\sqrt{2}$ irrational. Die Summe der beiden ist aber die ganze Zahl 3. Und das Produkt der irrationalen Zahl $\sqrt{7}$ mit sich selbst ist die ganze Zahl 7.

Ein noch schöneres Beispiel sind etwa $3 + \sqrt{2}$ und $3 - \sqrt{2}$. Beide sind irrational, aber in diesem Falls sind sogar Summe und Produkt rational. Rechnen Sie es nach!

Lösung 223: Erstaunlicherweise weiß niemand die Antwort. Ob $\pi + e$ rational oder irrational sind, ist ein bisher ungelöstes mathematisches Problem!

Lösung 224: Ohne Anspruch auf Vollständigkeit hier ein paar Dinge, die Sie über Wurzeln wissen sollten:

- Mit $\sqrt[n]{a}$ meint man die reelle Zahl x, die die Gleichung $x^n = a$ löst. Man nennt diese Zahl die *n-te Wurzel von a*.
- Wenn a eine nichtnegative reelle Zahl ist, gibt es immer so eine Lösung.
- Wenn a negativ ist, hat $x^n = a$ eine reelle Lösung, wenn n eine ungerade natürliche Zahl ist.
- Manchmal gibt es *zwei* reelle Lösungen, z.B. hat $x^4 = 16$ die Lösungen 2 und -2. In so einem Fall ist immer eine Lösung positiv und eine negativ. Mit $\sqrt[n]{a}$ ist in diesem Fall *immer* die positive Lösung gemeint.
- $\sqrt{a}$ ist eine Abkürzung für $\sqrt[2]{a}$. Man spricht von der *Quadratwurzel* von a. (Die dritte Wurzel nennt man auch die *Kubikwurzel*.)
- Es gilt immer $\sqrt[n]{a} \cdot \sqrt[n]{b} = \sqrt[n]{ab}$.

[27] Genau wie Summen und Produkte von ganzen Zahlen wieder ganze Zahlen sind. Und für natürliche Zahlen gilt das auch.

– Hingegen gilt das hier *nicht*: $\sqrt[n]{a} + \sqrt[n]{b} = \sqrt[n]{a+b}$. (Zum Beispiel ist $\sqrt{2} + \sqrt{2}$ ungefähr 2.828, $\sqrt{2+2}$ ist aber genau 2.) Das ist ein typischer Fehler, den Anfänger häufig machen!

Lösung 225: $(\sqrt{a})^2 = a$ gilt immer,[28] weil das ja die Definition der Wurzel ist. $\sqrt{a^2} = a$ gilt *nicht* immer. Wenn man z.B. $a = -2$ setzt, so gilt $\sqrt{a^2} = \sqrt{4} = 2 \neq a$, weil (siehe vorherige Aufgabe) die Quadratwurzel von 4 nach Definition die *positive* Lösung der Gleichung $x^2 = 4$ ist.

Hingegen gilt immer: $\sqrt{a^2} = |a|$.

Lösung 226: a^2 ist bekanntlich eine Abkürzung für $a \cdot a$, a^3 ist eine für $a \cdot a \cdot a$, und so weiter. Für positive natürliche Zahlen n ist also klar, was mit a^n gemeint ist. Es ergeben sich dann sofort die folgenden Rechenregeln:[29]

– $a^m \cdot a^n = a^{m+n}$
– $a^n \cdot b^n = (ab)^n$
– $(a^m)^n = a^{mn}$

Zudem liest man Terme wie a^{m^n} „von rechts nach links“, also so, als hätte man $a^{(m^n)}$ geschrieben. (a^{m^n} ist *nicht* dasselbe wie $(a^m)^n$. Setzen Sie z.B. $a = n = 2$ und $m = 3$.)

Wenn man nun definiert, dass a^0 immer eins sein soll und dass a^{-n} für $1/a^n$ stehen soll, dann hat man a^n für alle *ganzen* Zahlen n definiert. Und zwar so, dass die obigen Rechenregeln weiterhin gelten.

Man kann noch einen Schritt weiter gehen und $a^{1/n}$ definieren als $\sqrt[n]{a}$, falls a nichtnegativ ist. Damit hat man, bei erneuter Beibehaltung der Rechenregeln, a^r für alle *rationalen* Zahlen r definiert: Ist r der Bruch p/q, so ist $a^r = (a^p)^{1/q} = \sqrt[q]{a^p}$.[30]

Lösung 227: Bei Listen kommt es, anders als bei Mengen, auf die Reihenfolge an:[31]

```
[1, 2, 3] == [2, 3, 1]
```

Und auch Wiederholungen werden nicht einfach ignoriert:

```
[1, 2] == [1, 1, 2]
```

Lösung 228: `A` ist eine Liste. In diesem Fall gibt `len` die *Länge* der Liste zurück. `B` ist eine Menge. Hier gibt `len` die *Anzahl der Elemente* zurück. (Der Name der Funktion ist also etwas unglücklich.) `B` hat nur zwei Elemente, nämlich die Zahlen eins und zwei.

Lösung 229: So könnte das aussehen:

[28] Sofern die Wurzel existiert, wie in der Aufgabe ja schon gesagt. Dafür muss also a nichtnegativ sein.
[29] Machen Sie sich jeweils an Beispielen klar, dass das stimmt und warum es stimmt.
[30] Oder $a^r = (a^{1/q})^p = (\sqrt[q]{a})^p$, was dasselbe ist, wenn a nicht negativ sein kann.
[31] Hier und weiter unten sollten Sie als Ergebnis also jeweils `False` erhalten.

```
def subset(A, B):
    for x in A:
        if x not in B:
            return False
    return True
```

Lösung 230: Nein. Jedes Element von A ist auch eines von B, also hat B auf jeden Fall *nicht weniger* Elemente als A. Aber es könnte sein, dass A und B identisch sind. Dann würde natürlich auch $A \subseteq B$ gelten, aber B hätte genauso viele Elemente wie A und nicht mehr.

Lösung 231: Nein. Das Zeichen $\subseteq$ sieht zwar (durchaus absichtlich) so ähnlich wie $\leq$ aus, man sollte daraus aber keine voreiligen Schlüsse ziehen. Wenn Sie zwei (reelle) *Zahlen* a und b haben, für die $a \not\leq b$ gilt, dann können Sie sich sicher sein, dass $b \leq a$ gelten muss.

Bei Mengen muss das jedoch nicht so sein. Mit $A = \{1,2\}$ und $B = \{1,3\}$ gilt z.B. weder $A \subseteq B$ noch $B \subseteq A$.

Lösung 232: Wenn $A \not\subseteq B$ gilt, kann man keinerlei definitive Aussage darüber machen, wie sich die Anzahl der Elemente von A zur Anzahl der Elemente von B verhält. Betrachten Sie $A = \{1,2\}$ und $B_1 = \{1\}$, $B_2 = \{1,3\}$ und $B_3 = \{1,3,4\}$. A ist weder Teilmenge von B_1 noch von B_2 noch von B_3; aber eine der drei Mengen hat weniger Elemente als A, eine hat genauso viele und eine hat mehr.

Lösung 233: Wenn A Teilmenge von B ist, dann ist jedes Element von A auch eines von B. Wenn außerdem noch B Teilmenge von A ist, dann ist jedes Element von B zudem Element von A. Das bedeutet aber, dass A und B genau dieselben Elemente haben. Also sind A und B identisch.

Lösung 234: Wenn Sie Aufgabe 229 bearbeitet haben, sollte Ihnen eigentlich klar sein, warum das so ist.

Lösung 236: In beiden Fällen ist das Ergebnis A, für *jede* Menge A. Es sollte hoffentlich klar sein, warum das so ist.

Lösung 237: Zum Beispiel sind $\{1\}$ und $\{2\}$ disjunkt. Oder $\{1,2,3\}$ und $\{4,5,6\}$.

Lösung 238: $B \setminus A$ ist $\{5\}$, also *nicht* dasselbe wie $A \setminus B$, während grundsätzlich immer $A \cup B = B \cup A$ und $A \cap B = B \cap A$ gilt. Die beiden ersten Verknüpfungen sind also *kommutativ*, während die mengentheoretische Differenz das nicht ist.

Lösung 239: Hier meine Implementationen:

```
def union (A, B):
    result = set()
    for x in A:
        result.add(x)
    for x in B:
```

```
        result.add(x)
    return result

def intersection (A, B):
    result = set()
    for x in A:
        if x in B:
            result.add(x)
    return result

def difference (A, B):
    result = set()
    for x in A:
        if x not in B:
            result.add(x)
    return result
```

Unter den gleichen Namen gibt es in Python übrigens schon *Methoden* für Mengen. Man kann also sowas machen:

`intersection`

```
A = {1, 2, 3}
A.intersection({3, 4})
```

Lösung 240: Das Problem der Funktion sieht man, wenn man z.B. das hier eingibt:

```
A = {1, 2, 3}
myUnion(A, {3, 4, 5})
A
```

Der Wert von `A` wurde geändert. Mengen sind in Python Objekte, die man verändern kann.[32] Und diese Funktion verändert eines ihrer Argumente. Sowas sollte man wenn möglich vermeiden.

Lösung 241: Statt `A.add(x)` könnte man

```
A |= {x}
```

schreiben. (Als Abkürzung für: `A = A | {x}`. Siehe Aufgabe 5.)

Der Unterschied ist, dass `A.add(x)` tatsächlich das Objekt ändert (siehe Aufgabe 240), die obige Variante aber nicht unbedingt. Überlegen Sie sich vorher, welches Resultat herauskommen wird, wenn Sie den Code unten eingeben:

[32] Das ist ein wesentlicher Unterschied zur Mathematik, wo alle Objekte „auf ewig“ ihre Identität bewahren.

```
A = {1, 2, 3}
B = {1, 2, 3}
C = {1, 2, 3}
AA = A
BB = B
CC = C
A.add(4)
B |= {4}
C = C | {4}
AA, BB, CC
```

Sind Sie überrascht?

In der Mathematik gibt es übrigens keine abkürzende Schreibweise für „ein Element hinzufügen“. Man schreibt einfach $A \cup \{x\}$.

`remove`

Wie Listen haben auch Mengen in Python die Methode `remove`. Wenn es die nicht gäbe, könnte man `A.remove(x)` auch selbst implementieren:

```
A -= {x}
```

Lösung 243: Es könnte gemeint sein, dass C alle ungeraden Zahlen zwischen 3 und 59 enthält. Es könnte aber auch gemeint sein, dass C alle Primzahlen zwischen diesen beiden Zahlen enthält...

Lösung 244: Das würde man folgendermaßen machen:

```
A = set(range(101))
B = set(range(2, 81, 2))
```

Lösung 245: Es liegt nahe, $\mathbb{N}^+$ als $\mathbb{N} \setminus \{0\}$ zu definieren.

Lösung 246: Für die erste Interpretation könnte man es so machen:

$$C = \{x : x \in \mathbb{N} \text{ und } 3 \leq x \leq 59 \text{ und } 2 \nmid x\}$$

Für die zweite so:

$$C = \{x : x \in \mathbb{N} \text{ und } 3 \leq x \leq 59 \text{ und } x \text{ ist prim}\}$$

Oder noch kürzer so (siehe Seite 79):

$$C = \{x : x \in \mathbb{P} \text{ und } 3 \leq x \leq 59\}$$

Lösung 247: Mathematisch würde es so (oder so ähnlich) aussehen:

$$X_1 = \{z \in \mathbb{Z} : |z| < 10\}$$
$$X_2 = \{p \in \mathbb{P} : p^2 < 10000\}$$
$$X_3 = \{n \in \mathbb{N} : n < 500 \text{ und } n \text{ ist eine Quadratzahl}\}$$

In Python könnte man es so machen:

```
import sympy

def perfectSquare (n):
    for i in range(n + 1):
        if i * i == n:
            return True
    return False

X1 = set(range(-9, 10))
X2 = {p for p in range(2, 101)
        if sympy.isprime(p) and p * p < 10000}
X3 = {n for n in range(500) if perfectSquare(n)}
```

Lösung 248: Zunächst mathematisch:

$$Y_1 = \{1/n : n \in \mathbb{N} \text{ und } 0 < n \leq 100\}$$
$$Y_2 = \{2^n : n \in \mathbb{N} \text{ und } n \leq 20\}$$
$$Y_3 = \{(-1)^n \cdot n^3 : n \in \mathbb{N} \text{ und } 0 < n \leq 30\}$$

Und nun in PYTHON:

```
from fractions import Fraction

Y1 = {Fraction(1, n) for n in range(1, 101)}
Y2 = {2 ** n for n in range(21)}
Y3 = {((-1) ** n) * (n ** 3) for n in range(1,31)}
```

Lösung 249: $\mathbb{Z} = \mathbb{N} \cup \{-n : n \in \mathbb{N}\}$ wäre eine Möglichkeit. Beachten Sie, dass die Null auf beiden Seiten des Vereinigungzeichens vorkommt. Das macht aber natürlich nichts.

Lösung 250: A muss offenbar $3 \cdot 4$ Elemente haben, da es drei mögliche Zähler und vier mögliche Nenner gibt und da die Zähler und Nenner jeweils paarweise teilerfremd sind. Das Ergebnis sieht so aus:

$$A = \left\{ \frac{1}{3}, \frac{2}{3}, \frac{5}{3}, \frac{1}{7}, \frac{2}{7}, \frac{5}{7}, \frac{1}{9}, \frac{2}{9}, \frac{5}{9}, \frac{1}{11}, \frac{2}{11}, \frac{5}{11} \right\}$$

Lösung 251: So zum Beispiel:[33]

$$A = \{n/d : n \in \mathbb{N}^+ \text{ und } d \in \mathbb{N}^+ \text{ und } n \leq 4 \text{ und } 2 \leq d \leq 5\}$$

Die meisten Mathematiker würden es sogar noch etwas kürzer schreiben:

$$A = \{n/d : n, d \in \mathbb{N}^+ \text{ und } n \leq 4 \text{ und } 2 \leq d \leq 5\}$$

[33] Falls Sie nicht wissen, was $\mathbb{N}^+$ ist, haben Sie Aufgabe 245 nicht bearbeitet...

Lösung 252: Falls Ihre Antwort $4 \cdot 4 = 16$ war, so liegen Sie falsch. Die richtige Antwort wäre 13 gewesen. (Probieren Sie es ggf. in Python aus, wenn Sie es nicht glauben.)

Das liegt daran, dass die Eins drei Mal vorkommt (als $2/2$, $3/3$ und $4/4$) und das $1/2$ noch mal als $2/4$ vorkommt. Da es sich um eine Menge handelt, werden diese mehrfachen Vorkommen natürlich jeweils nur ein Mal „gewertet“.

Lösung 253: So könnte man's machen:

$$\mathbb{Q} = \{n/d : n \in \mathbb{Z} \text{ und } d \in \mathbb{N}^+\}$$

Haben Sie bei Ihrer Definition daran gedacht, dass im Nenner nicht null stehen darf?

Beachten Sie außerdem, dass in meiner Definition nur der Zähler ein negatives Vorzeichen haben kann. Das reicht, um auch alle negativen Brüche zu „erwischen“. (Siehe Aufgabe 166.) Außerdem ist die Definition im gewissen Sinne sehr „verschwenderisch“, weil jeder Bruch unendlich oft vorkommt. (Siehe Aufgabe 252.)

Lösung 254: A_1 ist kein Schnitt, weil die Zahl 42 das größte Element der Menge ist. A_2 ist kein Schnitt, weil z.B. 3 zu A_2 gehört, aber $5/2$ kleiner als 3 ist und nicht zu A_2 gehört. Damit ist die zweite Bedingung verletzt. A_3 verletzt ebenfalls die zweite Bedingung.

Lösung 255: Das ist eigentlich anschaulich ganz klar. Man sagt, dass $A < B$ dann und nur dann gilt, wenn $A \subsetneq B$ gilt.

Lösung 256: $\{1, 2, 3\}$ ist nach oben beschränkt. Als Schranke kann man z.B. 3 nehmen. Überhaupt ist jede endliche Menge nach oben beschränkt: Man kann sich immer einfach das größte Element herauspicken, um eine Schranke zu erhalten.

Aber nicht nur endliche Mengen sind nach oben beschränkt. $\{q \in \mathbb{Q} : |q| < 1\}$ ist auch nach oben beschränkt, z.B. durch die Schranke 1.

Lösung 257: $\mathbb{N}$ ist z.B. nicht nach oben beschränkt. Und das gilt auch für $\mathbb{P}$, $\mathbb{Z}$ oder $\mathbb{Q}$ selbst. Oder für $\{q \in \mathbb{Q} : |q| > 1\}$.

Lösung 258: Ist A ein Dedekindscher Schnitt, so ist A nach Definition nicht $\mathbb{Q}$, also gibt es eine rationale Zahl C, die nicht zu A gehört. Wäre C kleiner als eine der Zahlen aus A, so würde C zu A gehören, weil A ein Schnitt ist. Daher muss C größer als alle Zahlen aus A sein; C ist also eine obere Schranke von A.

Lösung 259: Wenn A nach unten beschränkt ist, dann ist $\{-a : a \in A\}$ nach oben beschränkt und hat ein Supremum S. $-S$ ist dann Infimum von A.

Lösung 261: Das Argument für `frozenset` muss selbst bereits eine Ansammlung von Objekten sein, also eine Menge, eine Liste oder sowas in der Art. Im ersten Fall ist das Argument für den äußeren Aufruf von `frozenset` eine Menge, deren einziges Element eine Menge ist. Das Ergebnis entspricht also der mathematischen Menge $\{\{1\}\}$. Im zweiten Fall ist das Argument für den äußeren Aufruf von `frozenset` eine Menge, deren einziges Element die Zahl eins ist. Das Ergebnis entspricht also der mathematischen Menge $\{1\}$.

Lösung 262: Es gibt viele Möglichkeiten, das zu machen. Zum Beispiel so:[34]

```
def pair (x, y):
    return frozenset({frozenset({x}), frozenset({x, y})})
```

Das Paar (x, y) wird also mittels der Menge $\{\{x\}, \{x, y\}\}$ definiert.

Es mag Ihnen müßig vorkommen, sich über sowas Gedanken zu machen. Dies ist aber ein Beispiel dafür, wie in der Untersuchung der Grundlagen der Mathematik gezeigt wird, dass sich jedes mathematische Objekt durch Mengen darstellen lässt. Man kann z.B. auch die natürlichen Zahlen über Mengen definieren und darauf aufbauend dann die ganzen Zahlen, die rationalen, und so weiter.

Lösung 263: Am kürzesten geht's wohl so:

```
def cartesianProdukt (A, B):
    return {(a, b) for a in A for b in B}
```

Python hat dafür auch die Funktion `product`. Siehe Kapitel 61.

`product`

Lösung 264: Nein. Z.B. ist $\{1\} \times \{2\} = \{(1,2)\}$, aber $\{2\} \times \{1\} = \{(2,1)\}$. Und da $(1,2)$ und $(2,1)$ zwei *verschiedene* Tupel sind, sind die beiden Mengen auch verschieden.

Lösung 266: $|A \times B| = |A| \cdot |B|$. Man kann das sofort an der tabellarischen Darstellung (siehe Seite 167) erkennen. Dies erklärt auch, warum man von einem *Produkt* spricht.

Lösung 267: Offensichtlich gilt immer $|A \cup B| \leq |A| + |B|$.

Lösung 268: So müsste es für $|A \cup B \cup C \cup D|$ aussehen:

$$\begin{gathered} |A| + |B| + |C| + |D| \\ -|A \cap B| - |A \cap C| - |A \cap D| - |B \cap C| - |B \cap D| - |C \cap D| \\ +|A \cap B \cap C| + |A \cap B \cap D| + |A \cap C \cap D| + |B \cap C \cap D| \\ -|A \cap B \cap C \cap D| \end{gathered}$$

Lösung 269: Wir bezeichnen mit A_n die Menge der Zahlen unter 1000, die durch n teilbar sind. Wir wollen wissen, wie viele Elemente die Menge

$$B = A_2 \cup A_3 \cup A_5 \cup A_7$$

hat. Die Lösung für die Aufgabe ist nämlich dann $1000 - |B|$.

Wie viele Elemente enthält eine der Mengen A_n? Für $n = 5$ sind es die Zahlen 0, 5, 10, 15 bis 995, also die Zahlen von $0 \cdot 5$ bis $199 \cdot 5$. Das sind 200, also $1000/5$ Zahlen. Für $n = 3$ sind es die Zahlen $0 \cdot 3$ bis $333 \cdot 3$, also 334 Stück. Auf die Zahl 334 kommt man,

[34] Diese Darstellung stammt von dem polnischen Mathematiker Kazimierz Kuratowski, der zu den Autoren des „Schottischen Buches" gehörte. Das war eine Kladde, in der Lemberger Mathematiker Mitte des 20. Jahrhunderts in einem Kaffeehaus ungelöste Probleme sammelten.

wenn man 1000 durch 3 teilt und dann die nächstgrößere ganze Zahl ermittelt. Man schreibt das mithilfe der sogenannten *Gaußklammer* als $\lceil 1000/3 \rceil$.[35] Allgemein hat A_n also $\lceil 1000/n \rceil$ Elemente.

Wenn man nun z.B. $A_3 \cap A_7$ betrachtet, so enthält diese Menge alle Zahlen unter 1000, die sowohl durch 3 als auch durch 7 teilbar sind. Das sind aber offensichtlich genau die Zahlen, die durch $3 \cdot 7 = 21$ teilbar sind, weil 3 und 7 Primzahlen sind. Mit anderen Worten bedeutet das: $A_3 \cap A_7 = A_{21}$. Ebenso ist $A_2 \cap A_3 \cap A_5 = A_{30}$, etc.

Mit dem Inklusions-Exklusions-Prinzip ergibt sich nun:

$$\begin{aligned}
|B| &= |A_2| + |A_3| + |A_5| + |A_7| \\
&\quad - |A_2 \cap A_3| - |A_2 \cap A_5| - |A_2 \cap A_7| - |A_3 \cap A_5| - |A_3 \cap A_7| - |A_5 \cap A_7| \\
&\quad + |A_2 \cap A_3 \cap A_5| + |A_2 \cap A_3 \cap A_7| + |A_2 \cap A_5 \cap A_7| + |A_3 \cap A_5 \cap A_7| \\
&\quad - |A_2 \cap A_3 \cap A_5 \cap A_7| \\
&= |A_2| + |A_3| + |A_5| + |A_7| - |A_6| - |A_{10}| - |A_{14}| - |A_{15}| - |A_{21}| - |A_{35}| \\
&\quad + |A_{30}| + |A_{42}| + |A_{70}| + |A_{105}| - |A_{210}| \\
&= \left\lceil \frac{1000}{2} \right\rceil + \left\lceil \frac{1000}{3} \right\rceil + \left\lceil \frac{1000}{5} \right\rceil + \left\lceil \frac{1000}{7} \right\rceil \\
&\quad - \left\lceil \frac{1000}{6} \right\rceil - \left\lceil \frac{1000}{10} \right\rceil - \left\lceil \frac{1000}{14} \right\rceil - \left\lceil \frac{1000}{15} \right\rceil - \left\lceil \frac{1000}{21} \right\rceil - \left\lceil \frac{1000}{35} \right\rceil \\
&\quad + \left\lceil \frac{1000}{30} \right\rceil + \left\lceil \frac{1000}{42} \right\rceil + \left\lceil \frac{1000}{70} \right\rceil + \left\lceil \frac{1000}{105} \right\rceil - \left\lceil \frac{1000}{210} \right\rceil \\
&= 500 + 334 + 200 + 143 - 167 - 100 - 72 - 67 - 48 - 29 \\
&\quad + 34 + 24 + 15 + 10 - 5 = 772
\end{aligned}$$

Die gesuchte Antwort ist somit 228. Überprüfen können wir das so:

```
c = 0
for n in range(1000):
    if n%2 != 0 and n%3 != 0 and n%5 != 0 and n%7 != 0:
        c += 1
c
```

Lösung 270: Wir bezeichnen jeweils mit A_i die Menge der Zahlen, die die i-te Bedingung erfüllen. Sicher ist $|A_1| = 99 - 9 = 90$. A_2 besteht aus den Zahlen (in Siebenerschritten) von $1 \cdot 7$ bis $142 \cdot 7 = 994$, hat also 142 Elemente. A_3 hat offensichtlich $3 \cdot 9 = 27$ Elemente. In $A_1 \cap A_2$ sind die Zahlen von $2 \cdot 7 = 14$ bis $14 \cdot 7 = 98$, also insgesamt 13 Stück. In $A_1 \cap A_3$ sind genau neun Zahlen. In $A_2 \cap A_3$ sind nur die Zahlen 7, 77 und 777. $A_1 \cap A_2 \cap A_3$ ist schließlich $\{77\}$. Mit dem Inklusions-Exklusions-Prinzip ergibt sich als Antwort nun:

$$|A_1 \cup A_2 \cup A_3| = 90 + 142 + 27 - 13 - 9 - 3 + 1 = 235$$

`ceil`

[35] Im Englischen spricht man von der *ceiling function*. In PYTHON gehört diese unter dem Namen `ceil` zur schon erwähnten MATH-Bibliothek.

Mit PYTHON könnte man das so überprüfen:

```
def identicalDigits (n):
    D = {n % 10}
    d10 = (n % 100) // 10
    d100 = n // 100
    if d100 != 0:
        D.add(d100)
    if d10 != 0 or d100 != 0:
        D.add(d10)
    return len(D) == 1

c = 0
for n in range(1, 1000):
    if (9 < n < 100) or n % 7 == 0 or identicalDigits(n):
        c += 1
c
```

Lösung 272: Mit `sum` von Seite 170 bekommen wir null als Ergebnis:

```
def s (n):
    return 3 * n
sum(s, 1, 0)
```

Und das entspricht auch der mathematischen Definition für so eine *leere Summe*: Wenn man gar keine Summanden hat, dann ist das Ergebnis null – und nicht etwa undefiniert, wie manche denken. Das ist keine willkürliche Festsetzung, sondern die einzig sinnvolle Definition, weil null das neutrale Element der Addition ist.

Lösung 276: Wie bei der Herleitung der Gaußschen Summenformel können wir die Summe doppelt aufaddieren und dabei die Reihenfolge umdrehen:

$$\begin{aligned} &s_a + s_{a+1} + \cdots + s_{b-1} + s_b \\ +\,&s_b + s_{b-1} + \cdots + s_{a+1} + s_a \end{aligned}$$

Da der Abstand von je zwei aufeinander folgenden Summanden nach Aufgabenstellung immer gleich ist, ist die Summe von zwei übereinander stehenden Summanden immer $s_a + s_b$. Als Ergebnis erhält man also die Summe des kleinsten und des größten Summanden multipliziert mit der Anzahl der Summanden, und das ganze dann wieder durch zwei geteilt:[36]

$$1/2 \cdot (s_a + s_b) \cdot (b - a + 1)$$

Beachten Sie, dass die Anzahl der Summanden $b - a + 1$ und nicht $b - a$ ist. (Siehe dazu auch die Anmerkung zu Lösung von Aufgabe 297.) Außerdem stimmt diese Formel natürlich nur unter der Voraussetzung, dass $b \geq a$ gilt.

[36] Man multipliziert also den Mittelwert der Summanden mit ihrer Anzahl.

Lösung 277: Man würde in diesem Fall

$$((5+500)\cdot 100)/2 \qquad \text{und} \qquad ((10+50)\cdot 41)/2$$

berechnen. Die Ergebnisse müssen natürlich identisch zu den bereits berechneten sein. Es ist Geschmackssache, welche Methode man einfacher findet.

Lösung 278: Man kann natürlich den Computer arbeiten lassen:

```
def s (n):
    return 42
sum(s, 3, 10)
```

Es sollte aber auch so klar sein, dass immer nur derselbe Summand aufaddiert wird. Man muss also einfach 42 mit der Anzahl der Summanden multiplizieren: $8 \cdot 42 = 336$.

Lösung 280: Das ist deshalb erlaubt, weil die Addition kommutativ und assoziativ ist. Wir dürfen also die Reihenfolge der Summanden vertauschen und beliebig klammern.

Lösung 281: Das geht so:

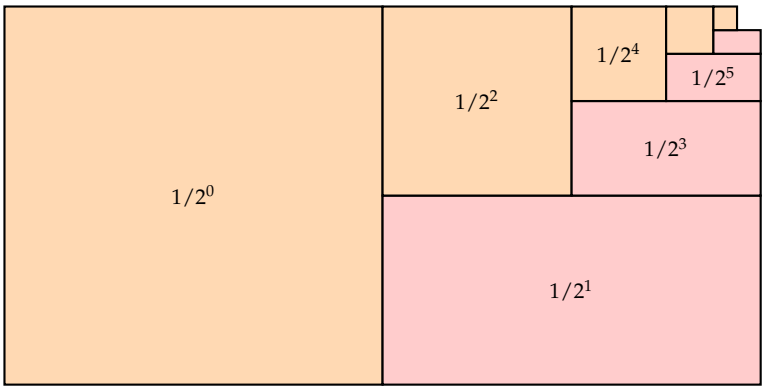
$$\sum_{k=1}^{20}(k+2)^2 = \sum_{k=1}^{20}(k^2+4k+4) = \left(\sum_{k=1}^{20}k^2\right) + 4\cdot\left(\sum_{k=1}^{20}k\right) + \left(\sum_{k=1}^{20}4\right)$$

Die einzelnen Summanden kann man nun mit bekannten Formeln berechnen. (Das sollten Sie natürlich auch tun und mit PYTHON überprüfen.)

Lösung 282: Bei der ersten Summe muss man beachten, dass der erste Index eins und nicht null ist. Zum Überprüfen der zweiten Summe bietet es sich an, auch in PYTHON mit Brüchen zu rechnen:

```
from fractions import Fraction
def q (n):
    return Fraction(1, 2 ** n)
sum(q, 0, 8)
```

Vielleicht ist Ihnen aufgefallen, dass das Ergebnis, das Sie herausbekommen, sehr nahe an 2 liegt. Das ist kein Zufall. Der erste Summand ist 1 und jeder weitere Summand ist halb so groß wie der vorherige. Stellt man sich die Summanden als Flächen vor, sieht es so aus:



Wenn wir uns in Kapitel 43 über *Reihen* unterhalten, werden wir sehen, dass wir dem Wert 2 beliebig nahe kommen können, wenn wir nur „lange genug addieren“.

Lösung 284: Die Funktion sollte in etwa so aussehen:

```
def prod (s, a, b):
    result = 1
    k = a
    while k <= b:
        result *= s(k)
        k += 1
    return result
```

Eine Änderung gegenüber `sum` ist natürlich, dass innerhalb der Schleife multipliziert wird. Eine zweite ganz wesentliche Änderung ist jedoch, dass der Anfangswert für `result` eins und nicht null ist, da sonst als Produkt nie etwas anderes als null herauskäme.

Lösung 285: Wie in Aufgabe 272 kann man argumentieren, dass das leere Produkt die Eins, also das neutrale Element der Multiplikation ist. 0! ist dann ebenfalls eins, was auf den ersten Blick etwas seltsam erscheinen mag, aber tatsächlich auch der einzig logische Wert ist. Das wird später deutlich werden, wenn wir z.B. mit Binomialkoeffizienten rechnen.

Lösung 286: Man kann konstante Faktoren aus einem Produkt herausziehen, muss aber vorsichtig sein. Z.B. ist $\prod_{k=1}^{7} 3k$ nicht etwa $3 \cdot 7!$, sondern $3^7 \cdot 7!$. Es ergibt sich nämlich

$$\prod_{k=1}^{7} 3k = (3 \cdot 1) \cdot (3 \cdot 2) \cdot \cdots \cdot (3 \cdot 7) = (3 \cdot 3 \cdot \cdots \cdot 3) \cdot (1 \cdot 2 \cdot \cdots \cdot 7),$$

der Faktor taucht also sieben Mal im Produkt auf.

Lösung 287: Angeblich handelt es sich bei dieser Aufgabe um eine Frage, die in ähnlicher Form gerne mal in Bewerbungsgesprächen gestellt wird. Sie sind also nun gewappnet...

Sei n die Zahl, die es herauszufinden gilt, d.h. die Chips können mit n aktiven Kernen zehn Minuten überstehen, mit $n+1$ jedoch nicht. Nach Aufgabenstellung kann n jede natürliche Zahl zwischen 0 und 100 (inklusive) sein.

Für den ersten Testplan bietet sich die aus der Informatik bekannte sogenannte *binäre Suche* an: Sie testen zunächst mit der Hälfte, also mit 50 aktiven Kernen. Wenn der Chip durchbrennt, wissen Sie, dass $n < 50$ ist, anderenfalls muss $n \geq 50$ gelten. Nun wählen Sie wieder die Mitte – im ersten Fall testen Sie mit 25 Kernen, im zweiten mit 75. Jeder Test halbiert die Anzahl der möglichen Werte. Da 2^7 größer als 100 ist, sind Sie nach maximal sieben Tests fertig. Im schlimmsten Fall kann Ihnen jedoch passieren, dass nacheinander bei 50, 25, 12, 6, 3 und 2 aktiven Kernen der Chip jeweils durchbrennt. Sie müssen dann immer noch mit einem einzigen Kern testen, und wenn auch dieser Chip den Test nicht überlebt, wissen Sie, dass die Entwicklungsingenieure nicht gerade ein Meisterwerk abgeliefert haben. Sie haben dafür allerdings auch sieben Chips „verbraten“.

Wie kann man nun die Zahl der zerstörten Chips minimieren? Hätten Sie 100 Chips zur Verfügung, so könnten Sie nacheinander mit einem Kern, mit zwei Kernen, mit dreien etc. testen und beim ersten durchgebrannten Chip aufhören. Sie würden dann auf jeden Fall nur einen zerstören. Leider haben Sie aber nur 20 bekommen. Daher kommt Ihnen vielleicht die Idee, zuerst mit 20 Kernen zu testen. Wenn der Chip durchbrennt, können Sie mit den verbleibenden 19 Chips nacheinander „von unten“ testen und machen höchstens einen weiteren kaputt. Wenn der Chip mit 20 aktiven Kernen nicht durchgebrannt ist, haben Sie keinen Chip zerstört und noch 19 Chips für weitere Tests zur Verfügung. Als nächstes testen Sie mit 39 Kernen. Brennt der Chip nun durch, testen Sie danach mit 21, 22, 23, ... Kernen, bis wieder ein Chip den Geist aufgibt. Wieder müssen Sie höchstens zwei Chips opfern. Ging es auch mit 39 Kernen gut, dann ist noch kein Chip kaputt, und sie haben noch 18 für die weiteren Tests. Nun testen Sie also mit $39 + 18 = 57$ Chips und so weiter. Bei diesem Testplan brennen grundsätzlich nie mehr als zwei Chips durch.

Es geht aber noch schneller, ohne dass man mehr als zwei Chips aufs Spiel setzen muss: Ihnen ist vielleicht aufgefallen, dass man so lange addiert, bis man 100 erreicht: $20 + 19 + 18 + 17 + 16 + 15 = 105$. Man könnte sich nun umgekehrt fragen, wie weit man in aufsteigender Richtung (also $1 + 2 + 3 + 4 + \ldots$) addieren muss, bis die Summe mindestens 100 ist. Die Gaußsche Formel für die arithmetische Summe verrät uns, dass es reicht, mit der 14 anzufangen. Der daraus resultierende Testplan sieht dann so aus: Man testet nacheinander mit 14, $14 + 13 = 27$, $27 + 12 = 39$, 50, 60, 69, 77, 84, 90, 95, 99 und 100 aktivierten Kernen, bis zum ersten Mal ein Chip durchbrennt. Dann wählt man einen Kern mehr als beim letzten „geglückten“ Versuch und arbeitet sich von dort hoch, bis der zweite Chip durchgebrannt ist. (Wenn also z.B. der erste Chip mit 69 Kernen durchbrennt, prüft man danach mit 61, 62, 63, 64, ... Kernen.) Mit dieser Methode werden höchstens zwei Chips zerstört und man braucht maximal 14 Tests.

Lösung 288: Es gibt insgesamt 24:

1234	1243	1324	1342	1423	1432
2123	2132	2314	2341	2413	2431
3124	3142	3214	3241	3412	3421
4123	4132	4213	4231	4312	4321

Lösung 289: Im Prinzip reicht es, als erste Zeile nach dem `def` den Befehl `print(L)` einzufügen. Eine etwas übersichtlichere Ausgabe erhält man aber so:

```
def perms (L, k = 0):
    print("  " * k + str(L))
    if len(L) <= 1:
        return [L]
    return [[L[i]] + P for i in range(len(L))
                       for P in perms(L[:i]+L[i+1:], k+1)]
```

Hier wird der Funktion als zweites Argument eine Zahl übergeben, die angibt, wie weit die ausgedruckte Liste eingerückt werden soll. Dann kann man besser erkennen, welche Funktion von welcher aufgerufen wurde.

Wir sehen hier zwei Features von Python, die wir bisher nicht besprochen haben:

- Funktionsparameter können *voreingestellte* Werte bekommen.[37] Man kann die Funktion dann ohne Angabe des entsprechenden Argumentes aufrufen. Das hat denselben Effekt, als hätte man den entsprechenden Wert benutzt. In diesem Fall bewirkt `perm([42])` also dasselbe wie `perm([42], 0)`.
- Man kann einen String mit einer Zahl „multiplizieren“. Das entspricht einer entsprechend häufigen Wiederholung des Strings. Probieren Sie z.B. das aus:

  ```
  "bla" * 3
  ```

  `*`

Lösung 290: So sollte es aussehen:

```
def fact (n):
    return 1 if n <= 1 else n * fact(n - 1)
```

Lösung 291: Wenn Sie diese Funktion aufrufen, werden Sie eine Fehlermeldung bekommen. Das Problem ist, dass es rein theoretisch immer weiter geht: Jeder Aufruf von `fact` resultiert in einem weiteren Aufruf von `fact`.[38] Wir sehen, dass es bei rekursiven Algorithmen unverzichtbar ist, eine Abbruchbedingung zu haben!

Lösung 292: Sie bekommen höchstwahrscheinlich eine Fehlermeldung wie bei Aufgabe 291. Rekursion ist ein sehr elegantes Werkzeug, erfordert aber einen gewissen

[37] Der übliche Fachbegriff dafür ist *Default*.

[38] In der Praxis geht es nicht immer weiter, weil irgendwann die Ressourcen des Python-Interpreters erschöpft sind. Dann bekommen Sie die Fehlermeldung.

Aufwand. Da PYTHON eine interpretierte Sprache ist, erlaubt sie nur eine gewisse *Rekursionstiefe*: eine Funktion darf sich nicht zu oft selbst aufrufen.[39]

Lösung 294: Im ersten Beispiel ist $n = 4$, $r = 2$, $k_1 = 3$ und $k_2 = 1$. (Oder $k_1 = 1$ und $k_2 = 3$. Das geht natürlich auch.) Im zweiten Beispiel ist $k_1 = k_2 = 2$; n und r haben dieselben Werte wie vorher.

Lösung 295: Offensichtlich muss immer $n = k_1 + k_2 + \cdots + k_r$ gelten.

Lösung 296: Sie können das mit dem Computer ja selbst überprüfen, daher rechnen wir exemplarisch nur das letzte Beispiel durch:

```
len(set(perms((1, 1, 2, 2, 3))))
```

Hier haben wir $n = 5$, $r = 3$, $k_1 = k_2 = 2$ und $k_3 = 1$. Daher muss das Ergebnis $5!/(2!\,2!\,1!) = 30$ sein.

Lösung 297: Ja, das ist schon richtig so. (Sonst wäre es ein Druckfehler. Ich würde nicht absichtlich falsche Formeln im Buch aufnehmen.) In unserem Beispiel etwa ist $n = 7$ und $k = 3$. $n - k$ wäre 4, der letzte Faktor ist aber 5.

Die Aufgabe wurde deshalb aufgenommen, da in solchen Situationen gerne falsch gerechnet wird, so dass man „um einen daneben“ liegt. In der Informatik ist das unter dem Namen *Off-by-one-Error* bekannt.

Lösung 298: Wir machen uns das an unserem Beispiel klar:

$$\frac{7!}{(7-3)!} = \frac{7!}{4!} = \frac{7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1}{4 \cdot 3 \cdot 2 \cdot 1} = 7 \cdot 6 \cdot 5$$

Die letzte Umformung ist korrekt, weil man die Terme, die im Bruch jeweils übereinander stehen, alle herauskürzen kann.

Lösung 299: Das ist sogar sehr sinnvoll. Bei einer Variation werden k von n Elementen geordnet ausgewählt. Wenn k und n identisch sind, hat man es mit einer Permutation zu tun. Beachten Sie, dass auch die Formeln übereinstimmen. Setzt man $k = n$, so wird aus dem Ausdruck (17.2) von Seite 186 einfach $n!$, weil $n - n + 1 = 1$ gilt.

Lösung 300: So klappt es:

```
def kperm (L, k):
    if k == 0:
        return [[]]
    return [[L[i]]+P for i in range(len(L))
                     for P in kperm(L[:i] + L[i+1:], k-1)]
```

Variationen werden in der englischen Fachsprache *k-permutations* genannt.

Lösung 301: Das ist natürlich noch leichter als in der vorherigen Aufgabe:

[39] Manche Programmiersprachen (z.B. SCHEME) garantieren dem Programmierer, dass es für sogenannte *endrekursive* Funktionen keinerlei Beschränkungen der Rekursionstiefe gibt.

```
def ktup (L, k):
    if k == 0:
        return [[]]
    return [[L[i]]+P for i in range(len(L))
                     for P in ktup(L, k-1)]
```

Variationen mit Wiederholungen werden in der englischen Fachsprache einfach *k-tuples* genannt.

Lösung 303: Das würde dann so aussehen:

```
def combs (L, k):
    if k == 0:
        return {frozenset()}
    return {frozenset([x]) | P for x in L
                               for P in combs(L-{x}, k-1)}
```

Man beachte die Übereinstimmung mit der Funktion `kperm` oben. Diese Lösung ist allerdings alles andere als effizient, weil sie viele „unnötige“ Lösungen berechnet, die dann später als Doubletten wieder entfernt werden müssen.

Lösung 304: Mit der Funktion `fact` von Seite 15 geht es ganz einfach:

```
def binom (n, k):
    return fact(n) // (fact(k) * fact(n - k))
```

Man hätte es aber auch so machen können:

```
def binom (n, k):
    numerator = 1
    denominator = 1
    while k > 0:
        denominator *= k
        numerator *= n
        n -= 1
        k -= 1
    return numerator // denominator
```

Diese Version sieht umständlicher aus, hat aber einen Vorteil gegenüber der ersten. Können Sie ihn erkennen? Noch besser kann man es folgendermaßen machen:

```
def binom (n, k):
    if k == 0:
        return 1
    if k * 2 > n:
```

```
        return binom(n, n - k)
    return binom(n, k - 1) * (n + 1 - k) // k
```

Hier wird (in der zweiten `if`-Anweisung) die Symmetrie des Binomialkoeffizienten ausgenutzt, die im Buch direkt hinter dieser Aufgabe erläutert wird. Außerdem ist dies offenbar eine rekursive Definition.

Lösung 306: Wir benutzen einfach Gleichung (17.6) von Seite 190 und formen um:

$$\begin{aligned}\binom{n}{k}+\binom{n}{k+1} &= \frac{n!}{k!(n-k)!}+\frac{n!}{(k+1)!(n-(k+1))!}\\ &= \frac{(k+1)n!}{(k+1)!(n-k)!}+\frac{(n-k)n!}{(k+1)!(n-k)!}\\ &= \frac{(k+1)n!+(n-k)n!}{(k+1)!(n-k)!} = \frac{(n+1)!}{(k+1)!(n-k)!}\\ &= \frac{(n+1)!}{(k+1)!((n+1)-(k+1))!} = \binom{n+1}{k+1}\end{aligned}$$

Der entscheidende Schritt dabei ist, dass wir die beiden Brüche auf einen gemeinsamen Nenner bringen, indem wir den linken Bruch mit $k+1$ und den rechten mit $n-k$ erweitern.

Lösung 307: Die ersten Summen sind 1, 2, 4, 8 und 16. Ihnen sollte aufgefallen sein, dass es sich um Zweierpotenzen handelt. Genauer: Wenn man bei null anfängt zu zählen, dann ist die Summe über die n-te Zeile immer 2^n.

Übrigens kann man mit so einer Schlussweise (fünf Werte anschauen und daraus ein Bildungsgesetz für alle anderen folgern) auch mal auf die Nase fallen. Das demonstriert das Video, dessen QR-Code Sie am Rand finden und in dem es auch um Kombinatorik geht.

Lösung 310: Das würde dann so aussehen:

```
def powerSet (A):
    if A == set():
        return {frozenset()}
    a = A.pop()
    P = powerSet(A)
    return P | set(X | {a} for X in P)
```

Lösung 311: Die Funktion (und auch die aus Aufgabe 310) modifiziert ihr Argument. Das kann man z.B. so erkennen:

```
A = {10, 20, 30}
powerSet(A)
A
```

Es gibt verschiedenen Möglichkeiten, dieses Problem zu beheben. Nach meiner Meinung ist aber keine von denen wirklich elegant. Ich würde es vielleicht so machen:

```
def powerSetHelper (A):
    if A == set():
        return [set()]
    a = A.pop()
    P = powerSetHelper(A)
    return P + [X | {a} for X in P]

def powerSet (A):
    return powerSetHelper(A.copy())
```

Die ursprüngliche Funktion wird zur Hilfsfunktion „degradiert“. Der Job der neuen Funktion `powerSet` ist es nur noch, eine Kopie des Arguments zu erstellen und diese Kopie an die Hilfsfunktion zu übergeben.

`copy`

Lösung 312: Hier ein Vorschlag:

```
def combs (S, k):
    if k == 0:
        return {frozenset()}
    if k > len(S):
        return set()
    x = S.pop()
    return combs(S.copy(), k) | \
           {A | frozenset([x])
                   for A in combs(S.copy(), k-1)}
```

Dies ist eine etwas kompliziertere Rekursion als die, die wir bisher gesehen haben, weil `combs` sich selbst zwei Mal aufruft. In jedem Aufruf hat jedoch das erste Argument (die Menge) ein Element weniger als vorher, so dass die Rekursion auf jeden Fall beendet wird.

In der drittletzten Zeile sieht man, wie man durch einen Backslash in Python eine Zeile „verlängern“ kann, wenn der Interpreter nicht von selbst erkennen würde, dass sie noch nicht zu Ende ist.

`\`

Lösung 313: Sei A eine Menge mit n Elementen. Man kann eine Sequenz der Länge n, die aus Nullen und Einsen besteht ($\{0,1\}$ ist eine zwei-elementige Menge), als eine Darstellung einer Teilmenge B von A interpretieren. Die k-te Komponente der Sequenz steht für das k-te Element von A: Ist sie eins, gehört dieses Element zu B, sonst nicht. Mit dieser Methode kann man Mengen als Zahlen (Sequenzen von Bits) „codieren“:

```
def setToBits (S):
    n = 0
    for x in S:
        n += 1 << x
```

```
    return n

def bitsToSet (n):
    S = set()
    x = 0
    while n > 0:
        if n % 2 == 1:
            S.add(x)
        x += 1
        n //= 2
    return S
```

`setToBits({0, 2, 3})` liefert z.B. $2^0 + 2^2 + 2^3 = 13$ und `bitsToSet(13)` liefert umgekehrt wieder `{0, 2, 3}`.

Lösung 316: Es handelt sich um Kombinationen ohne Wiederholungen. Die Anzahl der Möglichkeiten ist also $\binom{49}{6}$, was knapp 14 Millionen entspricht. Teilt man durch 520 (52 Wochen mit je 10 Tipps), so kommt man auf ca. 26 891 Jahre. Man muss in diesem Tempo also etwa 13 446 Jahre spielen, um die Hälfte aller möglichen Tipps einmal abgegeben zu haben.

Lösung 317: Da die Reihenfolge natürlich unerheblich ist, geht es erneut um Kombinationen ohne Wiederholungen, also ergeben sich $\binom{52}{5}$ Möglichkeiten – etwa 2.6 Millionen.

Lösung 318: Es gibt $5! = 120$ Permutationen der fünf Sehenswürdigkeiten. Da jedes Clubmitglied drei verschiedene Vorschläge abgegeben hat, kann es somit nicht mehr als 40 Mitglieder geben.

Lösung 319: Hier geht es um Kombinationen mit Wiederholungen. Es gibt also

$$\binom{3+12-1}{12} = \binom{14}{12} = 91$$

Möglichkeiten.

Lösung 320: Es handelt sich hierbei ebenfalls um Kombinationen mit Wiederholungen. (Es werden fünf von fünf Farben ausgewählt, die aber nicht verschieden sein müssen.) Daher gibt es $\binom{5+5-1}{5} = \binom{9}{5} = 126$ Möglichkeiten.

Lösung 321: Insgesamt hat man $2 \cdot 26 + 10 = 62$ Zeichen zur Verfügung. Ohne weitere Regeln könnte man also 62^n verschiedene Passwörter mit n Zeichen erzeugen. Es gibt jedoch $(62-10)^n$ Passwörter dieser Länge, in denen keine Dezimalziffer vorkommt, die also nicht zugelassen sind. Ebenso gibt es jeweils $(62-26)^n$ Möglichkeiten, ein Passwort mit n Zeichen zu generieren, ohne einen Groß- bzw. Kleinbuchstaben zu benutzen. Diese nicht erlaubten Kombinationen müssen abgezogen werden. Allerdings muss man (Inklusions-Exklusions-Prinzip!) darauf achten, welche Zeichenketten man ggf. mehrfach subtrahiert hat. Das sind die, in denen *zwei* Sorten von Zeichen nicht vorkommen, also z.B. die 10^n theoretisch möglichen Passwörter, in denen gar keine

Buchstaben vorkommen, oder die jeweils 26^n Kombinationen, in denen nur Groß- oder nur Kleinbuchstaben vorkommen. Man kommt so auf

$$62^n - (62-10)^n - 2 \cdot (62-26)^n + 10^n + 2 \cdot 26^n$$

legale Passwörter mit n Zeichen bzw.

$$\sum_{n=6}^{8} (62^n - (62-10)^n - 2 \cdot (62-26)^n + 10^n + 2 \cdot 26^n)$$

zugelassene Passwörter insgesamt. Das Ergebnis ist somit 162 042 094 456 320.

Lösung 322: (i) In einem Netzwerk der Klasse A stehen $32 - 8 = 24$ Bit für die Host-ID zur Verfügung, was 2^{24} verschiedene Möglichkeiten ergibt, von denen man noch zwei für Netzwerk bzw. Broadcast abziehen muss. Somit verbleiben 16 777 214 IDs. Für die Klassen B und C sind die Ergebnisse $2^{16} - 2 = 65\,534$ und $2^8 - 2 = 254$.

(ii) In der Klasse A stehen acht Bits für die Netzwerk-ID zur Verfügung. Davon ist aber das erste immer 0, so dass noch $2^7 = 128$ Möglichkeiten verbleiben. Analog erhält man für Klasse B $2^{16-2} = 16\,384$ und für Klasse C $2^{24-3} = 2\,097\,152$ Möglichkeiten.

(iii) Nach Produkt- und Summenregel ergibt sich aus den obigen Überlegungen als Gesamtzahl:

$$128 \cdot 16\,777\,214 + 16\,384 \cdot 65\,534 + 2\,097\,152 \cdot 254 = 3\,753\,869\,056$$

Es ist also relativ offensichtlich, dass das auf die Dauer nicht reichen wird, und deshalb wird ja auch auf IPv6 umgestellt.

Lösung 323: Hier geht es um Variationen mit Wiederholungen. Es gibt $6^4 = 1\,296$ Möglichkeiten.

Lösung 324: Jedes der neun Pixel kann schwarz oder weiß sein; es gibt also insgesamt 2^9 verschiedene Muster. (Man kann das als Variationen mit Wiederholung interpretieren – es wird neun Mal nacheinander jeweils schwarz oder weiß ausgewählt – oder als Anzahl der Teilmengen einer neun-elementigen Menge.) Man muss aber noch das eine Muster abziehen, bei dem alle Pixel weiß sind, und ebenso die neun Muster, bei denen genau ein Pixel schwarz ist. Und dieselbe Zahl von Mustern (also 10) muss man ein zweites Mal abziehen für die Muster bei denen zu wenige Pixel weiß sind. Somit ergibt sich als Antwort $2^9 - 2 \cdot (1 + 9) = 492$.

Lösung 325: Bei sechs Kästchen gibt es $2^6 = 64$ verschiedene Möglichkeiten zu antworten. (Anzahl der Teilmengen einer sechselementigen Menge.) Drei davon wurden durch die Aufgabenstellung ausgeschlossen, so dass maximal 61 Personen teilgenommen haben können.

Lösung 326: Jede Verteilung der vier Spielsteine entspricht einer Auswahl von vier von 16 Feldern. Dafür gibt es $\binom{16}{4} = 1\,820$ Möglichkeiten. Die sind aber nicht alle erlaubt. Nicht erlaubt ist es, alle vier Spielsteine auf den Ecken zu platzieren. Dafür gibt es nur eine Möglichkeit. Es ist aber auch nicht erlaubt, drei Spielsteine auf den

Ecken zu platzieren. Es gibt vier Möglichkeiten, von den vier Ecken drei auszuwählen. Und zu jeder dieser vier Möglichkeiten gibt es $16 - 4 = 12$ Möglichkeiten, den vierten Stein zu platzieren. Damit ergeben sich also $1\,820 - 1 - 4 \cdot 12 = 1\,771$ erlaubte Verteilungen.

Lösung 327: Die Zahlen 42, 23 und 10 sind irrelevant. Es reicht, dass es genug Bären von der jeweiligen Farbe gibt. Entscheidend ist das grüne Bärchen: entweder man nimmt es, oder man nimmt es nicht. Wenn man das grüne Bärchen nimmt, kann man sich noch vier weitere Bärchen nehmen und dabei unter drei Farben auswählen. Dafür gibt es $\binom{3+4-1}{4} = 15$ Möglichkeiten. (Es handelt sich um Kombinationen mit Wiederholungen.) Wenn man das grüne Bärchen *nicht* nimmt, kann man sich fünf Bärchen nehmen und dabei ebenfalls unter drei Farben auswählen. Dafür gibt es $\binom{3+5-1}{5} = 21$ Möglichkeiten. Zusammen hat man also $15 + 21 = 36$ Möglichkeiten.

Lösung 328: Die Zahlen, die in Frage kommen, müssen einem der Muster *ggg*, *ggu*, *gug*, *ugg*, *gg* oder *g* entsprechen, wobei natürlich *g* für eine gerade Ziffer und *u* für eine ungerade steht. Andererseits ist jede Zahl, die so einem Muster entspricht, eine gültige Wahl, und sowohl für *g* als auch für *u* gibt es im Prinzip jeweils fünf Möglichkeiten. Ist die erste Position des Musters jedoch ein *g*, so gibt es dort nur vier Möglichkeiten, sonst würde man die Null mitzählen. Somit ergibt sich insgesamt:

$$3 \cdot 4 \cdot 5 \cdot 5 + 5 \cdot 5 \cdot 5 + 4 \cdot 5 + 4 = 17 \cdot 25 + 20 + 4 = 449$$

Lösung 329: Für die erste Ziffer gibt es neun Möglichkeiten, weil es ja keine Null sein darf. Die zweite Ziffer kann irgendeine andere Ziffer sein, nur nicht die erste. Dafür gibt es auch neun Möglichkeiten, weil ja nun die Null erlaubt ist. Für die dritte Ziffer gibt es dann noch acht Möglichkeiten, weil zwei Ziffern schon verwendet wurden. Das ergibt insgesamt $9 \cdot 9 \cdot 8 = 648$ Möglichkeiten.

Lösung 330: In jeder Spalte stehen drei verschiedene Ziffern; für jede Spalte gibt es daher 3! Permutationen. Insgesamt ergeben sich so $(3!)^3 = 216$ Möglichkeiten.

Ist man nur an der Anzahl der unterschiedlichen Mengen von drei Summanden interessiert, so muss man eine der Spalten „festhalten“. Man kommt dann auf $(3!)^2 = 36$ mögliche Summen.

Lösung 332: Die Funktion sieht fast so wie `Nat` aus:

```
def evenNat ():
    n = 0
    while True:
        yield n
        n += 2
```

Lösung 333: Bei den beiden bisherigen Beispielen war es so, dass man die unendlichen Mengen einfach „der Reihe nach“ aufzählen konnte. Das lag auch daran, dass es einen klar definierten Anfang gab. $\mathbb{Z}$ hat so einen Anfang nicht, man kann aber den sich in beide Richtungen unendlich erstreckenden Zahlenstrahl „umklappen“ und abwechselnd z.B. positive und negative Zahlen aufzählen:

```
def Ints ():
    yield 0
    n = 1
    while True:
        yield n
        yield -n
        n += 1
```

Lösung 334: Das klappt offensichtlich im Prinzip mit jeder endlichen Menge:

```
def FinEx ():
    yield 23
    yield 42
    yield 101
```

Lösung 335: Für `perms` (Seite 180) könnte die modifizierte Version so aussehen:

```
def perms (L):
    if len(L) <= 1:
        yield L
    else:
        for i in range(len(L)):
            for P in perms(L[:i] + L[i+1:]):
                yield [L[i]] + P
```

Der Vorteil dieser Version ist, dass Sie, wenn Sie durch eine Liste aller Permutationen iterieren wollen, diese Liste nicht erst komplett generieren und zwischenspeichern müssen. (Das kostet Speicherplatz und Zeit.) Stattdessen werden Ihnen die einzelnen Permutationen dann geliefert, wenn Sie sie brauchen. (Siehe auch Seite 233 sowie Aufgabe 386.)

Lösung 336: Diese Funktion zählt alle Strings mit den erlaubten Buchstaben der Länge n auf (und ruft sich dabei selbst rekursiv auf):

```
def stringsOfLen (n):
    if n == 0:
        yield ""
    else:
        for p in range(ord("A"), ord("Z") + 1):
            for s in stringsOfLen(n - 1):
                yield chr(p) + s
```

Beachten Sie, dass die Strings nicht „irgendwie“ ausgegeben werden, sondern sortiert. Sie werden in der Reihenfolge erzeugt, in der sie auch im Telefonbuch stehen würden. Das ist die sogenannte *lexikographische Ordnung*, die in der Mathematik, aber auch in der Informatik häufig verwendet wird. Sämtliche Strings bekommt man dann so:

```
def allStrings ():
    n = 0
    while True:
        for s in stringsOfLen(n):
            yield s
        n += 1
```

Lösung 337: Analog zu Aufgabe 336 kann man zunächst eine Funktion schreiben, die alle Strings aufzählt, die man überhaupt in PYTHON eingeben kann. (Es sind zwar nicht nur die Buchstaben von `A` bis `Z` zugelassen, aber im Endeffekt handelt es sich auch nur um einen endlichen Zeichenvorrat.) Dann schreibt man „nur“ noch eine Funktion, die für jeden dieser Strings prüft, ob er legaler PYTHON-Code ist, der eine Funktion definiert. Das ist zwar aufwendig, aber sicher nicht unmöglich, da Ihr PYTHON-Interpreter das ja auch kann.

Lösung 338: So könnte es aussehen:

```
def nthInt (n):
    return (n + 1) // 2 if n % 2 == 1 else -n // 2
```

Ohne Fallunterscheidung gibt es viele Möglichkeiten. Eine von denen sieht so aus:

```
def nthInt (n):
    return (-1)**(n+1) * ((n+1)//2)
```

Sie funktioniert folgendermaßen:

n	$(n+1)/2$	$\lfloor (n+1)/2 \rfloor$	$(-1)^{n+1} \cdot \lfloor (n+1)/2 \rfloor$
0	$1/2$	0	0
1	1	1	1
2	$3/2$	1	-1
3	2	2	2
4	$5/2$	2	-2
5	3	3	3
6	$7/2$	3	-3

Lösung 339: Jeder positive Bruch kommt in dieser Aufzählung unendlich oft vor. Z.B. kommt der Bruch $3/5$ später wieder als $6/10$ und dann erneut als $9/15$ und dann noch mal als $12/20$ und so weiter vor.

Lösung 340: Das kann man so wie in Aufgabe 333 machen:

```
def Rat3 ():
    yield Fraction(0)
    seen = set()
    c = 2
```

```
    while True:
        den = 1
        while den < c:
            val = Fraction(c - den, den)
            if not val in seen:
                yield val
                yield -val
                seen.add(val)
            den += 1
        c += 1
```

Lösung 341: Wenn man einen Bruch kürzen kann, so haben Zähler und Nenner einen Teiler gemeinsam. Z.B. kann man 4/6 zu 2/3 kürzen, weil 4 und 6 den gemeinsamen Teiler 2 haben. Die gekürzten Brüche sind also genau die, bei denen Zähler und Nenner teilerfremd sind.

Lösung 342: Mithilfe von `convDecToBin` (siehe Aufgabe 33) sieht es so aus:

```
from fractions import Fraction

def CalkinWilf (n):
    num = 1
    den = 1
    for i in convDecToBin(n)[1:]:
        if i == 0:
            den = num + den
        else:
            num = num + den
    return Fraction(num, den)
```

Lösung 343: Das ist nun natürlich ganz einfach:

```
def CalkinWilfEnum ():
    c = 1
    while True:
        yield CalkinWilf(c)
        c += 1
```

Lösung 344: Wir benutzen in diesem Fall `convBinToDec` von Seite 24.

```
def CalkinWilfReverse (num, den):
    digits = []
    while num != den:
        if num > den:
            num = num - den
```

```
            digits.append(1)
        else:
            den = den - num
            digits.append(0)
    digits.append(1)
    return convBinToDec(list(reversed(digits)))
```

Lösung 345: $[3,3]$ ist dasselbe wie $\{3\}$. $[3,3)$ und $[4,3]$ sind zwei verschiedene Schreibweisen für die leere Menge $\varnothing$.

Lösung 346: Die wahren Aussagen sind blau, die falschen rot markiert:

$$\sqrt{2} \in [1,2] \qquad \sqrt{2} \in (0,3]$$
$$\sqrt{2} \in [-1,1] \qquad 42 \in (42,43]$$
$$42 \in [42,43) \qquad 42 \in (42,43)$$
$$42 \in [41.99, 42.01] \qquad 42 \in (41.999, 42.001)$$
$$[1,2] \cap [2,3] = \{2\} \qquad [1,2] \cap (2,3] = \{2\}$$
$$[0,100] \cap (2,3] = (2,3] \qquad [1,3) \cap (2,4] = (1,4)$$
$$[42,43] \setminus (43,44] = [42,43] \qquad [42,43) \setminus [43,44] = [42,43]$$
$$[\sqrt{2}, \pi] \subseteq (-1/2, 39/10) \qquad [-2,2) \cup (2,\infty) = [-2,\infty)$$
$$\mathbb{R}_{\geq 0} \cap (-\infty, 0) = \{0\} \qquad [0,\infty) \cup (-\infty, 0) = \mathbb{R} \setminus \{0\}$$
$$[0,\infty) \cap (-\infty, 0] = \varnothing \qquad [42,\infty) \cap (42,100] = [42,100]$$

Lösung 347: Das sind die richtigen Zuordnungen:

A	B	C	D	H
E	I	G	F	J
$[6,8]$	$\{8\}$	$[6,11]$	$(8,11]$	$\{6,8\}$

Lösung 350: Die Funktion sieht fast so aus wie vorher. Es gibt nur eine winzige Änderung, die unten markiert wurde:

```
def viewGen (gen, n):
    c = 1
    L = []
    for x in gen():          # <- HIER
        if c > n:
            break
        L.append(x)
        c += 1
    return L
```

Lösung 351: Es geht um diese Zahl:

$$0.1010010001000010000010000001000000010000000001\ldots$$

Offenbar kommt eine Eins vor an der ersten Stelle, dann an der dritten, dann an der sechsten, dann an der zehnten, etc. Das liegt natürlich daran, dass wir die folgenden Beziehungen haben:

$$\begin{aligned} 1 &= 1 \\ 3 &= 1+2 \\ 6 &= 1+2+3 \\ 10 &= 1+2+3+4 \end{aligned}$$

Nach der Gaußschen Summenformel (Seite 171) kommen Einsen also an den Stellen $1/2 \cdot k(k+1)$ für $k = 1, 2, 3, \ldots$ vor. Die Funktion testet, ob die eingegebene Zahl n von dieser Form ist. Dazu multipliziert sie erst n mit zwei und zieht dann die Wurzel. Ist n so eine Zahl, dann muss die Wurzel von $2n = k(k+1)$ offensichtlich zwischen k und $k+1$ liegen. Dann wäre k der ganzzahlige Anteil der Wurzel (das macht die Funktion `int`), und man könnte n nach der obigen Formel rekonstruieren.

Lösung 352: Nein, das kann nicht sein. Wenn eine Zahl zwei verschiedene Darstellungen im Dezimalsystem hat, dann endet die eine Darstellung mit unendlich vielen Neunen und die andere mit unendlich vielen Nullen. Die von `mystery` konstruierte Zahl hat als Nachkommastellen aber ausschließlich Vieren und Fünfen.

Lösung 353: $g(3) = 23$, weil $(3, 23)$ ein Element von g ist.

Lösung 354: Rechtseindeutigkeit bedeutet, dass von keinem Element des Definitionsbereiches zwei oder mehr Pfeile ausgehen.

Lösung 355: A ist eine Funktion, denn alle Elemente sind geordnete Paare. Und wenn m und n gleich sind, dann sind natürlich auch die Werte $2m+1$ und $2n+1$ gleich. A ist ein „klassisches“ Beispiel für eine Funktion. Sie ordnet jeder natürlichen Zahl n den Wert $2n+1$ zu. B ist zwar eine Menge von geordneten Paaren, aber keine Funktion, denn zu B gehören z.B. die beiden Elemente $(0, 1)$ und $(0, -1)$. B ist also nicht rechtseindeutig.[40] C hat nicht nur geordnete Paare als Elemente, sondern auch die Zahl 3. Daher ist C keine Funktion. D ist eine Funktion. E aber nicht, weil die Rechtseindeutigkeit verletzt ist, denn sowohl $(1, 2)$ als auch $(1, 3)$ gehören zu E.

Es wird Sie vielleicht überraschen, aber durch die streng formale Brille eines Mathematikers gesehen ist die leere Menge eine Funktion. Sie ist eine Menge von geordneten Paaren, weil sie kein Element hat, das *nicht* ein geordnetes Paar ist. Und weil sie gar keine Elemente hat, kann sie auch nicht *nicht* rechtseindeutig sein...

Lösung 356: Man könnte D so darstellen:[41]

$$D : \begin{cases} \{1, 3, 4\} \to \mathbb{N} \\ n \mapsto 2 \cdot \lceil \sqrt{n} \rceil + 1 \end{cases}$$

Aber natürlich gibt es unendlich viele andere Möglichkeiten, die Aufgabe zu lösen. Ihrer Phantasie sind da keine Grenzen gesetzt.

[40] B ist die Menge aller Punkte des *Einheitskreises*.

[41] Die hier benutzten Gaußklammern kamen im Buch schon mal vor. Falls Sie das vergessen haben, wäre das jetzt eine gute Gelegenheit, sich mit dem Index vertraut zu machen...

Lösung 357: `onlyTuples` testet, ob das Argument eine Menge (oder auch Liste) von *Tupeln* ist. Wir müssen aber eigentlich testen, ob wir es mit *Paaren* (also 2-Tupeln) zu tun haben. Man sollte `onlyTuples` durch so eine Funktion ersetzen:

```
def onlyPairs (s):
    return all(map(lambda x: type(x) is tuple and len(x) == 2, s))
```

Lösung 358: Das ist natürlich ganz einfach:

```
def rng (s):
    return map(lambda t: t[1], s)
```

Lösung 359: Die Funktion ist nicht injektiv, weil sowohl 4 als auch 3 auf 23 abgebildet werden. Anschaulich ist eine Funktion dann injektiv, wenn auf kein Element der Zielmenge mehr als ein Pfeil zeigt.

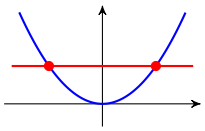
Lösung 360: Nein, weil $f(1) = f(3)$ gilt.

Lösung 361: Nein, weil z.B. $f(3.2) = f(3.3)$ gilt.

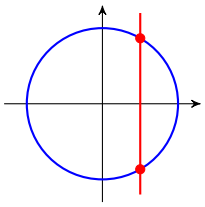
Lösung 362: Das Ergebnis hat sehr viel Ähnlichkeit mit `rightUnique`:

```
def injective (f):
    for x1, y1 in f:
        for x2, y2 in f:
            if x1 != x2 and y1 == y2:
                return False
    return True
```

Lösung 363: Wenn eine Funktion *nicht* injektiv ist, dann kann man eine horizontale Linie finden, die den Funktionsgraphen mehr als einmal schneidet. Hier ein Beispiel für die *Normalparabel* $x \mapsto x^2$:



Wenn ein „Funktionsgraph“ von einer *vertikalen* Linie mehr als einmal getroffen wird, dann ist es gar kein Funktionsgraph, weil die dargestellte Menge von Paaren nicht rechtseindeutig, also keine Funktion, ist. Hier ein Beispiel für den Einheitskreis aus Aufgabe 355:



Lösung 364: Ganz einfach, weil man ja nur die Tupel umdrehen muss:

```
def invRel (f):
    return set(map(lambda t: (t[1], t[0]), f))
```

Wenn man `invRel` hat, kann man `injective` aus Aufgabe 362 noch kürzer schreiben:

```
def injective (f):
    return rightUnique(invRel(f))
```

Lösung 365: Aus $y = x/3+3$ wird $x = y/3+3$. Auflösen nach y liefert die Abbildung $x \mapsto 3x-9$.

Lösung 366: Die nach y aufzulösende Gleichung wäre $x = y^2$. Für diese Gleichung gibt es aber entweder gar keine Lösung (wenn x negativ ist) oder zwei (wenn x positiv ist). Daher ergibt sich so keine sinnvolle Rechenvorschrift. Betrachtet man aber eine Funktion mit eingeschränktem Definitionsbereich wie z.B. diese:

$$f : \begin{cases} \mathbb{R}_{\geq 0} \to \mathbb{R} \\ x \mapsto x^2 \end{cases},$$

so kann man die Gleichung eindeutig lösen, weil man weiß, dass sowohl x als auch y nicht negativ sein können. Als Umkehrfunktion erhält man dann das hier:

$$f^{-1} : \begin{cases} \mathbb{R}_{\geq 0} \to \mathbb{R} \\ x \mapsto \sqrt{x} \end{cases}$$

Lösung 367: Mit dem Hinweis sollte es eigentlich ganz einfach sein. Eine Funktion bildet genau dann surjektiv auf B ab, wenn B der Wertebereich von f ist.

```
def surjective (f, B):
    return set(rng(f)) == B
```

Lösung 368: Nein. Das kann man schon am Beispiel direkt vor der Aufgabenstellung erkennen. $f \circ g$ ergibt die Abbildungsvorschrift $x \mapsto 3x+6$. $g \circ f$ hingegen würde $x \mapsto 3x+2$ ergeben.

Lösung 369: Die erste Beziehung gilt. Man muss dafür nur einen Wert x in die Funktionen einsetzen und nach Definition auflösen:

$$\begin{aligned}((f+g)\circ h)(x) &= (f+g)(h(x)) = f(h(x)) + g(h(x)) \\ &= (f\circ h)(x) + (g\circ h)(x) = ((f\circ h)+(g\circ h))(x)\end{aligned}$$

Die zweite Beziehung gilt allerdings nicht. Wenn wir für g und h z.B. jeweils die Funktion wählen, die einfach x auf x abbildet (die sogenannte *Identität*), und für f die durch $x \mapsto x^2$ definierte Funktion, so gilt z.B.:

$$(f \circ (g+h))(1) = f(1+1) = 4$$

$$((f \circ g) + (f \circ h))(1) = 1 + 1 = 2$$

Die beiden Funktionen sind also nicht gleich.

Lösung 371: Diese Funktion berechnet $f \circ g$:

```
def comp (f, g):
    result = set()
    for x, y1 in g:
        for y2, z in f:
            if y1 == y2:
                result.add((x, z))
                break
    return result
```

Lösung 372: Dass die Komposition von Funktionen *nicht* kommutativ ist, sieht man z.B. daran, dass das hier `False` ergibt:

```
compPy(f, g)(1) == compPy(g, f)(1)
```

Lösung 373: In Python haben wir ja schon die ganze Zeit mit mehrstelligen Funktionen gearbeitet. Darum sollte das kein Problem sein:

```
from math import sqrt

def f (m, n):
    return 2 * m + n
def g (x, y):
    return x * x * sqrt(y)
```

Lösung 374: f ist nicht injektiv. Z.B. gilt $f(20,2) = f(10,22)$. $(20,2)$ und $(10,22)$ sind also zwei verschiedene Elemente des Definitionsbereichs, die denselben Funktionswert haben. g ist auch nicht injektiv. Das kann man etwa mit $g(1,16) = g(2,1)$ begründen.

Lösung 375: Man könnte es z.B. folgendermaßen machen:

$$: \begin{cases} \mathbb{N}^2 \to \mathbb{N} \\ (m,n) \mapsto 2^m \cdot 3^n \end{cases}$$

Es ist nicht möglich, dass zwei unterschiedliche Paare (m_1, n_1) und (m_2, n_2) denselben Funktionswert haben, weil diese Zahl dann zwei verschiedene Zerlegungen in Primfaktoren hätte, was dem Fundamentalsatz der Arithmetik widersprechen würde.

Lösung 376: Für die Division. Man kann durch $(x,y) \mapsto x/y$ eine Verknüpfung auf $\mathbb{R} \setminus \{0\}$ definieren[42] oder alternativ eine Abbildung von $\mathbb{R} \times (\mathbb{R} \setminus \{0\})$ nach $\mathbb{R}$, aber keine Verknüpfung auf $\mathbb{R}$.

[42] Weil der Quotient von zwei Zahlen, die beide nicht null sind, nicht null ist.

Lösung 377: Die Addition ist eine Verknüpfung auf $\mathbb{N}$, weil die Summe von zwei natürlichen Zahlen wieder eine natürliche Zahl ist. Die Differenz ist das nicht, weil z.B. 2 und 3 natürliche Zahlen sind, $2-3$ aber kein Element von $\mathbb{N}$ ist.

Lösung 378: Ja, das geht immer. Nehmen Sie als Bijektion die Abbildung, die jedes Element von A auf sich selbst abbildet, also diese:

$$\mathrm{id}_A = \{(a,a) : a \in A\}$$

Diese Funktion nennt man die *Identität* von A.

Lösung 379: Durch die Umkehrfunktion f^{-1}. Unter anderem deswegen haben wir uns im Kapitel 19 so ausführlich mit Funktionen beschäftigt.

Lösung 380: Durch $n \mapsto n+1$ wird offenbar eine Bijektion von $\mathbb{N}$ auf $\mathbb{N}^+$ definiert:

0	1	2	3	4	5	6	7	8	9	10	11	
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	⋯
1	2	3	4	5	6	7	8	9	10	11	12	

Lösung 381: Das geht genau wie eben, nur dass wir diesmal 42 addieren.

0	1	2	3	4	5	6	7	8	9	10	11	
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	⋯
42	43	44	45	46	47	48	49	50	51	52	53	

Lösung 382: Alle endlichen Mengen sind abzählbar, insbesondere natürlich auch die leere Menge und alle Singletons. $\mathbb{N}$, $\mathbb{Z}$ und $\mathbb{Q}$ sind abzählbar und damit auch alle Teilmengen von diesen Mengen, also z.B. die Menge aller geraden Zahlen, die Menge der Primzahlen, die Menge der Brüche zwischen 42 und 43, und so weiter. Außerdem noch die Menge aller Zeichenketten, die man mit einem begrenzten Zeichenvorrat generieren kann. (Siehe Aufgabe 336.)

Und natürlich ist jede rekursiv aufzählbare Menge abzählbar. Rekursive Aufzählbarkeit ist eine stärkere Forderung als Abzählbarkeit. Wir werden im Verlauf dieses Kapitels noch sehen, dass es sogar sehr viele Mengen gibt, die abzählbar, aber nicht rekursiv aufzählbar sind.

Lösung 383: Man kann fast dieselbe Tabelle verwenden:

	0	1	2	3	4	5	6	⋯
0	(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	(0,6)	⋯
1	(1,0)	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	⋯
2	(2,0)	(2,1)	(2,2)	(2,3)	(2,4)	(2,5)	(2,6)	⋯
3	(3,0)	(3,1)	(3,2)	(3,3)	(3,4)	(3,5)	(3,6)	⋯
4	(4,0)	(4,1)	(4,2)	(4,3)	(4,4)	(4,5)	(4,6)	⋯
5	(5,0)	(5,1)	(5,2)	(5,3)	(5,4)	(5,5)	(5,6)	⋯
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱

Lösung 384: Das ist nun wirklich sehr einfach:

```
def singletons ():
    n = 0
    while True:
        yield {n}
        n += 1
```

Lösung 386: Auch hier fangen wir mit einer Funktion an, die wir so ähnlich (siehe Aufgabe 310) schon mal hatten:

```
def powerSet (A):
    if A == set():
        yield frozenset()
    else:
        a = A.pop()
        for X in powerSet(A):
            yield X
            yield X | {a}
```

Und dann machen wir eigentlich genau das, was wir bei fester Größe der Mengen auch gemacht haben:

```
def finiteSubsets ():
    k = 0
    seen = set()
    while True:
        for s in powerSet(set(range(k))):
            if not s in seen:
                yield s
                seen.add(s)
        k += 1
```

Lösung 387: Weil wir schon wissen, dass für jedes einzelne n die Menge der n-elementigen Teilmengen abzählbar ist. Die Menge aller endlichen Teilmengen ist dann einfach die Vereinigung dieser abzählbar vielen Mengen.

Lösung 390: Wenn $(0,1)$ abzählbar wäre, dann wäre $(0,1]$ als Vereinigung der beiden abzählbaren Mengen $(0,1)$ und $\{1\}$ auch abzählbar. $[0,1)$ lässt sich bijektiv auf $(0,1]$ abbilden, indem man 0 auf 1 abbildet und jede andere Zahl auf sich selbst. Also sind diese beiden Intervalle gleichmächtig. $[0,1]$ schließlich ist eine Obermenge von $(0,1]$. Wir wissen aber schon, dass Teilmengen von abzählbaren Mengen abzählbar sind. Also kann auch $[0,1]$ nicht abzählbar sein.

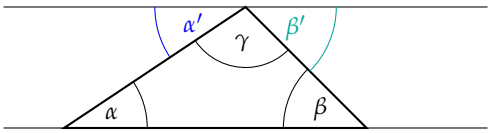
Lösung 391: Die Abbildung ist einfach eine Gerade von der Form $x \mapsto f(x) = mx + b$. Die korrekten Werte für die Steigung m und den Achsenabschnitt b erhält man, wenn man die gewünschten Werte $f(2) = 1$ und $f(5) = 2$ einsetzt:

$$f : \begin{cases} [2,5] \to [1,2] \\ x \mapsto x/3 + 1/3 \end{cases}$$

Lösung 392: Die Funktion `symbols` erzeugt ein oder mehrere SymPy-Symbole. In diesem Fall wird ein solches Symbol mit dem Namen h erzeugt. Und das wird dann in der Python-Variablen `g` gespeichert. Man muss hier unterscheiden zwischen dem *Symbol*, das ein Python-Objekt (also so etwas wie eine Zahl, ein String oder eine Liste) ist, und der Python-Variablen, in der dieses Symbol gespeichert wird. Solche Variablen haben wir schon das ganze Buch über benutzt.

Üblicherweise speichert man allerdings SymPy-Symbole in Variablen, die denselben Namen wie das Symbol haben, damit man gar nicht erst durcheinanderkommt. Wir werden das in diesem Buch auch so machen.

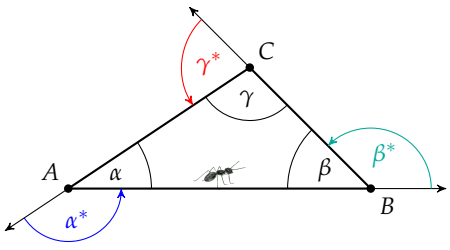
Lösung 394: Wir zeichnen zwei weitere Winkel in die Skizze ein:



Offensichtlich sind α und α' sowie β und β' Wechselwinkel und daher jeweils gleich groß. Zudem sieht man sofort, dass α', β' und γ sich zu 180° ergänzen. Somit gilt dann natürlich auch

$$\alpha + \beta + \gamma = \alpha' + \beta' + \gamma = 180°.$$

Es gibt natürlich noch andere Möglichkeiten, sich diese Tatsache klarzumachen. Hier eine Alternative: Stellen Sie sich eine Ameise vor, die einmal um das Dreieck herum läuft. Am Anfang befindet sie sich wie auf dem Bild unten auf der Strecke von A nach B und schaut in Richtung der Ecke B.



Am Punkt B muss sich die Ameise um den Winkel β^* drehen, dann geht sie weiter in Richtung C. Bei C dreht sie sich um γ^* und so weiter. Am Ende ihrer Reise ist sie wieder am Ausgangspunkt und schaut in dieselbe Richtung wie am Anfang. Daher muss sie sich insgesamt um 360° gedreht haben. Andererseits sind β und β^*

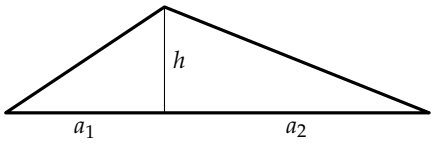
Ergänzungswinkel, d.h. $\beta + \beta^* = 180°$, und das gilt ebenso für die anderen beiden Winkelpaare. Die Gesamtdrehung setzt sich also so zusammen:

$$\begin{aligned} 360° &= \alpha^* + \beta^* + \gamma^* = (180° - \alpha) + (180° - \beta) + (180° - \gamma) \\ &= 540° - (\alpha + \beta + \gamma) \end{aligned}$$

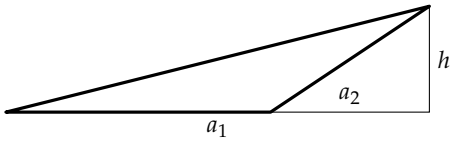
Aus dieser Beziehung folgt offenbar sofort $\alpha + \beta + \gamma = 180°$.

Lösung 396: Nach Pythagoras gilt $10^2 = 8^2 + b^2$. Es folgt $b^2 = 10^2 - 8^2 = 100 - 64 = 36$, also $b = 6$.

Lösung 397: Wir nennen die waagerechte Kathete des linken rechtwinkligen Dreiecks a_1 und die des rechten a_2. Die senkrechte Kathete hat bei beiden Dreiecken dieselbe Länge h.

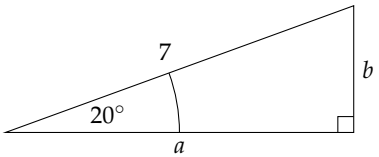


Somit hat das linke Dreieck die Fläche $a_1 h/2$ und für das rechte erhalten wir $a_2 h/2$. Als Fläche des großen Dreiecks bekommen wir daher $(a_1 + a_2)h/2$. Dabei ist $a_1 + a_2$ die Länge der sogenannten *Grundseite* und h die *Höhe* auf der Grundseite. Und das funktioniert auch dann noch, wenn die Höhe außerhalb des Dreiecks liegt:



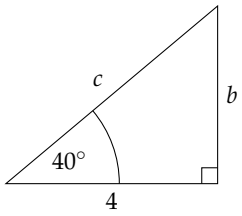
Hier ist a_1 die Kathete eines rechtwinkligen Dreiecks, das größer als das Dreieck ist, dessen Fläche wir wissen wollen. Um die gesuchte Fläche zu erhalten, müssen wir von der Fläche des großen rechtwinkligen Dreiecks die des kleinen (mit der Kathete a_2) subtrahieren. Das ergibt $(a_1 - a_2)h/2$ und das ist korrekt, weil $a_1 - a_2$ offenbar die Länge der Grundseite ist.

Lösung 398: Das Dreieck sollte ungefähr so aussehen:



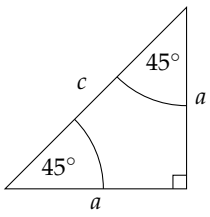
Mit $b/7 = \sin 20° \approx 0.342$ ergibt sich $b \approx 7 \cdot 0.342 = 2.394$. Ebenso erhält man $a = 7 \cos 20° \approx 7 \cdot 0.940 = 6.580$.

Lösung 399: Das Dreieck sieht in etwa folgendermaßen aus:



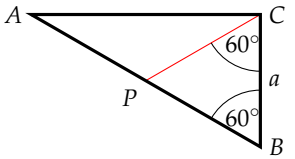
Wegen $b/4 = \tan 40° \approx 0.839$ erhalten wir $b \approx 4 \cdot 0.839 = 3.356$. Mit Pythagoras folgt nun $c = \sqrt{4^2 + b^2} \approx 5.221$.

Lösung 400: Da die Winkelsumme im Dreieck 180° beträgt, müssen *beide* nicht-rechten Winkel das Maß 45° haben, wenn einer von beiden dieses Maß hat. Daher müssen natürlich auch die beiden Katheten dieselbe Länge haben. Als Tangens ergibt sich damit sofort $a/a = 1$.



Mit Pythagoras erhält man für die Länge der Hypotenuse c nun $\sqrt{a^2 + a^2} = a\sqrt{2}$. Daher sind die Werte für den Sinus und den Kosinus von 45° beide $a/c = 1/\sqrt{2}$.

Lösung 401: Damit sich insgesamt 180 Grad ergeben, muss der dritte Winkel im Dreieck BCP ebenfalls das Maß 60° haben. Damit ist dieses Dreieck gleichseitig. Nennen wir die waagerechte Kathete a, so haben also sowohl die rote Strecke $\overline{PC}$ als auch $\overline{BP}$ die Länge a.



Der Winkel des Dreiecks CAP an der Ecke C beträgt 30°, weil bei C ja der rechte Winkel des Dreiecks ABC liegt. Außerdem ergibt sich (wieder wegen der Winkelsumme im Dreieck), dass der Winkel bei A ebenfalls das Maß 30° hat. Damit ist APC ein gleichschenkliges Dreieck und wir können folgern, dass $\overline{PA}$ dieselbe Länge wie $\overline{PC}$, also a, hat. Die Hypotenuse des rechtwinkligen Dreiecks hat somit die Länge $2a$ und als Kosinus von 60° ergibt sich $a/(2a) = 1/2$.

Jetzt können wir mit Pythagoras die Länge der zweiten Kathete ausrechnen und erhalten $\sqrt{(2a)^2 - a^2} = a\sqrt{3}$. Daher hat der Sinus von 60° den Wert $a\sqrt{3}/(2a) = \sqrt{3}/2$ und für den Tangens erhalten wir $a\sqrt{3}/a = \sqrt{3}$.

Lösung 402: Wir können das Dreieck von eben benutzen und müssen nur die Rollen von An- und Gegenkathete vertauschen. Damit erhalten wir:

$$\sin 30^\circ = 1/2 \qquad \cos 30^\circ = \sqrt{3}/2 \qquad \tan 30^\circ = 1/\sqrt{3}$$

Lösung 403: Man erhält

$$\cos 11.25^\circ = 1/2 \cdot \sqrt{2 + 2 \cdot 1/2 \cdot \sqrt{2+\sqrt{2}}} = 1/2 \cdot \sqrt{2+\sqrt{2+\sqrt{2}}}$$

Man kann hoffentlich ein Muster erkennen. Halbiert man erneut, so ergibt sich z.B.

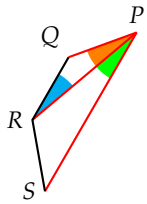
$$\cos 5.625^\circ = 1/2 \cdot \sqrt{2+\sqrt{2+\sqrt{2+\sqrt{2}}}}$$

und so geht es weiter.

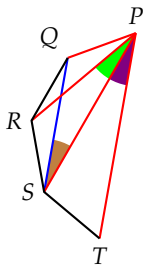
Lösung 404: Ein Fünfeck kann man in drei Dreiecke zerlegen. Da die Summe der Innenwinkel des Dreiecks 180° beträgt, ergibt sich für die entsprechende Summe im Fünfeck $3 \cdot 180^\circ = 540^\circ$.

Lösung 405: Mit jeder weiteren Ecke kann man ein weiteres Dreieck abschneiden. Bei fünf Ecken ergibt sich nach der letzten Aufgabe als Summe $(5-2) \cdot 180^\circ$, beim Sechseck dann $(6-2) \cdot 180^\circ$ und so weiter. Allgemein ist die Summe der Innenwinkel in einem Polygon mit n Ecken also $(n-2) \cdot 180^\circ$. (Natürlich ergibt diese Ausgabe nur dann Sinn, wenn die Seiten des Polygons sich nicht überschneiden.)

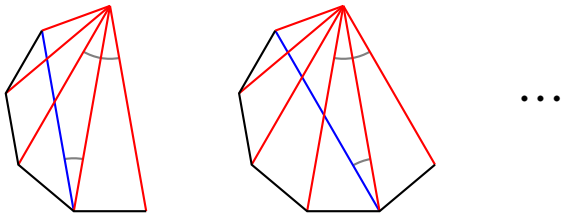
Lösung 406: Die Aussage gilt für alle regelmäßigen n-Ecke. Wir begründen das hier am Beispiel des Neunecks. Es wird sich allerdings zeigen, dass es für den Gedankengang keine Rolle spielt, wie viele Ecken das Polygon hat. Für die Bezeichnung von Winkeln wählen wir die Schreibweise $\measuredangle ABC$ für den Winkel am Punkt B zwischen den Strecken $\overline{BA}$ und $\overline{BC}$. Wir betrachten zunächst vier Ecken des Polygons und wollen uns überzeugen, dass die beiden Winkel $\measuredangle RPS$ und $\measuredangle QPR$ gleich groß sind:



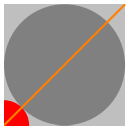
Da wir es mit einem regelmäßigen Polygon zu tun haben, sind die Strecken $\overline{PQ}$ und $\overline{RS}$ gleich lang und die Winkel, die diese beiden Strecken mit $\overline{QR}$ bilden, gleich groß. Daher sind $\overline{QR}$ und $\overline{SP}$ parallel und damit $\measuredangle RPS$ und $\measuredangle PRQ$ als Wechselwinkel gleich groß. Andererseits ist das Dreieck mit den Ecken R, P und Q aber gleichschenklig, daher muss $\measuredangle PRQ$ so groß wie $\measuredangle QPR$ sein. Das war's schon! Nun nehmen wir einen weiteren Punkt hinzu.



Wieder aus Symmetriegründen müssen $\overline{QS}$ und $\overline{TP}$ parallel sein. Daher sind $\measuredangle SPT$ und $\measuredangle PSQ$ als Wechselwinkel gleich groß. Wegen der Symmetrie ist $\measuredangle PSQ$ aber wiederum so groß wie der Winkel $\measuredangle RPS$, den wir oben behandelt haben. Dieses Spielchen können wir nun fortsetzen, um nach und nach die weiteren Winkel zu behandeln.



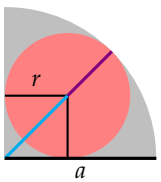
Lösung 407: Wir betrachten ein Viertel des Quadrats:



Wenn wir die Seitenlänge des ursprünglichen Quadrats a nennen, dann hat dieses kleine Quadrat die Seitenlänge $a/2$. Die orange Diagonale hat nach Pythagoras dann die Länge $a/\sqrt{2}$. (Rechnen Sie nach!) Da der Durchmesser des grauen Kreises sicher ebenfalls $a/2$ ist und der Radius des roten Kreises offenbar die Hälfte dessen ausmacht, was übrigbleibt, wenn man von der Diagonale den Durchmesser entfernt, erhält man diese Lösung:

$$\frac{1}{2}\cdot\left(\frac{a}{\sqrt{2}}-\frac{a}{2}\right)=\frac{\sqrt{2}-1}{4}\cdot a$$

Lösung 408: Für die erste Teilaufgabe schauen wir uns das obere rechte Viertel an und nennen den Radius des grauen Kreises a.



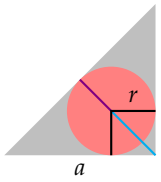
Nennen wir den gesuchten Radius des roten Kreises r, so ist die Länge des cyanfarbenen Streckenstücks in der obigen Skizze nach Pythagoras $\sqrt{2}r$ und die Länge des violetten Stücks r. Die beiden Strecken zusammen ergeben offenbar a:

$$a = r + \sqrt{2}r = (1+\sqrt{2})r$$

Somit ergibt sich:[43]

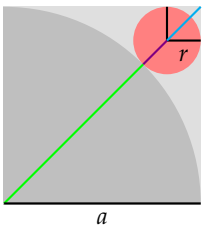
$$r = \frac{a}{1+\sqrt{2}} = \frac{a}{1+\sqrt{2}} \cdot \frac{1-\sqrt{2}}{1-\sqrt{2}} = a(\sqrt{2}-1)$$

Wenn man das Prinzip der obigen Lösung verstanden hat, dann stellt man fest, dass sich die anderen beiden Teilaufgaben sehr ähnlich lösen lassen. Für den zweiten Teil nennen wir die Seitenlänge des Quadrats a und den gesuchten Radius wieder r.



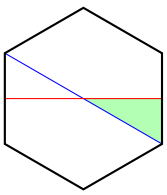
Hier setzen wir die Hälfte der Diagonalen des Quadrats aus zwei Teilen zusammen: $a/\sqrt{2} = r + \sqrt{2}r = (1+\sqrt{2})r$. Auflösen nach r liefert $r = a/(2+\sqrt{2})$.

Im dritten Fall benennen wir wie folgt:



Wir erhalten $\sqrt{2}a = \sqrt{2}r + r + a$ bzw. $a(\sqrt{2}-1) = r(\sqrt{2}+1)$. Löst man nach r auf und vereinfacht, so ergibt sich $r = a(3-2\sqrt{2})$.

Lösung 409: Wie vor der Aufgabe erwähnt, empfiehlt es sich, ein rechtwinkliges Dreieck zu finden, von dem man schon ein paar Eigenschaften kennt. Ein solches wurde hier grün eingezeichnet:

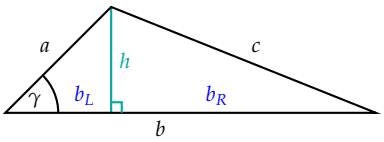


[43] Man entfernt die Wurzel aus dem Nenner (und vereinfacht so den Bruch), indem man so erweitert, dass man die dritte binomische Formel anwenden kann. Diesem Trick werden wir erneut begegnen, wenn wir uns mit komplexen Zahlen beschäftigen.

Wir wissen nach Aufgabenstellung, dass die (blaue) Hypotenuse die Länge 2 hat. Außerdem muss der Winkel links das Maß 30° haben, da man offenbar zwölf solche Dreiecke im Sechseck unterbringen könnte und da dieses regelmäßig ist. Die rote Kathete hat daher die Länge $2 \cdot \cos 30^\circ = 2 \cdot \sqrt{3}/2 = \sqrt{3}$. Die Antwort ist somit $2\sqrt{3}$.

Alternativ kann man argumentieren, dass man sechs gleichseitige Dreiecke erhält, wenn man den Mittelpunkt des Sechsecks mit den sechs Ecken verbindet. Daher muss die Länge der Seiten des Sechsecks die Hälfte von 4 (Länge der blauen Linie) sein. Man kennt dann zwei Seiten des grünen Dreiecks (außer der Hypotenuse noch die schwarze Kathete mit der Länge 1) und kann die Dritte mit Pythagoras und ohne Kenntnis der Winkel berechnen.

Lösung 410: Wir zerlegen das Dreieck folgendermaßen in zwei rechtwinklige:



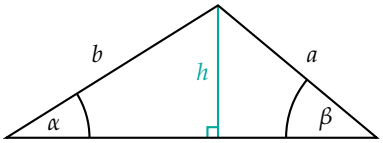
Für die Länge von c gilt nach Pythagoras im rechten Dreieck:

$$c^2 = h^2 + {b_R}^2 \tag{A.2}$$

Die Seite b_R ergibt sich als Differenz $b - b_L$. Im linken Dreieck erhält man $b_L = a \cdot \cos\gamma$ und für h gilt nach Pythagoras $h^2 = a^2 - {b_L}^2$. Setzt man in Gleichung (A.2) ein, so ergibt sich:

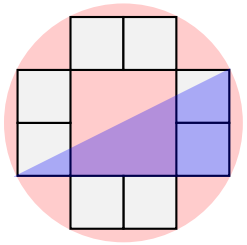
$$\begin{aligned} c^2 &= h^2 + {b_R}^2 = a^2 - {b_L}^2 + (b - b_L)^2 \\ &= a^2 - {b_L}^2 + b^2 - 2bb_L + {b_L}^2 = a^2 + b^2 - 2bb_L \\ &= a^2 + b^2 - 2ab\cos\gamma \end{aligned}$$

Lösung 411: Wir zerlegen wie folgt:



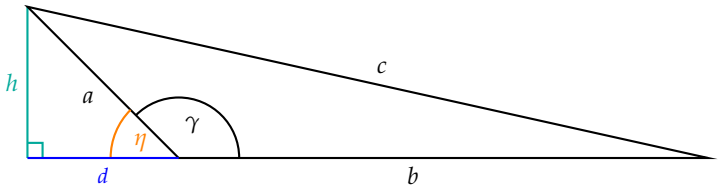
Im linken rechtwinkligen Dreieck gilt dann $\sin\alpha = h/b$, im rechten $\sin\beta = h/a$. Dividiert man $\sin\alpha$ durch $\sin\beta$, so kürzt sich h heraus und man erhält sofort die gesuchte Beziehung.

Lösung 412: Das rechtwinklige Dreieck, das uns in diesem Fall hilft, wurde hier blau eingezeichnet:



Da der Kreis die Fläche π hat, hat sein Durchmesser die Länge 2 und das ist die Hypotenuse des blauen Dreiecks. Nennt man die Seitenlänge eines Quadrats a, so ergibt sich mit Pythagoras $(2a)^2 + (4a)^2 = 2^2$ und damit $a = 1/\sqrt{5}$. Die Fläche eines Quadrats ist also $1/5$ und die aller acht Quadrate daher $8/5$.

Lösung 413: Wir fügen ein rechtwinkliges Dreieck hinzu:



Im großen rechtwinkligen Dreieck mit der Hypotenuse c gilt $c^2 = h^2 + (b+d)^2$. Im hinzugefügten rechtwinkligen Dreieck lesen wir $d = a\cos\eta$ ab und mit Pythagoras außerdem $h^2 = a^2 - d^2$. Fügen wir alles zusammen, so erhalten wir:

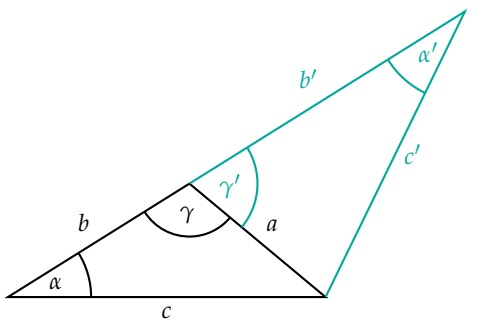
$$\begin{aligned} c^2 = h^2 + (b+d)^2 &= a^2 - d^2 + b^2 + 2bd + d^2 \\ &= a^2 + b^2 + 2bd = a^2 + b^2 + 2ab\cos\eta \end{aligned} \tag{A.3}$$

Wegen $\eta + \gamma = \pi$ folgt aber $\cos\eta = \cos(\pi - \gamma) = -\cos\gamma$, so dass aus (A.3) folgt:

$$c^2 = a^2 + b^2 - 2ab\cos\gamma \tag{A.4}$$

Das ist dieselbe Formel, die wir in Aufgabe 410 bereits für spitze Winkel ermittelt hatten; sie gilt also für *alle* Winkel. (Insbesondere ergibt sich für den Spezialfall, dass γ ein rechter Winkel ist, der Satz des Pythagoras, weil $\cos\gamma$ dann null ist.) Man nennt Aussage A.4 den *Kosinussatz*. Nebenbei sieht man, wie sinnvoll es war, den Definitionsbereich des Kosinus auf Winkel auszudehnen, die nicht spitz sind.

Lösung 414: Wir konstruieren ein am ursprünglichen Dreieck anliegendes Dreieck derart, dass b' die Verlängerung von b und c' genauso lang wie c ist:

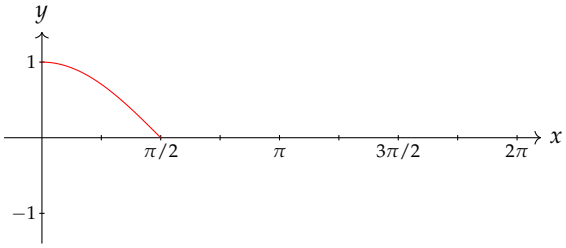


Da γ stumpf ist, muss γ' spitz sein. Im grünen Dreieck gilt also der in Aufgabe 411 bewiesene Sinussatz und damit haben wir:

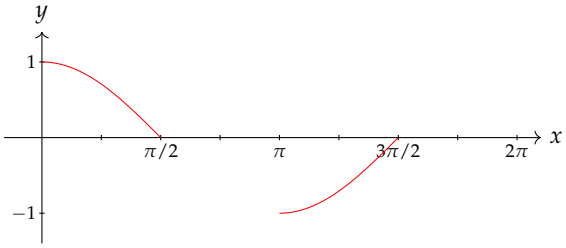
$$\frac{\sin\alpha'}{\sin\gamma'} = \frac{a}{c'} = \frac{a}{c} \tag{A.5}$$

Da das gesamte, sich von α nach α' erstreckende Dreieck gleichschenklig ist, muss $\alpha = \alpha'$ gelten. Außerdem gilt $\gamma + \gamma' = \pi$ und daher $\sin\gamma' = \sin(\pi - \gamma) = \sin\gamma$. Setzt man das beides in (A.5) ein, so erhält man die gewünschte Beziehung.

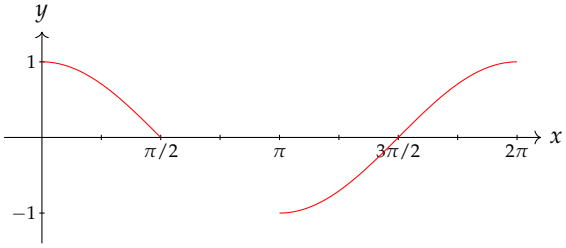
Lösung 415: Die bekannten Werte sind diese (wobei wir in dieser Lösung und der folgenden den Kosinus immer rot und den Sinus immer blau zeichnen):



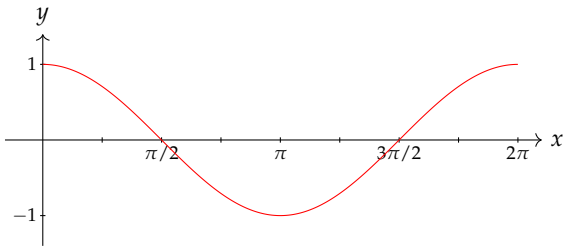
Mit $\cos(x+\pi) = -\cos x$ hat man schon mal die Kosinuswerte auf $[\pi, 3\pi/2]$, wenn man für x Werte von 0 bis $\pi/2$ einsetzt:



Mit $\cos(2\pi - x) = \cos(-x) = \cos x$ bekommt man nun die Werte auf $[3\pi/2, 2\pi]$, wenn man für x Werte von 0 bis $\pi/2$ einsetzt:



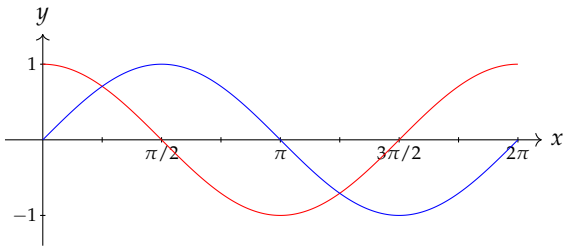
Und $\cos(x-\pi) = -\cos x$ liefert den Rest, wenn x von $3\pi/2$ bis 2π wandert:



Da der Kosinus 2π-periodisch ist, kennt man nun *alle* Werte. Daher kann man die erste Funktion schreiben:[44]

```
def myCos (x):
    if 0 <= x <= pi/2:
        return restrictedCos(x)
    if pi <= x <= 3*pi/2:
        return -myCos(x - pi)
    if 3*pi/2 <= x <= 2*pi:
        return myCos(2*pi - x)
    if pi/2 <= x <= pi:
        return -myCos(x + pi)
    if x <= 0:
        return myCos(x + 2*pi)
    return myCos(x - 2*pi)
```

Wegen $\sin(x + \pi/2) = \cos x$ hat man dann natürlich auch alle Sinuswerte:



Daher sieht die zweite Funktion so aus:

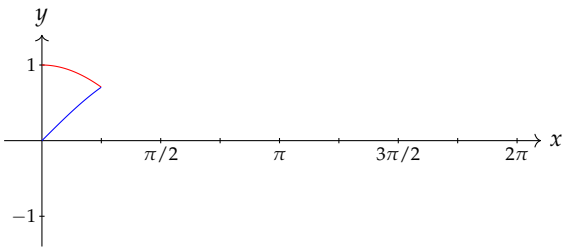
```
def mySin (x):
    return myCos(x - pi/2)
```

Wenn Sie testen wollen, ob Ihr eigener Lösungsversuch korrekt war, bietet es sich an, sich die Funktion zeichnen zu lassen (siehe Kapitel 41):

[44] Das ist so natürlich nicht besonders effizient. Darum geht's hier aber nicht.

```
from plot import *

plotFunc2D(myCos, [-pi, 3*pi])
```

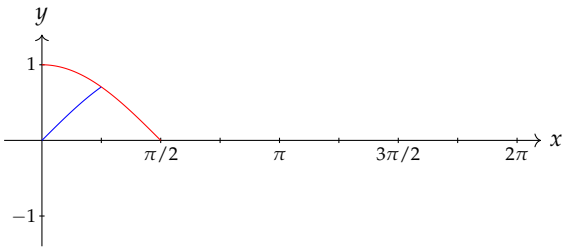
Lösung 416: Hier ist man mit dieser Situation konfrontiert:



Mit der Beziehung

$$\begin{aligned}\cos(x+\pi/4) &= \cos(-x-\pi/4) = \sin((-x-\pi/4)+\pi/2)\\ &= \sin(-x+\pi/4)\end{aligned}$$

kann man nun ein weiteres Stück Kosinus erreichen und ist so weit wie in der vorherigen Aufgabe:



Man muss die vorherige Lösung also nur leicht modifizieren. Außer den drei markierten Zeilen hat sich nichts geändert:

```
def myCos (x):
    if 0 <= x <= pi/4:                  # geändert
        return restrictedCos(x)
    if pi/4 <= x <= pi/2:               # neu
        return restrictedSin(pi/2 - x)  # neu
    if pi <= x <= 3*pi/2:
        return -myCos(x - pi)
    if 3*pi/2 <= x <= 2*pi:
        return myCos(2*pi - x)
    if pi/2 <= x <= pi:
        return -myCos(x + pi)
    if x <= 0:
```

```
        return myCos(x + 2*pi)
    return myCos(x - 2*pi)
```

Alternativ hätte man mit einem Additionstheorem auch auf den Zusammenhang

$$\begin{aligned}\cos(x + \pi/4) &= \cos x \cos \pi/4 - \sin x \sin \pi/4 \\ &= 1/\sqrt{2} \cdot (\cos x - \sin x)\end{aligned}$$

kommen können.

Lösung 417: Wir nennen die Hypotenuse c und die unbekannte Kathete b. Der Sinus von 42° ist das Verhältnis $4/c$. Der Kosinus des Winkels ist das Verhältnis b/c. Daher können wir die Werte so berechnen:

```
c = 4 / sin(42 * pi / 180)
b = c * cos(42 * pi / 180)
b, c
```

Wir mussten dabei natürlich darauf achten, den Winkel in Bogenmaß anzugeben.

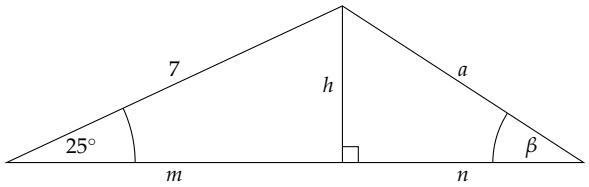
Lösung 419: Das Verhältnis der kurzen zur langen Kathete ist 3/7. Den kleinsten Winkel bekommt man also so:

```
from math import atan

alpha = atan(3/7) * 180 / pi
alpha
```

Der zweite Winkel ergibt sich als `90-alpha`. Die Länge der Hypotenuse könnte man nun z.B. so berechnen:

```
7 / cos(alpha * pi / 180), sqrt(3 * 3 + 7 * 7)
```

Lösung 420: Es bietet sich wohl diese Zerlegung an:

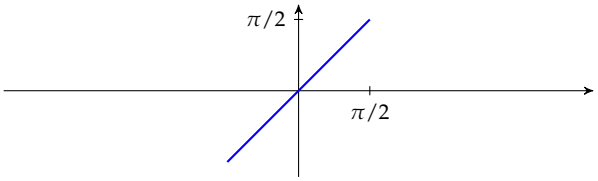


Die Länge der Strecke m ergibt sich sofort im linken rechtwinkligen Dreieck über die Hypotenuse und den Kosinus von 25°. Die Länge h erhält man mit Pythagoras. Der Winkel β ist die Differenz $180° - 25° - 122°$. Nun kennt man eine Seite und die Winkel im rechten rechtwinkligen Dreieck und kann die beiden anderen Seiten berechnen. Die gesuchten Seitenlängen des großen Dreieckes erhält man also so:

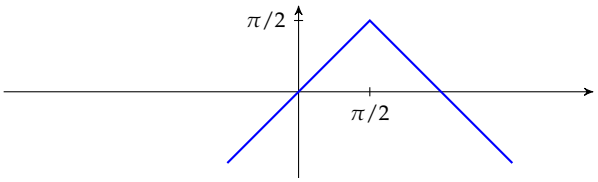
```
m = 7 * cos(25 * pi / 180)
h = sqrt(7 * 7 - m * m)
beta = 180 - 25 - 122
n = h / tan(beta * pi / 180)
a = n / cos(beta * pi / 180)
m + n, a
```

Natürlich gibt es noch viele andere Möglichkeiten, die Lösung zu ermitteln.

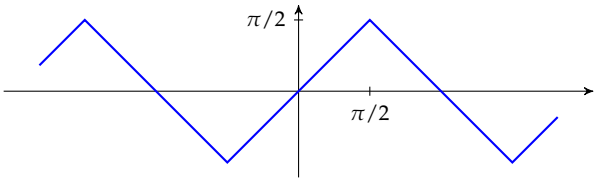
Lösung 421: Für Werte zwischen $-\pi/2$ und $\pi/2$ ist $\arcsin \circ \sin$ nach Definition des Arkussinus einfach die Identität. Daher sieht dieser Teil des Graphen so aus:



Setzt man nun Werte von $\pi/2$ bis $3\pi/2$ in den Sinus ein, so ist es so, als würde man den vorherigen Abschnitt rückwärts ablaufen.



Insgesamt ergibt sich eine Zickzacklinie.



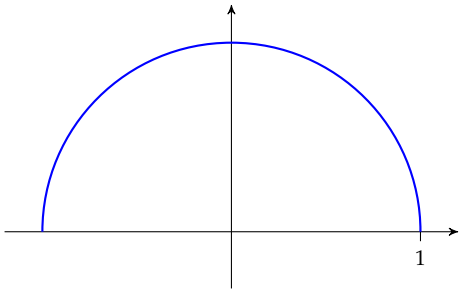
In PYTHON kann man das z.B. so erreichen (siehe Kapitel 41):

```
from plot import *
from math import asin, sin

plotFunc2D(lambda x: asin(sin(x)), [-10,10])
```

Lösung 422: Der Definitionsbereich ist der des Arkuskosinus, also das Intervall $[-1, 1]$. Den Graphen erhält man so:

```
from plot import *
from math import acos, sin

plotFunc2D(lambda x: sin(acos(x)), [-1,1])
```

Das sollte in etwa folgendermaßen aussehen:



Dass sich hier ein Halbkreis ergibt, kann man sich am einfachsten so erklären: Ein Punkt auf dem Graphen hat die Koordinaten $(x, y) = (x, \sin(\arccos x))$. Wenn man nun bedenkt, dass immer $\sin^2 x + \cos^2 x = 1$ gilt, so erhält man[45]

$$y^2 = \sin^2(\arccos x) = 1 - \cos^2(\arccos x) = 1 - x^2,$$

also $x^2 + y^2 = 1$. So ein Punkt (x, y) hat also den Abstand 1 vom Nullpunkt.

Lösung 424: Der Abstand ist

$$\sqrt{(3-(-1))^2 + (-2-1)^2} = \sqrt{4^2 + 3^2} = 5.$$

Sie haben hoffentlich keine Vorzeichenfehler gemacht. (Machen Sie sich anderenfalls eine Skizze. Die Punkte liegen auf unterschiedlichen Seiten der Koordinatenachsen!)

Lösung 425: Man kann es so machen:

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Oder z.B. auch so:

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Die Reihenfolge ist hier irrelevant, weil durch das Quadrieren die Vorzeichen keine Rolle spielen.

Lösung 426: Im Beispiel hatten wir zunächst

$$d_0 = \sqrt{(x_1 - x_2)^2 + (z_1 - z_2)^2}$$

berechnet und daraus dann den eigentlichen Abstand:

$$\sqrt{d_0^2 + (y_1 - y_2)^2} = \sqrt{\left(\sqrt{(x_1 - x_2)^2 + (z_1 - z_2)^2}\right)^2 + (y_1 - y_2)^2}$$

[45] Dabei gilt die letzte Gleichheit, weil der Arkuskosinus die Umkehrfunktion des Kosinus ist.

$$= \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

Bemerkenswert ist, dass wir am Ende eine Formel bekommen, die symmetrisch in allen drei Komponenten ist. Es macht keinen Unterschied, mit welcher Ebene wir anfangen.

Lösung 427: Der Punkt hat (in etwa) die Koordinaten $(3, 2)$:[46]

```
from math import cos, sin

r = 3.61
phi = 0.588
r * cos(phi), r * sin(phi)
```

Allgemein ist die Umrechnungsformel: $(x, y) = (r\cos\varphi, r\sin\varphi)$. Dafür muss man lediglich das entsprechende rechtwinklige Dreieck erkennen, das man immer finden kann, weil die Koordinatenachsen ja rechtwinklig aufeinander stehen.

Lösung 428: Man muss nur die letzte Zeile der vorherigen Lösung abschreiben:

```
from math import cos, sin

def cartesian (r, phi):
    return r * cos(phi), r * sin(phi)
```

Lösung 429: Das kann man mit dem Satz des Pythagoras ausrechnen: $r = \sqrt{x^2 + y^2}$. In diesem Fall ist r ungefähr 18.

```
from math import sqrt

sqrt(15*15 + 10*10)
```

Lösung 430: So könnte es aussehen:[47]

```
from math import sqrt, atan, pi

def polar (x, y):
    r = sqrt(x * x + y * y)
    if x == 0:
        if y > 0:
            phi = pi / 2
        elif y < 0:
```

[46] Aufgrund der ungenauen Angaben in der Aufgabenstellung kann die Antwort natürlich auch nicht exakt sein.

[47] `elif` ist quasi eine Abkürzung für `else: if`, mit der man sich eine Einrückung und eine zusätzliche Zeile erspart.

```
            phi = -pi / 2
        else:
            phi = 0
    else:
        phi = atan(y / x)
        if x < 0:
            phi += pi
    return r, phi, phi * 180 / pi
```

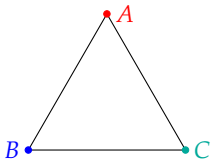
Die Funktion gibt den Winkel zweimal zurück, einmal in Bogenmaß und dann noch mal in Grad. Sie gibt im Quadranten rechts unten negative Winkel aus, ansonsten aber Winkel zwischen $0°$ und $270°$. Das ist zwar nicht falsch, aber ein kleiner Schönheitsfehler. Normalerweise würde man die Funktion noch so ergänzen, dass die zurückgegebenen Winkel entweder zwischen $0°$ und $360°$ oder zwischen $-180°$ und $180°$ liegen.

Lösung 431: So geht es sogar noch etwas kürzer:

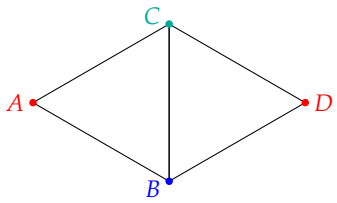
```
from math import sqrt, acos, pi

def polar (x, y):
    r = sqrt(x * x + y * y)
    if r == 0:
        phi = 0
    else:
        phi = acos(x / r)
        if y < 0:
            phi = -phi
    return r, phi, phi * 180 / pi
```

Diesmal liefert die Funktion Werte zwischen $-180°$ und $180°$.

Lösung 433: Zeichnen Sie ein gleichseitiges Dreieck mit der Seitenlänge 1:

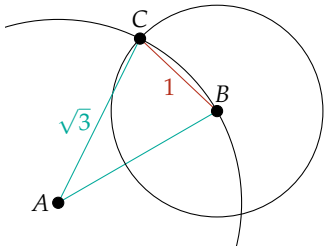


Punkt A bekommt irgendeine Farbe. Da der Abstand von B und A genau 1 ist, muss B eine andere Farbe bekommen. Der Abstand von C zu A und B ist aber auch jeweils 1, so dass C weder dieselbe Farbe wie A noch dieselbe Farbe wie B bekommen darf. Daher ist klar, dass *zwei* Farben nicht ausreichen. Nun gehen wir davon aus, dass wir drei Farben zur Verfügung haben. Zeichnen Sie zwei solche Dreiecke wie eben, die eine Seite gemeinsam haben:



Sie geben A eine bestimmte Farbe, wodurch sich automatisch ergibt, dass B und C die anderen beiden Farben bekommen müssen. Daraus folgt nun wiederum, dass D, dessen Abstand sowohl zu B als auch zu C ebenfalls 1 ist, eine Farbe haben muss, die weder B noch C haben. Das kann nur die Farbe von A sein.

Der Abstand von A und D ist $\sqrt{3}$.[48] Wir haben uns gerade überlegt, dass zwei Punkte deren Abstand $\sqrt{3}$ ist, dieselbe Farbe haben müssen.[49] Nun wählen Sie irgendeinen Punkt A aus und zeichen Sie einen Kreis mit dem Radius $\sqrt{3}$ um diesen Punkt. Auf dem Kreis wählen Sie einen Punkt B und zeichnen um diesen einen Kreis mit dem Radius 1. Einen der Schnittpunkte der beiden Kreise nennen Sie C:



Da B und C beide den Abstand $\sqrt{3}$ von A haben, müssen sie beide dieselbe Farbe wie A haben. Da ihr Abstand aber andererseits nach Konstruktion 1 ist, dürfen sie *nicht* dieselbe Farbe haben. Das zeigt, dass eine Einfärbung nach den Regeln mit lediglich drei Farben nicht möglich ist.

Wenn man die Fragestellung erweitert und sich fragt, was die minimale Anzahl an Farben ist, die man benötigt, um die Ebene nach diesen Regeln zu färben, dann wird daraus eine bisher ungelöste mathematische Frage, das sogenannte *Hadwiger-Nelson-Problem*. Bekannt ist nur, dass man mindestens fünf Farben braucht und dass sieben auf jeden Fall ausreichen. Die Antwort kann also fünf, sechs oder sieben sein. Dass die Antwort nicht vier sein kann, weiß man erst seit April 2018.

Lösung 434: Das sollte hoffentlich kein Problem sein:

$$\boldsymbol{a}+\boldsymbol{b}=\begin{pmatrix}-2\\5\end{pmatrix}+\begin{pmatrix}1\\-3\end{pmatrix}=\begin{pmatrix}-2+1\\5+(-3)\end{pmatrix}=\begin{pmatrix}-1\\2\end{pmatrix}$$

[48] Und zwar nach dem Satz von Pythagoras. Der Wert an sich spielt aber keine Rolle. Wichtig ist nur, dass der Abstand von zwei solchen Punkten an den gegenüberliegenden Ecken von zwei aneinanderliegenden gleichseitigen Dreiecken immer gleich ist.

[49] Denn zwischen zwei solchen Punkten können wir immer wie in der Zeichnung die beiden Dreiecke konstruieren.

$$3\boldsymbol{a} = 3 \cdot \begin{pmatrix} -2 \\ 5 \end{pmatrix} = \begin{pmatrix} 3 \cdot (-2) \\ 3 \cdot 5 \end{pmatrix} = \begin{pmatrix} -6 \\ 15 \end{pmatrix}$$
$$-\frac{1}{2}\boldsymbol{b} = -\frac{1}{2} \cdot \begin{pmatrix} 1 \\ -3 \end{pmatrix} = \begin{pmatrix} (-1/2) \cdot 1 \\ (-1/2) \cdot (-3) \end{pmatrix} = \begin{pmatrix} -1/2 \\ 3/2 \end{pmatrix}$$

Lösung 435: Zum Beispiel so:

```
def scalarVectorMult (x, A):
    return [x * a for a in A]

def vectorAdd (A, B):
    return [a + b for a, b in zip(A, B)]
```

Lösung 436: $\boldsymbol{a}$ und $\boldsymbol{b}$ sind parallel, denn es gilt $\boldsymbol{a} = -2\boldsymbol{b}$ bzw. $\boldsymbol{b} = -1/2 \cdot \boldsymbol{a}$. $\boldsymbol{a}$ und $\boldsymbol{c}$ sind nicht parallel. Würde es ein λ mit $\boldsymbol{c} = \lambda\boldsymbol{a}$ geben, dann müsste wegen der ersten Komponente $3 = \lambda \cdot 2$, also $\lambda = 3/2$ gelten. Das passt aber nicht zur zweiten Komponente, denn $3/2 \cdot 3$ ist nicht 2.

Lösung 437: Das Zeichen $+$ steht links für die Addition von zwei Vektoren, weiter rechts aber für die Addition von zwei reellen Zahlen. Ebenso steht das Multiplikationszeichen für zwei verschiedene Arten von Multiplikation: einmal werden zwei Objekte vom selben Typ (zwei reelle Zahlen) multipliziert, einmal zwei unterschiedliche Objekte, ein Skalar und ein Vektor.

Lösung 439: Solche Geraden haben die Steigung $m = 0$, wodurch der Term mx komplett wegfällt. Man erhält einen konstanten Ausdruck. Ein Beispiel wäre die Gerade $y = 42$.

Lösung 440: Geraden, die parallel zur y-Achse verlaufen, kann man so nicht darstellen. Sie müssten unendliche Steigung haben. Wir werden aber im folgenden Text Darstellungsmöglichkeiten kennenlernen, die auch für solche Geraden funktionieren.

Lösung 442: Mit $\lambda = 2$ erhält man $\begin{pmatrix} 2 \\ 0 \end{pmatrix} = \begin{pmatrix} -1 \\ -3/2 \end{pmatrix} + 2\begin{pmatrix} 3/2 \\ 3/4 \end{pmatrix}$, also liegt $(2, 0)$ auf g. Macht man hingegen den Ansatz

$$\begin{pmatrix} 3 \\ 1 \end{pmatrix} = \begin{pmatrix} -1 \\ -3/2 \end{pmatrix} + \lambda\begin{pmatrix} 3/2 \\ 3/4 \end{pmatrix},$$

so ergibt sich in der ersten Komponente $3 = -1 + \lambda \cdot 3/2$ bzw. $\lambda = 8/3$. Setzt man das in der zweiten Komponente ein, so erhält man $1 = 1/2$, was offensichtlich falsch ist. Daher ist $(3, 1)$ kein Punkt der Geraden.

Lösung 443: Aus Aufgabe 441 kennen wir andere Punkte der Geraden, z.B. $\begin{pmatrix} 1/2 \\ -3/4 \end{pmatrix}$ (für $\lambda = 1$). Einen anderen Richtungsvektor kann man z.B. durch $4/3 \cdot \begin{pmatrix} 3/2 \\ 3/4 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$ erhalten. Damit wäre

$$g = \left\{ \begin{pmatrix} 1/2 \\ -3/4 \end{pmatrix} + \lambda \begin{pmatrix} 2 \\ 1 \end{pmatrix} : \lambda \in \mathbb{R} \right\}$$

eine weitere Punkt-Richtungs-Form.

Lösung 444: Löst man $x = -1 + \lambda \cdot 3/2$ nach λ auf, so ergibt sich $\lambda = 2/3 \cdot (x+1)$. In der anderen Komponente erhält man durch Einsetzen dieses Wertes:

$$y = -\frac{3}{2} + \frac{2}{3} \cdot (x+1) \cdot \frac{3}{4} = \frac{x}{2} - 1$$

Das ist natürlich die ursprüngliche Geradengleichung (25.2).

Lösung 445: Wir haben in der ersten Komponente $x = 3 + \lambda \cdot 2$ bzw. $\lambda = 1/2 \cdot (x-3)$. Einsetzen in die zweite Komponente liefert

$$y = -2 + \frac{1}{2} \cdot (x-3) \cdot (-1) = -\frac{x}{2} - \frac{1}{2},$$

also $m = b = -1/2$.

Lösung 446: Für die Gerade $y = 12$ kann man als einen Punkt auf der Geraden etwa $(0, 12)$ wählen. Für die Richtung kann man den Vektor $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ wählen. Das ergibt:

$$\begin{pmatrix} 0 \\ 12 \end{pmatrix} + \mathbb{R} \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

Für $y = -2x + 5$ könnte man in diese Gleichung z.B. $x = 0$ und $x = 1$ einsetzen. Das liefert die beiden Punkte $(0, 5)$ und $(1, 3)$. Damit erhält man:

$$\left\{ \begin{pmatrix} 0 \\ 5 \end{pmatrix} + \lambda \left(\begin{pmatrix} 0 \\ 5 \end{pmatrix} - \begin{pmatrix} 1 \\ 3 \end{pmatrix} \right) : \lambda \in \mathbb{R} \right\} = \begin{pmatrix} 0 \\ 5 \end{pmatrix} + \mathbb{R} \begin{pmatrix} -1 \\ 2 \end{pmatrix}$$

Lösung 447: Eine Möglichkeit sieht so aus:

$$\left\{ \begin{pmatrix} 1 \\ 2 \end{pmatrix} + \lambda \left(\begin{pmatrix} -3 \\ 1 \end{pmatrix} - \begin{pmatrix} 1 \\ 2 \end{pmatrix} \right) : \lambda \in \mathbb{R} \right\} = \begin{pmatrix} 1 \\ 2 \end{pmatrix} + \mathbb{R} \begin{pmatrix} -4 \\ -1 \end{pmatrix} \tag{A.6}$$

Man könnte es aber auch so schreiben:

$$\left\{ \lambda \begin{pmatrix} 1 \\ 2 \end{pmatrix} + \mu \begin{pmatrix} -3 \\ 1 \end{pmatrix} : \lambda, \mu \in \mathbb{R} \text{ und } \lambda + \mu = 1 \right\}$$

Lösung 448: Setzt man den Punkt in (A.6) ein, so kann man wie in den vorherigen Aufgaben nach λ auflösen:

$$\begin{pmatrix} 5 \\ 3 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix} + (-1) \cdot \begin{pmatrix} -4 \\ -1 \end{pmatrix} \tag{A.7}$$

Daher liegt $(5,3)$ auf g. Ersetzt man $\begin{pmatrix}-4\\-1\end{pmatrix}$ jedoch durch die Differenz $\begin{pmatrix}-3\\1\end{pmatrix} - \begin{pmatrix}1\\2\end{pmatrix}$, so liefert (A.7):

$$\begin{pmatrix}5\\3\end{pmatrix} = 2 \cdot \begin{pmatrix}1\\2\end{pmatrix} - 1 \cdot \begin{pmatrix}-3\\1\end{pmatrix}$$

Da weder 2 noch -1 aus dem Intervall $[0,1]$ sind, liegt der Punkt nicht auf der Verbindungsstrecke von P_1 und P_2.

Lösung 449: Man kann das lineare Gleichungssystem z.B. dadurch lösen, dass man die erste Gleichung nach μ auflöst ($\mu = -1 - 2\lambda$) und diesen Wert in die zweite einsetzt:

$$7\lambda - 5(-1 - 2\lambda) = 17\lambda + 5 = 3$$

Daraus folgt $\lambda = -2/17$ und Einsetzen in die zugehörige Geradengleichung liefert den folgenden Schnittpunkt:

$$\boldsymbol{p} = \begin{pmatrix}3\\-1\end{pmatrix} - \frac{2}{17} \cdot \begin{pmatrix}2\\7\end{pmatrix} = \frac{1}{17} \cdot \begin{pmatrix}47\\-31\end{pmatrix}$$

Sie können Ihre Lösung überprüfen, indem Sie μ ermitteln ($\mu = -1 - 2 \cdot (-2/17)$) und in die andere Geradengleichung einsetzen. Es muss natürlich derselbe Punkt herauskommen.

Lösung 450: Die beiden Geraden können parallel sein. In diesem Fall schneiden sie sich gar nicht, d.h. ihre Schnittmenge ist leer. Oder im Extremfall sind die beiden Geraden nicht nur parallel, sondern sogar identisch, man hat es also eigentlich nur mit einer Geraden zu tun. In diesem Fall besteht die Schnittmenge natürlich aus der ganzen Geraden. Im „normalen“ Fall schneiden die beiden Geraden sich aber in genau einem Punkt wie in der vorherigen Aufgabe.

Machen Sie sich klar, dass es *nicht* möglich ist, dass zwei Geraden mehr als einen Punkt, aber nicht alle gemeinsam haben: wenn zwei Geraden zwei verschiedene Punkte gemeinsam haben, müssen sie bereits identisch sein, da eine Gerade durch zwei Punkte eindeutig festgelegt ist.

Im Raum können auch nur diese drei Fälle (Schnittmenge ist leer, besteht aus einem Punkt oder ist die ganze Gerade) eintreten. Allerdings kann die Schnittmenge auch leer sein, wenn die Geraden nicht parallel sind, was in der Ebene nicht möglich ist. Man spricht dann von *windschiefen* Geraden.

Lösung 451: Zum Beispiel so:

$$\begin{pmatrix}1\\2\\0\end{pmatrix} + \mathbb{R} \begin{pmatrix}-4\\-1\\5\end{pmatrix}$$

Lösung 452: Sie dürfen nicht parallel sein, denn sonst zeigen beide in dieselbe Richtung und man erhält die Darstellung einer Geraden.

Lösung 453: Wie in der Grafik auf Seite 289 zu sehen ist, ist es sinnvoll, beide Richtungsvektoren im selben Punkt „starten“ zu lassen. Man kann also z.B. $\boldsymbol{p}_1$ als Ortsvektor und $\boldsymbol{p}_2 - \boldsymbol{p}_1$ sowie $\boldsymbol{p}_3 - \boldsymbol{p}_1$ als Richtungsvektoren wählen. Die resultierende Darstellung wäre dann diese:

$$\begin{pmatrix}1\\2\\1\end{pmatrix} + \mathbb{R}\begin{pmatrix}2\\-3\\4\end{pmatrix} + \mathbb{R}\begin{pmatrix}3\\-2\\-7\end{pmatrix}$$

Lösung 454: Ein Punkt $\boldsymbol{p}$, der sowohl auf der Geraden als auch auf der Ebene liegt, lässt sich auf zwei Arten darstellen:

$$\boldsymbol{p} = \begin{pmatrix}1\\2\\0\end{pmatrix} + \lambda\begin{pmatrix}-4\\-1\\5\end{pmatrix} = \begin{pmatrix}1\\2\\1\end{pmatrix} + \mu\begin{pmatrix}2\\-3\\4\end{pmatrix} + \nu\begin{pmatrix}3\\-2\\-7\end{pmatrix}$$

Wenn man das komponentenweise aufschreibt, erhält man ein lineares Gleichungssystem mit drei Gleichungen für die drei Unbekannten λ, μ und ν.

$$\begin{aligned}4\lambda + 2\mu + 3\nu &= 0\\ -\lambda + 3\mu + 2\nu &= 0\\ 5\lambda - 4\mu + 7\nu &= 1\end{aligned}$$

Falls Sie sich die Mühe gemacht haben, das auszurechnen: man erhält den Wert $-5/117$ für λ. Als Schnittpunkt ergibt sich dann $\frac{1}{117} \cdot \begin{pmatrix}137\\239\\-25\end{pmatrix}$.

Lösung 455: Die Gerade trifft die Ebene entweder in genau einem Punkt oder sie verläuft parallel zu ihr und trifft sie gar nicht oder sie liegt ganz in der Ebene drin.

Zwei Ebenen können identisch sein oder verschieden und parallel (also einen leeren Schnitt haben). Oder sie schneiden sich in einer Geraden. Es ist nicht möglich, dass zwei Ebenen im Raum nur genau einen Punkt gemeinsam haben.[50]

Lösung 456: Ich habe die Funktionen aus Aufgabe 435 wiederverwendet:[51]

```
def matrixAdd (A, B):
    return [vectorAdd(a, b) for a, b in zip(A, B)]

def scalarMatrixMult (x, A):
    return [scalarVectorMult(x, a) for a in A]
```

[50] Anschaulich ist das wohl klar. Formal kann man das damit begründen, dass die Berechnung des Durchschnitts auf ein lineares Gleichungssystem mit drei Gleichungen für vier Unbekannte führt. So ein System nennt man *unterbestimmt*, weil es sozusagen „zu wenige“ Gleichungen hat. Es hat entweder gar keine Lösung oder unendlich viele, aber nicht genau eine.

[51] Das Wiederverwenden von Funktionen, die man bereits hat und von denen man weiß, dass sie funktionieren, ist grundsätzlich eine gute Strategie.

Lösung 458: Das Produkt sieht so aus:

$$\begin{pmatrix} 3 & 7 & 0 \\ -1 & -2 & 9 \\ -1 & 4 & 3 \\ 8 & 0 & -6 \end{pmatrix} \cdot \begin{pmatrix} 5 & 2 & -4 \\ -1 & 3 & 7 \\ 0 & -2 & 2 \end{pmatrix} = \begin{pmatrix} 8 & 27 & 37 \\ -3 & -26 & 8 \\ -9 & 4 & 38 \\ 40 & 28 & -44 \end{pmatrix}$$

Lösung 459: $\boldsymbol{A} \cdot \boldsymbol{C}$ und $\boldsymbol{B} \cdot \boldsymbol{B}$ sind 3×3-Matrizen, $\boldsymbol{B} \cdot \boldsymbol{A}$ ist eine 3×5-Matrix, $\boldsymbol{C} \cdot \boldsymbol{A}$ eine 5×5-Matrix und $\boldsymbol{C} \cdot \boldsymbol{B}$ eine 5×3-Matrix. Die anderen Produkte kann man alle nicht bilden.

Lösung 460: Die Produkte sind:

$$\boldsymbol{A} \cdot \boldsymbol{B} = \begin{pmatrix} 2 & 4 \\ 1 & 2 \end{pmatrix} \qquad \boldsymbol{B} \cdot \boldsymbol{A} = \begin{pmatrix} 4 & 2 \\ 0 & 0 \end{pmatrix}$$

Lösung 461: Nein. Sie haben hoffentlich sofort bemerkt, dass Aufgabe 460 diese Frage schon beantwortet.

Lösung 462: Meine Lösung sieht so aus:

```
def scalarProd (A, B):
    return sum(a * b for a, b in zip(A, B))

def col (A, i):
    return [row[i] for row in A]

def matrixProd (A, B):
    return [[scalarProd(row, col(B, i))
             for i in range(len(B[0]))] for row in A]
```

Warum die eine Hilfsfunktion `scalarProd` heißt, wird später noch klar werden.

Lösung 464: Die Antwort folgt im Text.

Lösung 466: Es ergibt sich das folgende Produkt:

$$\begin{pmatrix} 4 & 5 & 0 \\ -2 & -2 & 8 \\ -1 & 4 & 3 \\ 7 & 0 & -6 \end{pmatrix} \cdot \begin{pmatrix} 2 \\ 3 \\ -2 \end{pmatrix} = \begin{pmatrix} 23 \\ -26 \\ 4 \\ 26 \end{pmatrix}$$

Lösung 467: Zum Beispiel so:

```
def matTimesVec (A, V):
    return [p[0] for p in matrixProd(A, [[v] for v in V])]
```

Lösung 468: Benutzt man `col` aus Aufgabe 462, dann wird das recht kurz:

```
def transpose (A):
    return [col(A, i) for i in range(len(A[0]))]
```

Lösung 470: Man bekommt wieder einen transponierten Vektor. Wegen (26.2) ist das Ergebnis auch der transponierte Vektor der „normalen“ Multiplikation:

$$\boldsymbol{v}^T \cdot \boldsymbol{A} = (\boldsymbol{A}^T \cdot \boldsymbol{v})^T$$

Lösung 471: Die erste Gleichung hat die zwei Löungen -5 und 5. In der zweiten kann man für y die Werte -4 und 4 einsetzen. Für (x, y) gibt es also *vier* Lösungen:

$$(5,4) \qquad (5,-4) \qquad (-5,4) \qquad (-5,-4)$$

Lösung 472: Die Matrix sieht so aus:

$$\left(\begin{array}{ccc|c} 6 & -1 & -1 & 4 \\ 1 & 1 & 10 & -6 \\ 2 & -1 & 1 & -2 \end{array}\right)$$

Lösung 473: Die dritte Gleichung liefert $z = -3$. Einsetzen in die zweite Gleichung ergibt $2y - 3 = 7$ bzw. $y = 5$. Mit diesen beiden Werten ergibt sich in der ersten Gleichung $-x + 15 - 3 = 13$, also $x = -1$. Die einzige Lösung ist somit $(-1, 5, -3)$, die Lösungs*menge* also $\{(-1, 5, -3)\}$.

Lösung 474: Das könnte so aussehen:

$$\begin{pmatrix} 0 & \textcircled{1} & 1 \\ 0 & \textcircled{1} & 1 \\ 0 & 0 & \textcircled{1} \end{pmatrix}$$

Dies ist wegen der blauen Nullen eine obere Dreiecksmatrix, aber der Leitkoeffizient der zweiten Zeile steht nicht weiter rechts als der der ersten Zeile, also hat die Matrix nicht Zeilenstufenform.

Lösung 475: Zunächst sei darauf hingewiesen, dass es beim Gaußverfahren viele Wege gibt, die nach Rom führen. Wenn Sie die Aufgabe anders gelöst haben als ich, ist das kein Problem. Auch die Matrix in Zeilenstufenform am Ende kann eine andere sein. Allerdings muss die Lösungsmenge dieselbe sein, die auch ich hier erhalte, sonst hat einer von uns beiden einen Fehler gemacht!

Wir beginnen mit dieser Matrix.

$$\left(\begin{array}{ccc|c} 6 & -1 & -1 & 4 \\ 1 & 1 & 10 & -6 \\ 2 & -1 & 1 & -2 \end{array}\right)$$

Wir haben eigentlich schon ein Pivotelement an der richtigen Stelle. Wenn man ohne elektronische Hilfsmittel rechnet, ist es aber oft hilfreich, ein Pivotelement zu wählen, mit dem sich leicht rechnen lässt. Das geht besonders gut mit den Zahlen 1 und -1

oder ersatzweise mit ganzen Zahlen, die nicht weit von der Null entfernt sind. Ich vertausche daher zunächst die ersten beiden Zeilen.

$$\left(\begin{array}{ccc|c} \textcircled{1} & 1 & 10 & -6 \\ 6 & -1 & -1 & 4 \\ 2 & -1 & 1 & -2 \end{array}\right)$$

Nun subtrahiere ich das Sechsfache der ersten von der zweiten Zeile.

$$\left(\begin{array}{ccc|c} \textcircled{1} & 1 & 10 & -6 \\ 0 & -7 & -61 & 40 \\ 2 & -1 & 1 & -2 \end{array}\right)$$

Anschließend wird das Doppelte der ersten von der dritten Zeile abgezogen.[52]

$$\left(\begin{array}{ccc|c} 1 & 1 & 10 & -6 \\ 0 & -7 & -61 & 40 \\ 0 & -3 & -19 & 10 \end{array}\right)$$

Nun hat man keine „schöne“ Zahl mehr, die man als Pivotelement wählen kann. Nimmt man z.B. -7, so muss man die zweite Zeile mit $3/7$ multiplizieren und von der dritten subtrahieren, um eine Null zu erzeugen. Dabei kommen unangenehme Brüche ins Spiel.

Man kann das aber vermeiden (auf Kosten größerer Zahlen), indem man vorher „prophylaktisch“ die dritte Zeile mit 7 multipliziert.

$$\left(\begin{array}{ccc|c} 1 & 1 & 10 & -6 \\ 0 & \textcircled{-7} & -61 & 40 \\ 0 & -21 & -133 & 70 \end{array}\right)$$

Jetzt kann man das Dreifache der zweiten von der dritten Zeile subtrahieren, ohne dass irgendwo Brüche auftauchen.

$$\left(\begin{array}{ccc|c} 1 & 1 & 10 & -6 \\ 0 & -7 & -61 & 40 \\ 0 & 0 & 50 & -50 \end{array}\right)$$

Wir haben eine Zeilenstufenform erreicht und können durch „Rückwärtseinsetzen“ wie in Aufgabe 473 die Lösung finden. Die Lösungsmenge ist $\{(1,3,-1)\}$.

Lösung 476: Im ersten Schritt ist die Eins links oben unser Pivotelement und wir erledigen die zweite und die dritte Zeile in einem Schritt.

$$\left(\begin{array}{ccc|c} \textcircled{1} & 0 & 1 & 5 \\ 0 & 1 & 1 & -2 \\ 0 & 1 & 1 & -3 \end{array}\right)$$

[52] Die beiden letzten Schritte hätte man auch in einem zusammenfassen können. Das bietet sich insbesondere dann an, wenn solche Berechnung von Hand durchgeführt werden und Schreibarbeit gesparen werden soll. Man muss dabei aber aufpassen, dass man die elementaren Zeilenumformungen immer noch *nacheinander* ausführt und lediglich beim Aufschreiben Schritte spart, sonst macht man evtl. Fehler.

Nun haben wir netterweise wieder eine Eins als Pivotelement und können die Null unter ihr auch ganz einfach erzeugen.

$$\left(\begin{array}{ccc|c} 1 & 0 & 1 & 5 \\ 0 & \textcircled{1} & 1 & -2 \\ 0 & 0 & 0 & -1 \end{array}\right)$$

Lösung 477: Um uns die Arbeit zu erleichtern, vertauschen wir zunächst die erste mit der dritten Zeile. Dann werden in den Zeilen darunter Nullen erzeugt.

$$\left(\begin{array}{ccc|c} \textcircled{-1} & -1 & 9 & -12 \\ 0 & -1 & 16 & -18 \\ 0 & -2 & 32 & -36 \end{array}\right)$$

Wir fahren fort mit der zweiten Zeile und sind in einem Schritt schon fertig:

$$\left(\begin{array}{ccc|c} -1 & -1 & 9 & -12 \\ 0 & \textcircled{-1} & 16 & -18 \\ 0 & 0 & 0 & 0 \end{array}\right)$$

Lösung 478: Wenn man die Lösungstupel als Vektoren schreibt, dann sieht die Lösungsmenge so aus:

$$\begin{aligned} \left\{ \begin{pmatrix} -6-7z \\ 18+16z \\ z \end{pmatrix} : z \in \mathbb{R} \right\} &= \left\{ \begin{pmatrix} -6 \\ 18 \\ 0 \end{pmatrix} + \begin{pmatrix} -7z \\ 16z \\ z \end{pmatrix} : z \in \mathbb{R} \right\} \\ &= \left\{ \begin{pmatrix} -6 \\ 18 \\ 0 \end{pmatrix} + z \cdot \begin{pmatrix} -7 \\ 16 \\ 1 \end{pmatrix} : z \in \mathbb{R} \right\} \\ &= \begin{pmatrix} -6 \\ 18 \\ 0 \end{pmatrix} + \mathbb{R} \begin{pmatrix} -7 \\ 16 \\ 1 \end{pmatrix} \end{aligned}$$

Das ist die Punkt-Richtungs-Form einer Geraden im Raum $\mathbb{R}^3$. Wie wir noch sehen werden, ist das kein Zufall.

Lösung 479: Ja, das geht. In unserem ersten Beispiel hatten wir es etwa mit dieser Matrix zu tun, in deren erster Zeile der Leitkoeffizient zu weit rechts stand:

$$\begin{pmatrix} 0 & 0 & -1 & 5 \\ 2 & -2 & 1 & 3 \\ 4 & -4 & 3 & 8 \end{pmatrix}$$

Statt die erste Zeile mit einer anderen zu vertauschen, können wir aber z.B. einfach die zweite zur ersten addieren:

$$\begin{pmatrix} \textcircled{2} & -2 & 0 & 8 \\ 2 & -2 & 1 & 3 \\ 4 & -4 & 3 & 8 \end{pmatrix}$$

Nun ist der Leitkoeffizient in der ersten Zeile an der richtigen Stelle und wir können wie üblich fortfahren. Man kann also beim Gaußverfahren ausschließlich mit elementaren Zeilenumformungen vom Typ (III) auskommen.

Lösung 480: Mit dieser Eingabe

```
M = [[6, -1, -1, 4], [1, 1, 10, -6], [2, -1, 1, -2]]
rowEchelon(M)
M
```

erhält man diese Antwort:

```
[[6, -1, -1, 4],
 [0.0, 1.1666666666666667, 10.166666666666666,
      -6.666666666666667],
 [0.0, 1.1102230246251565e-16, 7.142857142857143,
      -7.142857142857143]]
```

Das Problem dabei ist die rot hervorgehobene Zahl. Diese Zahl ist *fast* null, aber eben nicht ganz. Die Matrix ist also *nicht* in Zeilenstufenform!

Was genau ist passiert? Die Funktion benutzt zuerst die Zahl 6 links oben als Pivotelement und zieht ein Sechstel der ersten von der zweiten Zeile ab. Außerdem zieht sie ein Drittel der ersten von der dritten Zeile ab.

In der zweiten Spalte sollte in der zweiten Zeile nun $a_{2,2} = 1 + 1/6 = 7/6$ stehen und in der Zeile darunter $a_{3,2} = -1 + 1/3 = -2/3$. Da $a_{2,2}$ jetzt das Pivotelement ist, muss das Programm die zweite Zeile mit $a_{3,2}/a_{2,2} = -4/7$ multiplizieren und von der dritten subtrahieren.

Was PYTHON wirklich berechnet, sehen Sie so:

```
a22 = 1 + 1/6
a32 = -1 + 1/3
factor = a32 / a22
a22, a32, factor, factor * a22, a32 - factor * a22
```

Dass solche Fehler auftreten können, wissen wir aus Kapitel 13. Hier sehen wir aber zum ersten Mal anhand eines praktischen Beispiels, dass dadurch ein Ergebnis nicht nur „knapp daneben“ liegt, sondern komplett falsch ist.

Lösung 481: Wenn es um exakte Ergebnisse geht, kann man mit Brüchen, also mit dem Modul FRACTIONS rechnen. (Probieren Sie's aus! Den Code von `rowEchelon` müssen Sie dafür nicht ändern.) Wenn man bei Fließkommazahlen bleiben will, wäre eine Alternative, Zahlen, die nahe genug bei Null sind, quasi „abzuschneiden“. Dafür könnte man z.B. `rowMod` modifizieren:

```
from math import isclose

def checkZero (x):
    return 0 if isclose(x, 0, abs_tol = 1e-14) else x

def rowMod (M, i, j, x):
    M[i] = [checkZero(a + x * b)
            for a, b in zip(M[i], M[j])]
```

Das ist aber auch etwas unbefriedigend. Wir werden auf dieses Thema noch einmal zurückkommen.

Lösung 484: Man könnte es so machen:

```
def identityMatrix (n):
    E = [[0] * n for i in range(n)]
    for i in range(n):
        E[i][i] = 1
    return E

def type3Matrix (n, i, j, x):
    A = identityMatrix(n)
    A[i][j] = x
    return A

def rowMod (M, i, j, x):
    return matrixProd(type3Matrix(len(M), i, j, x), M)
```

`rowEchelon` müsste dann aber auch leicht verändert werden, weil `rowMod` nun nicht mehr sein erstes Argument direkt modifiziert, sondern stattdessen die neue Matrix zurückgibt.

Lösung 485: Wir zeigen die Lösungen mithilfe von Beispielen, wobei $\boldsymbol{A}$ jeweils eine $4 \times n$-Matrix sein soll. Für den Typ (I) multiplizieren wir die dritte Zeile von $\boldsymbol{A}$ mit 7:

$$3.\text{ Zeile} \to \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 7 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \boldsymbol{A}$$

Und für Typ (II) vertauschen wir die erste Zeile mit der dritten:

$$\begin{array}{r} \text{1. Zeile} \rightarrow \\ \\ \text{3. Zeile} \rightarrow \\ \\ \end{array} \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \boldsymbol{A}$$

↑ 1. Spalte ↑ 3. Spalte

Das demonstriert hoffentlich, wie man es im allgemeinen Fall machen würde.

Lösung 486: Wir haben in `pointOnCircle` auf Gleichheit getestet. Aus Kapitel 13 wissen wir aber, dass das bei Fließkomma-Arithmetik nicht sinnvoll ist. Besser wäre so etwas:

```
def pointOnCircle (p):
    dist = abs(sqrt((p[0]-2) ** 2 + (p[1]-1) ** 2) - 3)
    return dist < 0.01
```

Lösung 488: Der Parameter in der Darstellung (28.1) ist natürlich der Winkel α.

Lösung 489: Wegen $\lambda + \mu = 1$ folgt sofort $\mu = 1 - \lambda$, also kann man die Strecke auch so darstellen:

$$\{ \lambda \boldsymbol{p}_1 + (1-\lambda)\boldsymbol{p}_2 : \lambda \in [0,1] \} = \{ \boldsymbol{p}_2 + \lambda(\boldsymbol{p}_1 - \boldsymbol{p}_2) : \lambda \in [0,1] \}$$

Lösung 490: Das könnte man z.B. so machen:

```
def pointOnSegment (L, p1, p2):
    return (p2[0] + L * (p1[0] - p2[0]),
            p2[1] + L * (p1[1] - p2[1]))

def draw (c):
    c.clear()
    c.drawAxes()
    L = 0
    while L <= 1:
        c.drawPoint(pointOnSegment(L, (-2,1), (4,-2)))
        L += .05
```

Lösung 491: `pointOnSegment` sieht dann etwas eleganter aus:

```
from vectors import Vector

def pointOnSegment (L, p1, p2):
    return p2 + L * (p1 - p2)
```

```
def draw (c):
    c.clear()
    c.drawAxes()
    L = 0
    p1 = Vector(-2, 1)
    p2 = Vector(4, -2)
    while L <= 1:
        c.drawPoint(pointOnSegment(L, p1, p2))
        L += .05
```

Lösung 492: Zum Beispiel so:

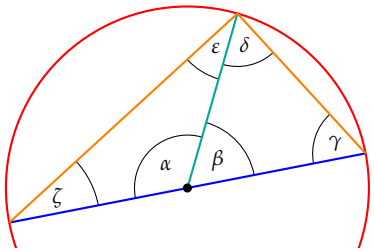
```
def animate (c):
    L = 0
    inc = 0.02
    P1 = Vector(-2, 1)
    P2 = Vector(4, -2)
    while True:
        c.clear()
        c.drawPoint(pointOnSegment(L, P1, P2))
        if L < 0 or L > 1:
            inc = -inc
        L = L + inc
        sleep(0.02)
```

Diese Funktion läuft „ewig“. Sie müssen sie mit dem „Stop“-Button von JUPYTER abbrechen.

Lösung 494: Ich habe es so gelöst:

```
def circlePoint (alpha, M, r):
    return M + r * Vector(cos(alpha), sin(alpha))

def Thales (c):
    c.clear()
    c.drawGrid()      # optional
    M = Vector(-3,2)
    r = 4
    c.drawCircle(M, r, color = "red")
    P1 = circlePoint(.2, M, r)
    P2 = circlePoint(.2 + pi, M, r)
    c.drawLine(P1, P2, color = "blue")
    P3 = circlePoint(1.3, M, r)
    c.drawSegment(P3, P1, color = "orange")
    c.drawSegment(P3, P2, color = "orange")
```

Dass wir es mit einem rechten Winkel zu tun haben, besagt der aus der Schule bekannte *Satz des Thales*.[53] Die Korrektheit des Satzes kann man sich leicht selbst klarmachen. Dazu zieht man zunächst vom Mittelpunkt des Kreises eine gerade Linie zur oberen Ecke des Dreiecks, durch die man dieses in zwei kleine Dreiecke zerlegt.



Da die drei vom Mittelpunkt ausgehenden Strecken alle Radien des Kreises sind, sind sie alle gleich lang. Die beiden kleinen Dreiecke sind also beide gleichschenklig und daher gilt $\varepsilon = \zeta$ und $\gamma = \delta$. Im linken Dreieck haben wir $\alpha + \varepsilon + \zeta = \alpha + 2\varepsilon = 180°$ und rechts entsprechend $\beta + 2\delta = 180°$. Addiert man beide Identitäten, so erhält man $\alpha + \beta + 2\varepsilon + 2\delta = 360°$. Gleichzeitig sind α und β aber Ergänzungswinkel, deren Summe $180°$ beträgt. Daraus folgt dann $2\varepsilon + 2\delta = 180°$. Teilt man durch zwei, so ergibt sich, dass $\varepsilon + \delta$ ein rechter Winkel ist.

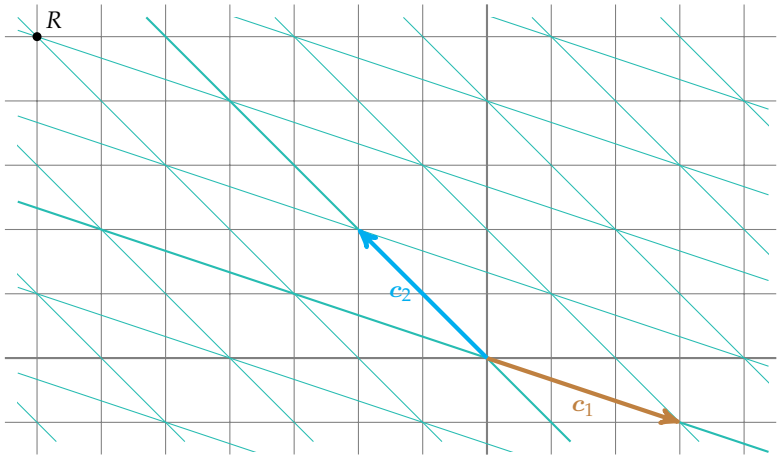
Lösung 495: Die Abbildung zur Berechnung der alten Koordinaten eines Punktes ist diese:

$$f : \begin{cases} \mathbb{R}^2 \to \mathbb{R}^2 \\ \boldsymbol{x} \mapsto \begin{pmatrix} 3 & -2 \\ -1 & 2 \end{pmatrix} \cdot \boldsymbol{x} \end{cases}$$

Die Spalten der Matrix sind die Vektoren $\boldsymbol{c}_1$ und $\boldsymbol{c}_2$. Setzt man die neuen Koordinaten von R ein, so ergibt sich $f((-1,2)^T) = (-7,5)^T$. In alten Koordinaten also $R = (-7,5)_a$.

Für die Visualisierung habe ich diesmal eine etwas andere Darstellung gewählt. Hier ist das alte Koordinatensystem ein graues Gitter, das neue ist grün.

[53] Das ist der älteste mathematische Satz, der nach einer Person benannt ist; er ist also sogar noch etwas älter als der Satz des Pythagoras. Und über den Menschen Thales, der vor über 2600 Jahren geboren wurde, weiß man genauso wenig wie über Pythagoras. Insbesondere hat er sicher nicht so ausgesehen, wie ihn die obige Illustration darstellt, die wiederum einem Bild aus dem 19. Jahrhundert nachempfunden wurde.



Lösung 496: Im Prinzip ändert sich gar nichts. Die Abbildung zum Umrechnen von neuen Koordinaten in alte würde nun so aussehen:

$$: \begin{cases} \mathbb{R}^3 \to \mathbb{R}^3 \\ \boldsymbol{x} \mapsto \begin{pmatrix} 2 & 1 & 0 \\ 1 & 2 & 3 \\ -2 & 0 & -1 \end{pmatrix} \cdot \boldsymbol{x} \end{cases}$$

Setzt man die neuen Koordinaten von S ein, dann ergibt sich in alten Koordinaten $S = (3, 15, -15)_a$.

Lösung 497: Wenn man in Gleichung (29.3) $\boldsymbol{b}_1$ und $\boldsymbol{b}_2$ in alten Koordinaten hinschreibt, dann erhält man ein lineares Gleichungssystem für x und y:

$$x \cdot \boldsymbol{b}_1 + y \cdot \boldsymbol{b}_2 = x \cdot \begin{pmatrix} 3 \\ 1 \end{pmatrix}_a + y \cdot \begin{pmatrix} -1 \\ 2 \end{pmatrix}_a = \begin{pmatrix} 5 \\ 2 \end{pmatrix}_a$$

Mit den Methoden aus Kapitel 27 erhält man sofort die Lösung $x = 12/7$ und $y = 1/7$. Und dass dieses Ergebnis korrekt ist, sehen Sie z.B. so:

```
def draw (c):
    c.clear()
    b1 = Vector(3, 1)
    b2 = Vector(-1, 2)
    A = Matrix([b1, b2])
    c.drawAxes()
    c.drawAxes(b1, b2,
               colors = ["lightgray", "orange", "violet"])
    # alte Koordinaten
    c.drawPoint((5,2), color = "blue", radius = 3)
    # neue Koordinaten
    c.drawPoint(A * (1/7 * Vector(12,1)), color = "white")
```

Lösung 499: Es wird nun oben nach links und unten nach rechts geschert.

Lösung 500: Zum Beispiel so:

$$\begin{pmatrix} 1 & 0 \\ 1.3 & 1 \end{pmatrix}$$

Allgemein erreicht man achsenparallele Scherungen, indem man mit einer Einheitsmatrix beginnt und eine der Nullen durch eine andere Zahl ersetzt.

Lösung 501: Die erste Matrix beschreibt eine Drehung um $\alpha = 90°$. Die zweite ist die Spiegelungsmatrix für $\alpha = 45°$, d.h. es wird an der Hauptdiagonalen gespiegelt.

Lösung 502: In beiden Fällen kann man sich sofort, ohne Verwendung von Sinus und Kosinus, überlegen, auf welche Vektoren die beiden kanonischen Einheitsvektoren abgebildet werden müssen. Das müssen dann ja die Spalten der Matrix sein. Das Ergebnis sieht so aus:

$$\begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix} \qquad \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

Lösung 503: Man kann das so erreichen:

$$\begin{pmatrix} 1 & 2 \\ 2 & 4 \end{pmatrix}$$

Die Spalten sind nun Vielfache voneinander, also parallel. Die gesamte Ebene wird also auf die Gerade abgebildet, die durch den Nullpunkt geht und den Richtungsvektor $(1, 2)^T$ hat.

Lösung 504: Der Vektor $\mathbf{0}$ lässt sich als $0 \cdot \mathbf{0}$ darstellen. Aus der zweiten Bedingung folgt: $f(\mathbf{0}) = f(0 \cdot \mathbf{0}) = 0 \cdot f(\mathbf{0}) = \mathbf{0}$. (Das liegt natürlich daran, dass $0 \cdot \boldsymbol{v} = \mathbf{0}$ für jeden Vektor $\boldsymbol{v}$ gilt.)

Lösung 505: Das könnte z.B. so aussehen:

$$: \begin{cases} \mathbb{R}^2 \to \mathbb{R}^2 \\ \begin{pmatrix} x \\ y \end{pmatrix} \mapsto \begin{pmatrix} x^2 \\ y \end{pmatrix} \end{cases}$$

Eine leicht modifizierte Version dieses Beispiels, die ein etwas hübscheres Bild ergibt, liefert der folgende Python-Code. Man sieht sehr deutlich, dass die „neuen“ Achsen parallel zu den „alten“ sind, aber die Hauptdiagonale nicht auf eine Gerade abgebildet wird. Das liegt natürlich daran, dass die Streckenverhältnisse nicht erhalten bleiben.

```
from math import copysign

def f (v):
    return Vector(copysign(0.5, v[0]) * v[0] * v[0], v[1])
```

```
def diagPoints ():
    return [Vector(k/5 - 4, k/5 - 4) for k in range(40)]

def draw (c):
    c.drawGrid()
    for x in range(-10, 10):
        c.drawLine(f((x,0)), f((x,1)),
                   color = "LightGreen", width = 1)
    for y in range(-10, 10):
        c.drawLine(f((0,y)), f((1,y)),
                   color = "LightGreen", width = 1)
    for P in diagPoints():
        c.drawPoint(P, color = "DeepPink")
        c.drawPoint(f(P), color = "DarkRed")
```

Die Python-Funktion `copysign` liefert den Wert ihres ersten Arguments mit dem Vorzeichen des zweiten. `copysign(3,-2)` gibt also z.B. das Ergebnis `-3` zurück. (Siehe auch Aufgabe 203.)

`copysign`

Lösung 506: Es gilt $f(\mathbf{0}) = (1,-2)^T$. Wäre f linear, so müsste jedoch $f(\mathbf{0}) = \mathbf{0}$ gelten.

Lösung 507: Die Matrizen zu den Abbildungen sind diese:

$$\boldsymbol{A}_f = \begin{pmatrix} 2 & -3 & 0 \\ -1 & 0 & 1 \end{pmatrix} \quad \text{und} \quad \boldsymbol{A}_g = \begin{pmatrix} 1 & 0 & 2 \\ 0 & 5 & -1 \\ 1 & -1 & 1 \end{pmatrix}$$

Die Matrix zu $f \circ g$ sieht folglich so aus:

$$\boldsymbol{A}_f \cdot \boldsymbol{A}_g = \begin{pmatrix} 2 & -15 & 7 \\ 0 & -1 & -1 \end{pmatrix}$$

$f \circ g$ ist eine Abbildung von $\mathbb{R}^3$ nach $\mathbb{R}^2$. $g \circ f$ kann man hingegen nicht bilden, weil der Wertebereich von f die Ebene $\mathbb{R}^2$ ist, der Definitionsbereich von g aber der Raum $\mathbb{R}^3$. Man erkennt das auch daran, dass man das Produkt $\boldsymbol{A}_g \cdot \boldsymbol{A}_f$ nicht bilden kann.

Lösung 508: Die Matrix für eine Drehung um 180° kennen wir schon aus Aufgabe 502:

$$\boldsymbol{D} = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}$$

Durch Nachrechnen überprüft man nun, dass tatsächlich $\boldsymbol{D} \cdot \boldsymbol{D} = \boldsymbol{E}_2$ stimmt.

Die Matrix für das Spiegeln an der y-Achse sieht so aus:

$$\boldsymbol{S} = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}$$

Auch hier ergibt sich sofort $\boldsymbol{S} \cdot \boldsymbol{S} = \boldsymbol{E}_2$.

Lösung 509: Die Matrizen für f_α und f_β sind diese:

$$\boldsymbol{D}_\alpha = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \qquad \boldsymbol{D}_\beta = \begin{pmatrix} \cos\beta & -\sin\beta \\ \sin\beta & \cos\beta \end{pmatrix}$$

Für $f_\alpha \circ f_\beta$ ergibt sich mit den Additionstheoremen dieses Produkt:

$$\begin{aligned} \boldsymbol{D}_\alpha \cdot \boldsymbol{D}_\beta &= \begin{pmatrix} \cos\alpha\cos\beta - \sin\alpha\sin\beta & -\cos\alpha\sin\beta - \sin\alpha\cos\beta \\ \sin\alpha\cos\beta + \cos\alpha\sin\beta & -\sin\alpha\sin\beta + \cos\alpha\cos\beta \end{pmatrix} \\ &= \begin{pmatrix} \cos(\alpha+\beta) & -\sin(\alpha+\beta) \\ \sin(\alpha+\beta) & \cos(\alpha+\beta) \end{pmatrix} \end{aligned}$$

Das ist eine Drehung um den Winkel $\alpha + \beta$.

Man rechnet leicht nach, dass $\boldsymbol{D}_\beta \cdot \boldsymbol{D}_\alpha$ dieselbe Matrix ist. Unabhängig von den Winkeln kommt es also bei der Komposition von Drehungen nicht auf die Reihenfolge an (was ja auch anschaulich zu erwarten war).

Lösung 510: Die Spiegelungsmatrizen sehen so aus:

$$\boldsymbol{S}_x = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \qquad \boldsymbol{S}_y = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}$$

Es ergibt sich:

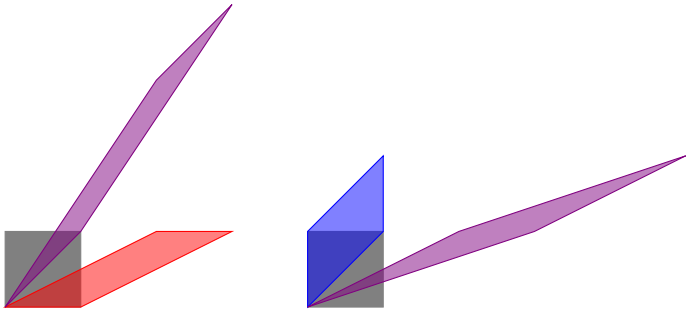
$$\boldsymbol{S}_x\boldsymbol{S}_y = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix} = \boldsymbol{S}_y\boldsymbol{S}_x$$

Es ist also egal, welche von den beiden Spiegelungen man zuerst durchführt.

Lösung 511: Die beiden Scherungen könnten z.B. so aussehen:

$$\boldsymbol{S}_1 = \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix} \qquad \boldsymbol{S}_2 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$$

Man rechnet nach, dass $\boldsymbol{S}_1\boldsymbol{S}_2$ und $\boldsymbol{S}_2\boldsymbol{S}_1$ verschieden sind. In diesem Fall kommt es also tatsächlich auf die Reihenfolge an. Wendet man beide Scherungen auf das (graue) Quadrat mit den Eckpunkten $(0,0)$ und $(1,1)$ an, so ergibt sich (links) durch Scherung längs der x-Achse erst das rote und dann durch Scherung längs der y-Achse daraus das violette Parallelogramm. Schert man jedoch erst nach oben, so ergibt sich (rechts) erst das blaue und dann ein *anderes* violettes Parallelogramm.



Lösung 513: Nach unseren Vorüberlegungen müssen wir eine Matrix finden, deren zugehörige lineare Abbildung nicht injektiv ist, weil sie (mindestens) eine Gerade auf einen Punkt abbildet. Die einfachste Antwort ist die Nullmatrix, denn sie gehört zu der Abbildung, die sämtliche Punkte der Ebene auf den Nullpunkt abbildet.

Man könnte aber auch die Projektion aus dem letzten Kapitel als Beispiel nehmen:

$$\boldsymbol{A} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

Geraden, die parallel zur y-Achse sind, werden durch diese Projektion auf einen Punkt abgebildet, daher ist sie nicht injektiv und die Matrix kann nicht regulär sein.

Man kann sogar „zu Fuß“ leicht ausrechnen, dass $\boldsymbol{A}$ nicht invertierbar ist. Dazu wählen wir uns eine beliebige 2×2-Matrix und multiplizieren diese mit $\boldsymbol{A}$:

$$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} b_{11} & b_{12} \\ 0 & 0 \end{pmatrix}$$

Völlig unabhängig davon, welche Werte in der Matrix stehen, an der rot markierten Stelle wird immer eine Null auftauchen. Daher kann rechts vom Gleichheitszeichen nie die Einheitsmatrix stehen.

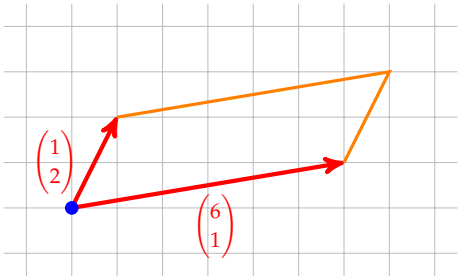
Lösung 516: Die inverse Matrix zu der aus (29.2) ist diese:

$$\boldsymbol{B} = \frac{1}{7} \cdot \begin{pmatrix} 2 & 1 \\ -1 & 3 \end{pmatrix}$$

Multipliziert man $\boldsymbol{B}$ mit $(5,2)^T$, so erhält man $1/7 \cdot (12,1)^T$, also zum Glück dasselbe Ergebnis wie in Aufgabe 497.

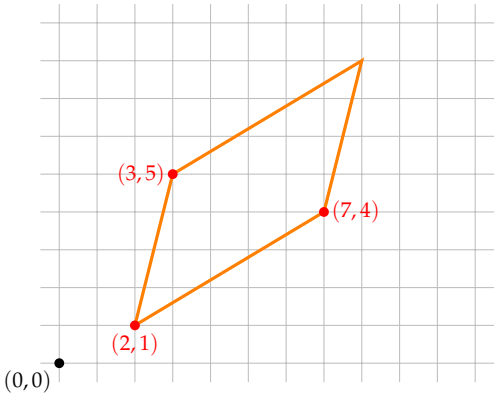
Lösung 517: Es ergibt sich $5 \cdot (-3) - 2 \cdot (-8) = 1$ für die erste Determinante und $42 \cdot 0 - 1/3 \cdot 6 = -2$ für die zweite.

Lösung 518: Legt man den Ursprung eines gedachten Koordinatensystems auf den blauen Punkt, so kann man die Koordinaten der beiden (roten) Vektoren ablesen, die das Parallelogramm aufspannen.



Als Fläche ergibt sich somit $6 \cdot 2 - 1 \cdot 1 = 11$.

Lösung 519: Grafisch sieht das so aus:



Man geht nun wie in der vorherigen Aufgabe vor und verschiebt den Ursprung auf den Punkt $(2,1)$. Als aufspannende Vektoren des Parallelogramms erhält man:

$$\begin{pmatrix}7\\4\end{pmatrix} - \begin{pmatrix}2\\1\end{pmatrix} = \begin{pmatrix}5\\3\end{pmatrix} \qquad \text{und} \qquad \begin{pmatrix}3\\5\end{pmatrix} - \begin{pmatrix}2\\1\end{pmatrix} = \begin{pmatrix}1\\4\end{pmatrix}$$

Daher ist die Fläche $5 \cdot 4 - 1 \cdot 3 = 17$. (Beachten Sie, dass wir zur Berechnung der Fläche die Koordinaten der vierten Ecke gar nicht zu ermitteln brauchten.)

Lösung 520: Die erste Determinante ist:

$$1 \cdot 7 \cdot (-4) + 3 \cdot 0 \cdot 2 + 5 \cdot 8 \cdot 1 - 5 \cdot 7 \cdot 2 - 3 \cdot 8 \cdot (-4) - 1 \cdot 0 \cdot 1 = 38$$

Im zweiten Fall ergibt sich null. (Die drei Vektoren liegen nämlich alle in einer Ebene, daher hat der Spat kein Volumen.)

Lösung 523: Man muss es nur ausrechnen. Für die Matrix

$$\boldsymbol{A} = \begin{pmatrix}a & b\\c & d\end{pmatrix}$$

gilt $\det(\boldsymbol{A}) = ad - bc$. Und für die transponierte Matrix erhält man $\det(\boldsymbol{A}^T) = ad - cb$. Diese beiden Werte sind natürlich identisch, weil $bc = cb$ gilt.

Lösung 524: Weil wir aus dem vorherigen Abschnitt wissen, dass sich die Determinante durch Transponieren nicht ändert. Daher sind in Bezug auf Determinanten Spalten und Zeilen gleichberechtigt.

Lösung 525: Es gilt $\det(\lambda \cdot \boldsymbol{A}) = \lambda^n \cdot \det(\boldsymbol{A})$. Das liegt daran, dass man *jede* der n Zeilen (oder Spalten) von $\boldsymbol{A}$ mit λ multiplizieren muss, um aus $\boldsymbol{A}$ die Matrix $\lambda \cdot \boldsymbol{A}$ zu machen.

Das Resultat sollte Sie nicht überraschen, weil Sie es im Prinzip schon kennen. Wenn Sie z.B. die Seitenlänge eines Quadrats verdoppeln, dann vervierfacht sich seine Fläche, wird also mit dem Faktor 2^2 multipliziert. Verdoppeln Sie die Seitenlänge eines Kubus, so verachtfacht sich sein Volumen, d.h. es wird mit 2^3 multipliziert. Das gilt ebenso für Parallelogramme und Spate.

Lösung 526: Nein, denn es ist ja $-\boldsymbol{A} = (-1) \cdot \boldsymbol{A}$ und damit nach der vorherigen Aufgabe $\det(-\boldsymbol{A}) = (-1)^n \cdot \det(\boldsymbol{A})$, wenn $\boldsymbol{A}$ eine $n \times n$-Matrix ist. $\det(-\boldsymbol{A}) = -\det(\boldsymbol{A})$ gilt also genau dann, wenn n ungerade ist. Anderenfalls gilt $\det(-\boldsymbol{A}) = \det(\boldsymbol{A})$.

Lösung 527: Das Volumen eines Spats erhält man, indem man die Fläche einer Seite mit der Höhe auf dieser Seite multipliziert. Wenn eine 3×3-Matrix obere Dreiecksform hat, so lässt sich aus den ersten beiden Spalten sofort eine Fläche des Spats berechnen und diese Fläche liegt in der x-y-Ebene. Für die Höhe auf dieser Seite ist dann nur noch die z-Komponente der dritten Spalte relevant.

Lösung 530: $\boldsymbol{A}$ beschreibt eine Drehung. Für die Determinante ergibt sich nach Pythagoras: $\cos^2 \pi/5 + \sin^2 \pi/5 = 1$. Das entspricht der Tatsache, dass Drehungen weder Flächen (bzw. Volumina) noch Orientierungen ändern.

$\boldsymbol{B}$ beschreibt eine Spiegelung und als Determinante ergibt sich -1. Spiegelungen ändern Flächeninhalte nicht, aber sie ändern Orientierungen.

$\boldsymbol{C}$ ist die Matrix einer Scherung längs der x-Achse. Die Determinante ist 1, denn so eine Scherung verändert Flächen nicht. (Falls Ihnen das nicht ohnehin klar ist, zeichnen Sie sich das Parallelogramm, in das das von den kanonischen Einheitsvektoren aufgespannte Quadrat durch die Scherung überführt wird.)

$\boldsymbol{D}$ schließlich beschreibt eine Streckung in x-Richtung um den Faktor 3. Dadurch verdreifachen sich natürlich alle Flächen und entsprechend hat die Determinante von $\boldsymbol{D}$ auch den Wert 3.

Die Determinanten von $\boldsymbol{C}$ und $\boldsymbol{D}$ sind natürlich beide positiv, weil sich keine Orientierungen ändern.

Lösung 532: Nein, natürlich nicht. Wir wissen ja schon, dass die Multiplikation von Matrizen nicht kommutativ ist.[54] Allerdings gilt natürlich immer

$$\det(\boldsymbol{A} \cdot \boldsymbol{B}) = \det(\boldsymbol{B} \cdot \boldsymbol{A}),$$

wenn $\boldsymbol{A}$ und $\boldsymbol{B}$ quadratische Matrizen derselben Größe sind. Das liegt einfach an der Kommutativität der „normalen“ Multiplikation.

Lösung 533: Eine Einheitsmatrix hat natürlich immer die Determinante 1. Es handelt sich um eine Dreiecksmatrix, also muss man einfach das Produkt der Diagonaleinträge bilden; und das sind alles Einsen.

Das ist aber auch geometrisch klar, weil eine Einheitsmatrix ja eine Abbildung beschreibt, die nichts ändert. Daher kann der zugehörige Vergrößerungsfaktor natürlich nur eins sein.

Lösung 534: Nach Definition der inversen Matrix ist $\boldsymbol{A} \cdot \boldsymbol{A}^{-1}$ die Einheitsmatrix $\boldsymbol{E}_n$ der entsprechenden Größe. Mit (30.4) und Aufgabe 533 erhält man dies:

$$1 = \det(\boldsymbol{E}_n) = \det(\boldsymbol{A} \cdot \boldsymbol{A}^{-1}) = \det(\boldsymbol{A}) \cdot \det(\boldsymbol{A}^{-1})$$

[54] Berechnen Sie aber bitte zur Probe die Matrix $\boldsymbol{B} \cdot \boldsymbol{A}$ und deren Determinante, falls Sie das nicht bereits getan haben.

Nun muss man nur noch nach $\det(\boldsymbol{A}^{-1})$ auflösen und bekommt als Ergebnis:[55]

$$\det(\boldsymbol{A}^{-1}) = 1/\det(\boldsymbol{A}) = \det(\boldsymbol{A})^{-1}$$

Lösung 536: Zum Beispiel so:

```
def dotProduct (X, Y):
    return sum(x * y for x, y in zip(X, Y))
```

`sum`

Die Funktion `sum` wurde u.a. auf Seite 245 besprochen.

Lösung 538: Man muss das jeweils nur mal hingeschrieben haben, um zu sehen, dass alle drei Eigenschaften wirklich offensichtlich sind. Wir machen das exemplarisch nur für die dritte im zweidimensionalen Fall:

$$\begin{aligned}\left(\begin{pmatrix}a_1\\a_2\end{pmatrix}+\begin{pmatrix}b_1\\b_2\end{pmatrix}\right)\cdot\begin{pmatrix}c_1\\c_2\end{pmatrix} &= \begin{pmatrix}a_1+b_1\\a_2+b_2\end{pmatrix}\cdot\begin{pmatrix}c_1\\c_2\end{pmatrix}\\ &= (a_1+b_1)\cdot c_1+(a_2+b_2)\cdot c_2\\ &= a_1c_1+b_1c_1+a_2c_2+b_2c_2\\ \begin{pmatrix}a_1\\a_2\end{pmatrix}\cdot\begin{pmatrix}c_1\\c_2\end{pmatrix}+\begin{pmatrix}b_1\\b_2\end{pmatrix}\cdot\begin{pmatrix}c_1\\c_2\end{pmatrix} &= a_1c_1+a_2c_2+b_1c_1+b_2c_2\end{aligned}$$

Lösung 540: Es ergibt sich:

$$\begin{aligned}(\boldsymbol{a}-\boldsymbol{b})^2 &= (\boldsymbol{a}-\boldsymbol{b})\cdot(\boldsymbol{a}-\boldsymbol{b})\\ &= \boldsymbol{a}\cdot(\boldsymbol{a}-\boldsymbol{b})-\boldsymbol{b}\cdot(\boldsymbol{a}-\boldsymbol{b})\\ &= \boldsymbol{a}\cdot\boldsymbol{a}-\boldsymbol{a}\cdot\boldsymbol{b}-\boldsymbol{b}\cdot\boldsymbol{a}+\boldsymbol{b}\cdot\boldsymbol{b}\\ &= \boldsymbol{a}^2-2\boldsymbol{a}\boldsymbol{b}+\boldsymbol{b}^2\end{aligned}$$

(Machen Sie sich für die einzelnen Zwischenschritte jeweils klar, dass man das wirklich tun darf. Teilweise habe ich zwei Schritte zusammengefasst oder stillschweigend die Kommutativität des Skalarprodukts ausgenutzt.)

Das ist natürlich nichts weiter als die zweite binomische Formel für das Skalarprodukt. Es ging bei dieser Aufgabe in erster Linie darum, Vertrautheit mit dem Skalarprodukt zu entwickeln.

Lösung 541: Es geht hier eigentlich nur darum, die Definition verstanden zu haben.

```
from math import sqrt

def norm (X):
    return sqrt(dotProduct(X, X))
```

Lösung 542: Mittels Aufgabe 538 kann man sich das auch ganz allgemein überlegen:

$$\|\lambda\boldsymbol{a}\| = \sqrt{(\lambda\cdot\boldsymbol{a})\cdot(\lambda\cdot\boldsymbol{a})} = \sqrt{(\lambda\cdot\lambda)\cdot(\boldsymbol{a}\cdot\boldsymbol{a})}$$

[55] Beachten Sie, dass der Exponent -1 hier zwei unterschiedliche Bedeutungen hat. (Welche?)

$$= \sqrt{\lambda^2} \cdot \sqrt{\boldsymbol{a} \cdot \boldsymbol{a}} = |\lambda| \cdot \|\boldsymbol{a}\|$$

Beachten Sie dabei, dass $\sqrt{\lambda^2}$ den Wert $|\lambda|$ und nicht etwa λ hat; siehe Aufgabe 225.

Lösung 543: $\boldsymbol{a}$ hat die Norm $\sqrt{3^2 + 4^2 + (-1)^2} = \sqrt{26}$. Wir wissen bereits, dass $\lambda \cdot \boldsymbol{a}$ für jeden Skalar $\lambda \neq 0$ parallel zu $\boldsymbol{a}$ ist und dass dieser Vektor auch noch in dieselbe Richtung wie $\boldsymbol{a}$ zeigt, wenn λ positiv ist. Wir müssen also offensichtlich $\boldsymbol{a}$ einfach mit $1/\sqrt{26}$ multiplizieren, dann erhalten wir mithilfe der vorherigen Aufgabe:

$$\left\| 1/\sqrt{26} \cdot \boldsymbol{a} \right\| = \left| 1/\sqrt{26} \right| \cdot \|\boldsymbol{a}\| = 1/\sqrt{26} \cdot \sqrt{26} = 1$$

Ganz allgemein kann man zu jedem Vektor $\boldsymbol{a}$, der nicht gerade der Nullvektor ist, einen Vektor finden, der in dieselbe Richtung zeigt, aber die Länge 1 hat: man nehme einfach $(1/\|\boldsymbol{a}\|) \cdot \boldsymbol{a}$. Man sagt dann auch, dass man $\boldsymbol{a}$ *normiert* hat.

Lösung 544: Auf Seite 294 kann man erkennen, das jeder einzelne Eintrag eines Produktes von Matrizen im Prinzip ein Skalarprodukt einer Zeile des ersten Faktors mit einer Spalte des zweiten ist. Im angesprochenen Beispiel ist etwa 8 das Skalarprodukt der zweiten Zeile und der dritten Spalte.

Insbesondere kann man das Skalarprodukt $\boldsymbol{a} \cdot \boldsymbol{b}$ auch als das Matrixprodukt $\boldsymbol{a}^T \cdot \boldsymbol{b}$ interpretieren. Dabei betrachtet man im zweiten Produkt die Vektoren wie auf Seite 296 als Matrizen mit nur einer Spalte. Das Ergebnis ist dann formal eine 1×1-Matrix, die wir einfach mit ihrem einzigen Eintrag identifizieren.

$$(a_1\ a_2\ a_3) \cdot \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = (a_1 b_1 + a_2 b_2 + a_3 b_3)$$

Lösung 545: Analog zur Funktion `vectorAdd` aus Kapitel 25 kann man eine Funktion zum Subtrahieren von Vektoren schreiben:

```
def vectorSub (A, B):
    return [a - b for a, b in zip(A, B)]
```

Damit ist die Aufgabe dann sofort gelöst:

```
def distance (A, B):
    return norm(vectorSub(A, B))
```

Lösung 546: Es ergeben sich ungefähr 42°:

```
from math import acos, pi

a = [32, 63]
b = [-4, 15]
180 * acos(dotProduct(a, b) / (norm(a) * norm(b))) / pi
```

Lösung 547: Ja, das gilt auch umgekehrt. Das liegt schlicht und einfach daran, dass das Skalarprodukt völlig symmetrisch definiert ist; beide Vektoren werden gleich behandelt.

Lösung 548: Bei einem Vektor in der Ebene ist es ganz einfach, einen dazu orthogonalen zu finden. Ist $(x,y)^T$ orthogonal zu $\boldsymbol{v}$, so gilt $3x+7y=0$ bzw. $3x=-7y$. Das lässt sich natürlich ganz einfach erreichen, indem man $x=7$ und $y=-3$ wählt oder umgekehrt $x=-7$ und $y=3$. Man vertauscht also einfach die erste mit der zweiten Komponente und ändert bei einer von beiden das Vorzeichen.[56] Skalare Vielfache dieser beiden Vektoren sind natürlich ebenfalls orthogonal zu $\boldsymbol{v}$, was ja auch geometrisch klar ist.

Falsch wäre es übrigens, als Antwort für diese Aufgabe den Nullvektor zu nennen. Zwar ist das Skalarprodukt von $\boldsymbol{v}$ und dem Nullvektor tatsächlich null, aber wie wir oben schon bemerkt haben, ergibt es keinen Sinn, über den Winkel zwischen dem Nullvektor und einem anderen Vektor zu sprechen.

Lösung 549: Sie sind hoffentlich auf eine Idee wie $(7,-3,0)^T$ gekommen. Man kann sich zwei Komponenten auswählen, mit denen wie in der vorherigen Aufgabe verfahren und die dritte Komponente verschwinden lassen. $(2,0,3)^T$ würde es also auch tun. Und übrigens auch jede Linearkombination von so erhaltenen Vektoren. Z.B. ist

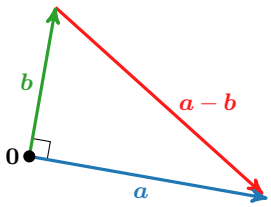
$$3\cdot\begin{pmatrix}7\\-3\\0\end{pmatrix}+5\cdot\begin{pmatrix}2\\0\\3\end{pmatrix}=\begin{pmatrix}31\\-9\\15\end{pmatrix}$$

auch ein Vektor, der orthogonal zu $\boldsymbol{v}$ ist. Können Sie mit den Rechengesetzen für das Skalarprodukt begründen, warum das so sein muss? Und ist Ihnen geometrisch klar, warum es im Raum so viele Vektoren gibt, die orthogonal zu einem vorgegeben sind?

Lösung 550: Wenn $\boldsymbol{a}$ und $\boldsymbol{b}$ orthogonal zueinander sind, ist $\boldsymbol{a}\cdot\boldsymbol{b}=0$. Das Ergebnis von Aufgabe 540 vereinfacht sich dann:

$$(\boldsymbol{a}-\boldsymbol{b})^2=\boldsymbol{a}^2+\boldsymbol{b}^2$$

Und das ist natürlich einfach der Satz des Pythagoras. Die Grafik von Seite 359 sieht für orthogonale Vektoren so aus:



Warum lässt sich der Satz mithilfe des Skalarproduktes durch eine einfache algebraische Umstellung beweisen? Weil wir ihn in unsere Definitionen quasi schon „eingebaut“ haben. Dass wir $\boldsymbol{a}^2$ als das Quadrat der Länge von $\boldsymbol{a}$ interpretieren, haben wir ja mit dem Satz des Pythagoras begründet.

[56] Das entspricht Drehungen um $\pm 90°$.

Wenn die beiden Vektoren nicht senkrecht aufeinander stehen, so entspricht die Darstellung aus Aufgabe 540 übrigens dem Kosinussatz.[57] Sehen Sie das?

Lösung 551: Die Antwort ist $(13, -14, -16)^T$.

Lösung 552: Es ergibt sich:

$$\begin{aligned}(\boldsymbol{a} \times \boldsymbol{b}) \cdot \boldsymbol{a} &= (a_2b_3 - a_3b_2)a_1 + (a_3b_1 - a_1b_3)a_2 + (a_1b_2 - a_2b_1)a_3 \\ &= a_1a_2b_3 - a_1a_3b_2 + a_2a_3b_1 - a_1a_2b_3 + a_1a_3b_2 - a_2a_3b_1 = 0\end{aligned}$$

$\boldsymbol{a} \times \boldsymbol{b}$ und $\boldsymbol{a}$ stehen also senkrecht aufeinander.

Lösung 553: Das erfordert nichts weiter als ein lästiges Abschreiben der Definition:

```
def crossProduct (A, B):
    return [A[1] * B[2] - A[2] * B[1],
            A[2] * B[0] - A[0] * B[2],
            A[0] * B[1] - A[1] * B[0]]
```

Lösung 554: Das Vektorprodukt ist *nicht* kommutativ, wie man mit so ziemlich jedem Beispiel leicht feststellen kann:

```
crossProduct([2, 3, -1], [4, -2, 5]),
crossProduct([4, -2, 5], [2, 3, -1])
```

Man sieht aber auch (und kann leicht nachrechnen), dass immer $\boldsymbol{a} \times \boldsymbol{b} = -(\boldsymbol{b} \times \boldsymbol{a})$ gilt. Man sagt in so einem Fall auch, dass die Verknüpfung *antikommutativ* ist.

Lösung 555: Da wir eine Python-Funktion haben, die uns die Rechenarbeit abnimmt, können wir das zunächst mal experimentell angehen:

```
crossProduct([2, 3, -1], [4, 6, -2])
```

Da kommt der Nullvektor heraus.

Und das ist immer der Fall. Sind $\boldsymbol{a}$ und $\boldsymbol{b}$ parallel, so kann man $\boldsymbol{b}$ als $\lambda\boldsymbol{a}$ darstellen. In Gleichung (31.2) eingesetzt ergibt sich dann:

$$\boldsymbol{a} \times (\lambda\boldsymbol{a}) = \begin{pmatrix} a_2\lambda a_3 - a_3\lambda a_2 \\ a_3\lambda a_1 - a_1\lambda a_3 \\ a_1\lambda a_2 - a_2\lambda a_1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Lösung 556: Gleichung (A.6) liefert uns einen Punkt auf der Geraden und einen Vektor der parallel zur Geraden ist, nämlich $\boldsymbol{p} = (1, 2)^T$ und $\boldsymbol{v} = (-4, -1)^T$. Mit der Technik aus Aufgabe 548 bekommen wir einen Vektor, der senkrecht zu $\boldsymbol{v}$ ist, z.B. $\boldsymbol{n} = (-1, 4)^T$. Damit können wir das Produkt $k = \boldsymbol{n} \cdot \boldsymbol{p} = 7$ berechnen und kommen so insgesamt auf diese Darstellung der Geraden:

$$\{\boldsymbol{x} \in \mathbb{R}^2 : \begin{pmatrix} -1 \\ 4 \end{pmatrix} \cdot \boldsymbol{x} - 7 = 0\}$$

[57] Siehe Aufgabe 413.

Setzen Sie versuchsweise ein paar Punkte ein.

Lösung 557: Dafür müssen wir zunächsten den Normalenvektor $\boldsymbol{n} = (-1,4)^T$ normieren. Mit $\|\boldsymbol{n}\| = \sqrt{1^2+4^2} = \sqrt{17}$ erhalten wir $\boldsymbol{n}_0 = 1/\sqrt{17} \cdot (-1,4)^T$. Diesen Vektor multiplizieren wir nun mit einem Punkt auf der Geraden wie z.B. $\boldsymbol{p} = (1,2)^T$, um den Abstand zur Geraden zu erhalten. Das ergibt $7/\sqrt{17}$. Da dieser Wert positiv ist, ist auch die Richtung von $\boldsymbol{n}_0$ korrekt. (Anderenfalls hätten wir noch mit -1 multiplizieren müssen.) Die hessesche Normalenform sieht also so aus:

$$\left\{\boldsymbol{x} \in \mathbb{R}^2 : \frac{1}{\sqrt{17}} \cdot \begin{pmatrix} -1 \\ 4 \end{pmatrix} \cdot \boldsymbol{x} - \frac{7}{\sqrt{17}} = 0\right\}$$

Lösung 558: Das kann man an der Skizze auf Seite 365 ablesen. Ist der Wert positiv, so liegt der Punkt auf der Seite der Geraden g, auf die $\boldsymbol{n}_0$ von g aus zeigt. Oder anders ausgedrückt: auf der Seite von g, auf der der Ursprung *nicht* liegt.

Lösung 559: Dafür muss man den Punkt nur in die Geradengleichung einsetzen, also den Wert

$$\frac{1}{\sqrt{17}} \cdot \left(\begin{pmatrix} -1 \\ 4 \end{pmatrix} \cdot \begin{pmatrix} -2 \\ 1 \end{pmatrix} - 7\right)$$

berechnen. Das Ergebnis ist $-1/\sqrt{17}$. Der Abstand ist also $1/\sqrt{17}$ und der Punkt liegt auf derselben Seite der Geraden wie der Ursprung.

Lösung 560: In diesem Fall vereinfacht sich die Darstellung, weil der Abstand zum Nullpunkt verschwindet:

$$\{\boldsymbol{x} \in \mathbb{R}^2 : \boldsymbol{n}_0 \cdot \boldsymbol{x} = 0\}$$

Der „Nachteil“ ist allerdings, dass man sozusagen die Orientierung verliert. Man kann immer noch zwei Seiten der Geraden durch das Vorzeichen des Produktes $\boldsymbol{n}_0 \cdot \boldsymbol{x}$ unterscheiden. Aber es gibt nicht mehr eine Seite, die dadurch ausgezeichnet ist, dass dort der Ursprung liegt. Beide sind gleichberechtigt.

Lösung 561: Wir haben zwei Richtungsvektoren. Aus denen basteln wir uns mithilfe des Vektorprodukts einen Vektor $\boldsymbol{n}$, der senkrecht auf beiden steht:

```
crossProduct([2, -3, 4], [3, -2, -7])
```

Wenn wir $\boldsymbol{n}$ nun noch normieren, erhalten wir:

$$\boldsymbol{n}_0 = \frac{1}{\sqrt{1542}} \cdot \begin{pmatrix} 29 \\ 26 \\ 5 \end{pmatrix}$$

Nun multiplizieren wir $\boldsymbol{n}_0$ mit einem Punkt[58] der Geraden und erhalten $86/\sqrt{1542}$. Da dieser Wert positiv ist, hat $\boldsymbol{n}_0$ bereits die richtige Richtung. Wir erhalten als Darstellung der Ebene:

$$\left\{ \boldsymbol{x} \in \mathbb{R}^3 : \frac{1}{\sqrt{1542}} \cdot \left(\begin{pmatrix} 29 \\ 26 \\ 5 \end{pmatrix} \cdot \boldsymbol{x} - 86 \right) = 0 \right\}$$

Lösung 562: Es gilt $\boldsymbol{v} \cdot \boldsymbol{w} = 1 + 1 = 2$. Mit $f(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x}$ gilt jedoch

$$f(\boldsymbol{v}) = f(\boldsymbol{w}) = \begin{pmatrix} \sqrt{2} \\ \sqrt{2} \end{pmatrix}$$

und daher $f(\boldsymbol{v}) \cdot f(\boldsymbol{w}) = 2 + 2 = 4 \neq 2$.

Lösung 563: Mit $\boldsymbol{v} = (v_1, v_2)^T$ und $\boldsymbol{w} = (w_1, w_2)^T$ ergibt sich:

$$\begin{aligned}
\boldsymbol{A}\boldsymbol{v} &= \frac{1}{\sqrt{2}} \begin{pmatrix} v_1 - v_2 \\ v_1 + v_2 \end{pmatrix} \\
\boldsymbol{A}\boldsymbol{w} &= \frac{1}{\sqrt{2}} \begin{pmatrix} w_1 - w_2 \\ w_1 + w_2 \end{pmatrix} \\
(\boldsymbol{A}\boldsymbol{v}) \cdot (\boldsymbol{A}\boldsymbol{w}) &= \left(\frac{1}{\sqrt{2}} \begin{pmatrix} v_1 - v_2 \\ v_1 + v_2 \end{pmatrix} \right) \cdot \left(\frac{1}{\sqrt{2}} \begin{pmatrix} w_1 - w_2 \\ w_1 + w_2 \end{pmatrix} \right) \\
&= 1/2 \cdot ((v_1 - v_2) \cdot (w_1 - w_2) + (v_1 + v_2) \cdot (w_1 + w_2)) \\
&= 1/2 \cdot (v_1 w_1 - v_2 w_1 - v_1 w_2 + v_2 w_2 + v_1 w_1 + v_2 w_1 + v_1 w_2 + v_2 w_2) \\
&= 1/2 \cdot (v_1 w_1 + v_2 w_2 + v_1 w_1 + v_2 w_2) = v_1 w_1 + v_2 w_2 = \boldsymbol{v} \cdot \boldsymbol{w}
\end{aligned}$$

Lösung 564: Die entsprechenden Rechnungen sehen so aus:

$$\begin{aligned}
\mathrm{d}(f(\boldsymbol{v}), f(\boldsymbol{w})) &= \|f(\boldsymbol{v}) - f(\boldsymbol{w})\| = \|f(\boldsymbol{v} - \boldsymbol{w})\| \\
&= \|\boldsymbol{v} - \boldsymbol{w}\| = \mathrm{d}(\boldsymbol{v}, \boldsymbol{w}) \\
\measuredangle(f(\boldsymbol{v}), f(\boldsymbol{w})) &= \arccos \frac{f(\boldsymbol{v}) \cdot f(\boldsymbol{w})}{\|f(\boldsymbol{v})\| \cdot \|f(\boldsymbol{w})\|} \\
&= \arccos \frac{\boldsymbol{v} \cdot \boldsymbol{w}}{\|\boldsymbol{v}\| \cdot \|\boldsymbol{w}\|} = \measuredangle(\boldsymbol{v}, \boldsymbol{w})
\end{aligned}$$

Wie üblich sollten Sie sich bei jeder einzelnen Umformung klarmachen, warum sie erlaubt ist.

Lösung 565: Falls Ihnen nicht klar war, was zu tun ist: Sie sollen $\boldsymbol{A}^T \cdot \boldsymbol{A}$ ausrechnen. Das Ergebnis muss, wenn Sie sich nicht verrechnen, $\boldsymbol{E}_2$ sein.

Lösung 566: Für die Drehmatrix $\boldsymbol{D}_\alpha$ ergibt sich mithilfe des Satzes von Pythagoras:

$$\boldsymbol{D}_\alpha^T \cdot \boldsymbol{D}_\alpha = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \cdot \begin{pmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{pmatrix}$$

[58] Ihnen ist klar, dass es egal ist, welchen wir nehmen, weil ohnehin für alle dasselbe herauskommen muss?

$$= \begin{pmatrix} \cos^2\alpha + \sin^2\alpha & \cos\alpha \cdot \sin\alpha - \sin\alpha \cdot \cos\alpha \\ \sin\alpha \cdot \cos\alpha - \cos\alpha \cdot \sin\alpha & \sin^2\alpha + \cos^2\alpha \end{pmatrix}$$
$$= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \boldsymbol{E}_2$$

Und für $\boldsymbol{S}_\alpha$ führt fast exakt dieselbe Rechnung ebenfalls auf $\boldsymbol{E}_2$.

Lösung 567: Im Prinzip müsste es so gehen:

```
def identity (n):
    A = [[0] * n for i in range(n)]
    for i in range(n):
        A[i][i] = 1
    return A

def isOrthogonal (A):
    return identity(len(A)) == matrixProd(transpose(A), A)
```

Sie werden aber sehen, dass das hier (siehe Aufgabe 563) z.B. nicht wie erwartet `True`, sondern `False` ergibt:

```
from math import sqrt

isOrthogonal(scalarMatrixMult(1/sqrt(2), [[1,-1], [1,1]]))
```

Das liegt natürlich wieder mal an den lästigen Rundungsfehlern, die wir der Fließkomma-Arithmetik zu verdanken haben. Man könnte das Problem z.B. so lösen:

```
from math import isclose

def matrixClose (A, B):
    return all(all(isclose(a, b)
                   for a, b in zip(rowA, rowB))
               for rowA, rowB in zip(A, B))

def isOrthogonal (A):
    return matrixClose(identity(len(A)),
                       matrixProd(transpose(A), A))
```

Allerdings weiß man dann auch nur „ungefähr“, ob die Matrix orthogonal ist. Besser klappt das mit einem Computeralgebrasystem wie SymPy:[59]

`eye` `shape`

[59] Falls Ihnen nicht klar ist, was die Methode `shape` bzw. die Funktion `eye` machen, schauen Sie bitte in der Dokumentation nach.

```
from sympy import *

def isOrthogonal (M):
    return eye(M.shape[0]) == M.T * M

M = Matrix([[1, -1], [1, 1]])
isOrthogonal(M), isOrthogonal(1 / sqrt(2) * M)
```

Lösung 568: Es gibt verschiedene Wege, das zu begründen. Man könnte z.B. damit argumentieren, dass die Komposition von zwei Funktionen, die das Skalarprodukt nicht ändern, natürlich auch nicht das Skalarprodukt ändern kann. Man kann es aber auch mit dem gerade gelernten Kriterium machen: Da $\boldsymbol{A}$ und $\boldsymbol{B}$ beide orthogonal sind, gilt $\boldsymbol{A}^T \cdot \boldsymbol{A} = \boldsymbol{E}_n$ und $\boldsymbol{B}^T \cdot \boldsymbol{B} = \boldsymbol{E}_n$. Nach Gleichung (26.2) gilt dann:

$$\begin{aligned}(\boldsymbol{A} \cdot \boldsymbol{B})^T \cdot (\boldsymbol{A} \cdot \boldsymbol{B}) = (\boldsymbol{B}^T \cdot \boldsymbol{A}^T) \cdot (\boldsymbol{A} \cdot \boldsymbol{B}) &= \boldsymbol{B}^T \cdot (\boldsymbol{A}^T \cdot \boldsymbol{A}) \cdot \boldsymbol{B} \\ &= \boldsymbol{B}^T \cdot \boldsymbol{E}_n \cdot \boldsymbol{B} = \boldsymbol{B}^T \cdot \boldsymbol{B} = \boldsymbol{E}_n\end{aligned}$$

Die Matrix $\boldsymbol{A} \cdot \boldsymbol{B}$ ist somit auch orthogonal.

Lösung 569: Nein, diese Aussage gilt nicht. Ein einfaches Gegenbeispiel liefert die Scherungsmatrix $\boldsymbol{S} = \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}$. Ihre Determinante ist 1 (sie ändert ja auch die Flächeninhalte nicht), aber sie ist nicht orthogonal, denn $\boldsymbol{S}^T \cdot \boldsymbol{S}$ ist nicht die Einheitsmatrix.

Lösung 570: Wie im vorherigen Beispiel verschieben wir zunächst so, dass der Punkt, der sich durch die folgende Abbildung (in diesem Fall die Spiegelung) nicht bewegen soll, auf den Nullpunkt fällt. Es bietet sich an, um zwei Einheiten nach links zu schieben, also um den Vektor $\boldsymbol{v} = (-2, 0)^T$.[60] Nun müssen wir an der y-Achse spiegeln. Die zugehörige Matrix können wir mit (29.6) erstellen. In diesem Fall ist es aber wohl einfacher, sich zu überlegen, wohin die kanonischen Einheitsvektoren gespiegelt werden; das ergibt ja die Spalten der Matrix. Die sieht dann so aus:

$$\boldsymbol{S} = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}$$

Mit $g(\boldsymbol{x}) = \boldsymbol{S} \cdot \boldsymbol{x}$ können wir nun f zerlegen als $f = \tau_{-\boldsymbol{v}} \circ g \circ \tau_{\boldsymbol{v}}$. In Python:

```
from matrices import Matrix

S = Matrix([[-1, 0], [0, 1]])
v = Vector(-2, 0)

        g = lambda x: S * Vector(x)
     tauV = lambda x: Vector(x) + v
```

[60] Das ist aber im Gegensatz zum Beispiel mit der Drehung nicht die einzige Möglichkeit. Wir hätten theoretisch jeden Punkt auf der Spiegelachse auf den Nullpunkt verschieben können.

```
tauMinusV = lambda x: Vector(x) - v
        f = lambda x: tauMinusV(g(tauV(x)))
```

Lösung 571: Der Punkt hat natürlich die Koordinaten $(0, 0, 1)$.

Lösung 572: Die Punkte auf der Geraden haben alle die Form $(3\lambda, 4\lambda, 5\lambda)$. Da die Punkte auf der Ebene alle als dritte Komponente den Wert eins haben, muss $5\lambda = 1$ gelten, woraus $\lambda = 1/5$ folgt. Also hat der Schnittpunkt die Koordinaten $(3/5, 4/5, 1)$.

Lösung 573: Ja, man muss ja einfach nur den Punkt auf der Ebene mit dem Nullpunkt verbinden.

Lösung 574: Nein. Die Geraden, die in der x-y-Ebene liegen, verlaufen parallel zu E und haben daher keinen Schnittpunkt mit E. In der bereits angesprochenen projektiven Geometrie entsprechen dieses Geraden den „unendlich fernen“ Punkten der Ebene. Für unsere Zwecke sind sie aber unerheblich.

Lösung 575: Das muss so aussehen:

$$\begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} & 0 \\ 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Zur Erinnerung: $\sin 45^\circ = \cos 45^\circ = 1/\sqrt{2}$.

Lösung 577: Es handelt sich offenbar um die durch $\boldsymbol{x} \mapsto \boldsymbol{x} + (3, -2)^T$ gegebene Translation. Die gesuchte Matrix sieht also so aus:

$$\begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{bmatrix}$$

Lösung 580: Die Matrix sieht folgendermaßen aus:

$$\begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 4 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Im PYTHON-Code, den wir bereits haben, müssen wir nur `M` ändern:

```
M = Matrix([[-1, 0, 4], [0, 1, 0]])
```

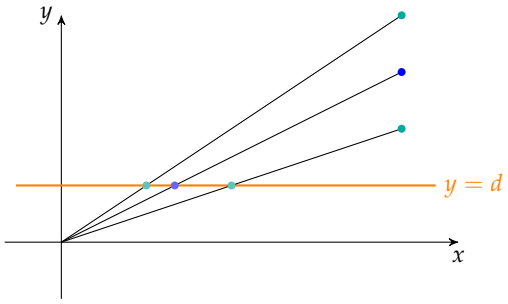
Lösung 581: Die Ortsvektoren der Punkte auf g haben alle die Form $\lambda \cdot (21, 126, 42)^T$. Die Punkte auf E haben alle die z-Koordinate 1. Mit $\lambda \cdot 42 = 1$ folgt $\lambda = 1/42$. Also hat der gesuchte Punkt die folgenden Koordinaten:

$$(1/42 \cdot 21, 1/42 \cdot 126, 1/42 \cdot 42) = (1/2, 3, 1)$$

In diesem Sinne sind sowohl $[21, 126, 42]$ als auch $[1/2, 3, 1]$ homogene Koordinaten desselben Punktes $(1/2, 3)$.

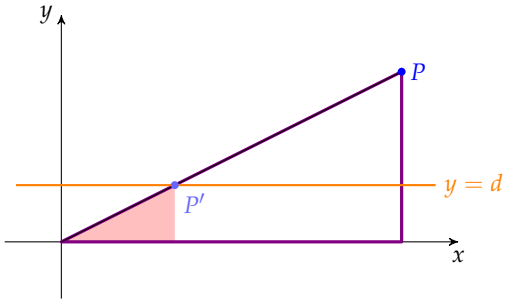
Lösung 582: Die vorherige Aufgabe sollte deutlich gemacht haben, dass die Koordinaten des Punktes $(x/z, y/z)$ sind. Natürlich darf z nicht null sein, aber die Ebene durch den Nullpunkt kommt ja aus offensichtlichen Gründen auch nicht infrage.

Lösung 583: Das sollte so aussehen:



Man sieht sofort, dass nach der Projektion der blaue Punkt nicht mehr genau in der Mitte zwischen den beiden grünen Punkten liegt.

Lösung 584: Es handelt sich um das rosa gefüllte und das violett umrandete Dreieck in der Skizze unten.



Lösung 585: Alle Bildpunkte haben die folgende Form:

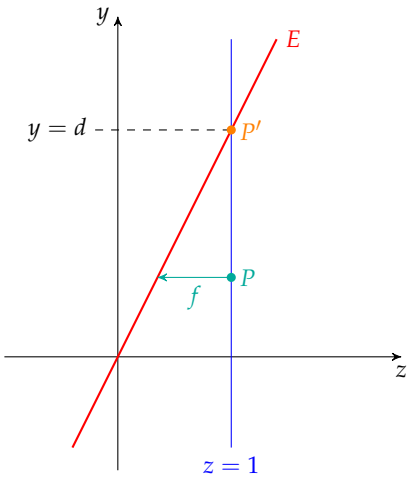
$$\begin{pmatrix} x \\ y \\ y/d \end{pmatrix} = x \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + y \cdot \begin{pmatrix} 0 \\ 1 \\ 1/d \end{pmatrix}$$

f ist also eine Projektion des Raums parallel zur z-Achse[61] auf diese Ebene E:

$$E = \mathbb{R} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + \mathbb{R} \cdot \begin{pmatrix} 0 \\ 1 \\ 1/d \end{pmatrix} = \mathbb{R} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + \mathbb{R} \cdot \begin{pmatrix} 0 \\ d \\ 1 \end{pmatrix}$$

Von der Seite (also parallel zur x-Achse) gesehen sieht es so aus:

[61] Der Bildpunkt hängt nicht von der z-Koordinate ab. Punkte, die sich nur in der z-Koordinate unterscheiden, haben also denselben Bildpunkt.



Hier wird exemplarisch ein Punkt P auf E abgebildet. In homogenen Koordinaten interessiert uns aber nur der Schnittpunkt der Geraden durch $f(P)$ und den Nullpunkt mit der Ebene $z = 1$. Das ist der Punkt P' in der Skizze. Er ist der Schnittpunkt der Ebene $z = 1$ mit der Geraden $y = d$ in der Skizze von Seite 381.

Lösung 586: Die Kamera wird nach links geschoben und dann gedreht. Das lässt sich einfacher darstellen, wenn man erst (um den Nullpunkt) dreht und dann verschiebt. In homogenen Koordinaten sieht das so aus:

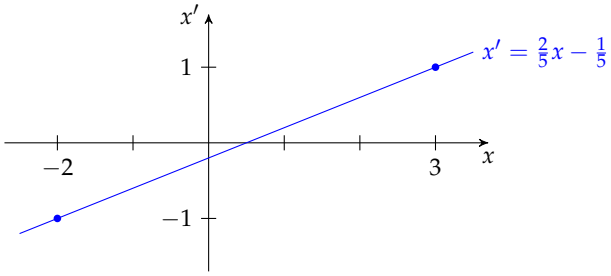
$$\begin{bmatrix} 1 & 0 & -3 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos \pi/6 & -\sin \pi/6 & 0 \\ \sin \pi/6 & \cos \pi/6 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \frac{1}{2} \cdot \begin{bmatrix} \sqrt{3} & -1 & -6 \\ 1 & \sqrt{3} & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

Um die neuen Koordinaten des Punktes $(4, 1)$ zu bekommen, muss man die obige Matrix invertieren und $[4, 1, 1]^T$ mit dieser Inversen multiplizieren:

$$\frac{1}{2} \cdot \begin{bmatrix} \sqrt{3} & 1 & 3\sqrt{3} \\ -1 & \sqrt{3} & -3 \\ 0 & 0 & 2 \end{bmatrix} \cdot \begin{bmatrix} 4 \\ 1 \\ 1 \end{bmatrix} = \frac{1}{2} \cdot \begin{bmatrix} 1 + 7\sqrt{3} \\ -7 + \sqrt{3} \\ 2 \end{bmatrix} \approx \begin{bmatrix} 6.6 \\ -2.6 \\ 1.0 \end{bmatrix}$$

Aus Sicht der roten Kamera hat der grüne Punkt also ungefähr die Koordinaten $(6.6, -2.6)$.

Lösung 587: Zunächst mal kann man so eine Umrechnung mit simplem Schulwissen (Dreisatz!) durchführen. Um z.B. vom `left-right`-Intervall $[-2, 3]$ auf das NDC-Intervall $[-1, 1]$ umzurechnen, reicht eine ganz einfache Geradengleichung:



Da wir später jedoch möglichst alle Operationen in einer Matrix zusammenfassen wollen, ist es sinnvoll, auch diesen Umrechnungsprozess mit homogenen Koordinaten darzustellen. Die eben gezeigte Operation kann man sich nämlich auch in zwei Teile zerlegt vorstellen:

Zuerst wird das Intervall $[-2,3]$ durch Verschieben zentriert, indem man seinen Mittelpunkt auf den Nullpunkt verschiebt. Das entspricht dem Subtrahieren der Zahl $(-2+3)/2 = 1/2$. Dann wird das Intervall mit dem Faktor $2/5$ skaliert, so dass aus der ursprünglichen Breite von 5 die normalisierte Breite 2 wird:

$$x' = 2/5 \cdot (x - 1/2) = 2/5 \cdot x - 1/5$$

Führt man diese beiden Prozesse simultan für alle drei Koordinaten durch, dann sieht es als Matrix so aus:

$$\begin{bmatrix} 2/(3-(-2)) & 0 & 0 & 0 \\ 0 & 2/(2-(-5)) & 0 & 0 \\ 0 & 0 & 2/(11-2) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & -(-2+3)/2 \\ 0 & 1 & 0 & -(-5+2)/2 \\ 0 & 0 & 1 & -(2+11)/2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
$$= \begin{bmatrix} 2/5 & 0 & 0 & -1/5 \\ 0 & 2/7 & 0 & 3/7 \\ 0 & 0 & 2/9 & -13/9 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(Sie erkennen hoffentlich in der ersten Zeile die Geradengleichung von eben wieder.)

Lösung 588: Um den Ort eines projizierten Punktes auf dem Bildschirm zu finden, braucht man die z-Koordinate tatsächlich nicht. Allerdings liefert diese Koordinate Informationen über die Lage von Objekten zueinander; konkret: welche Objekte aus Sicht der Kamera *vor* oder *hinter* anderen liegen. Für eine realistische Darstellung ist es natürlich unabdingbar, dass Gegenstände einander verdecken können. Das ist aber ein fortgeschrittenes Thema, mit dem wir uns hier leider nicht beschäftigen können.

Lösung 590: Das gezeigte Bild ist unabhängig vom Abstand immer gleich. Das liegt an der Parallelprojektion. Ein Blick auf die Skizze auf Seite 385 sollte das deutlich machen: Die Projektionslinien ändern sich nicht, wenn man sich dem Objekt nähert oder

sich von ihm entfernt. (Und was soll überhaupt „nähern“ bedeuten? In der Skizze ist kein Auge, das sich „nähern“ könnte.)

Lösung 591: Man müsste zumindest die Zeile `c.drawSegment(P1, P2)` durch das hier ersetzen:

```
if -10 <= P1[2] <= 10 and -10 <= P2[2] <=10:
    c.drawSegment(P1, P2)
```

Das würde dafür sorgen, dass Kanten, die nach vorne oder hinten aus dem Quader herausragen, gar nicht gezeichnet werden.[62] Für die x- und y-Koordinaten ist so ein Test nicht nötig, weil die Grenzen des Bildschirms dafür sorgen, dass wir keine Objekte außerhalb des Quaders sehen.

Lösung 592: Man muss am Ende mit dieser Matrix multiplizieren:

$$\begin{bmatrix} 256/5 & 0 & 0 & 512 \\ 0 & -192/5 & 0 & 384 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Die dritte Zeile ändert die z-Koordinate nicht; die wird danach also immer noch zwischen -10 und 10 liegen.

Lösung 593: Das Ergebnis für den ersten Punkt ist:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & A & B \\ 0 & 0 & 1/\texttt{near} & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ \texttt{near} \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ A \cdot \texttt{near} + B \\ 1 \end{bmatrix}$$

Das entspricht dem Punkt $(x, y, A \cdot \texttt{near} + B)$.

Für den zweiten Punkt erhalten wir:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & A & B \\ 0 & 0 & 1/\texttt{near} & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ \texttt{far} \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ A \cdot \texttt{far} + B \\ \texttt{far}/\texttt{near} \end{bmatrix}$$

Rechnen wir wie in Aufgabe 582 um, so entspricht das diesem Punkt:

$$(\texttt{near} \cdot (x/\texttt{far}), \texttt{near} \cdot (y/\texttt{far}), A \cdot \texttt{near} + \texttt{near} \cdot (B/\texttt{far}))$$

Lösung 594: Weil die z-Koordinate eines projizierten Punktes offenbar nicht von seinen ursprünglichen x- und y-Koordinaten abhängt.

[62] Vielleicht ist das jedoch ein bisschen radikal. Eleganter, aber auch aufwendiger, wäre es, solche Kanten an den Quadergrenzen abzuschneiden.

Lösung 595: Wir multiplizieren $\boldsymbol{M}$ mit den homogenen Koordinaten des Punktes:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & (\texttt{near}+\texttt{far})/\texttt{near} & -\texttt{far} \\ 0 & 0 & 1/\texttt{near} & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$
$$= \begin{bmatrix} x \\ y \\ (z/\texttt{near}) \cdot (\texttt{near}+\texttt{far}) - \texttt{far} \\ z/\texttt{near} \end{bmatrix}$$

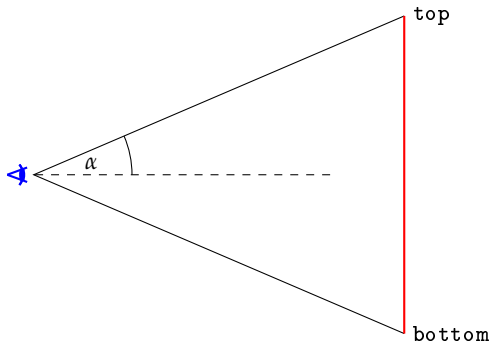
Nach Aufgabe 582 ist das ein Punkt mit der z-Koordinate $\texttt{near} + \texttt{far} - \texttt{far} \cdot \texttt{near}/z$.

Lösung 596: Die Matrix $\boldsymbol{M}$ soll ja nur auf Werte innerhalb des Pyramidenstumpfes angewendet werden. Die liegen aber im Blickfeld der Kamera und werden im Koordinatensystem der Kamera angegeben. Daher sind die z-Koordinaten solcher Punkte immer positiv.

Lösung 597: Hier ein Beispiel:

```
z = 0
for angle in range(-10, 50):
    D = translationMatrix(0, 0, z) * \
        xRotationMatrix(angle / 180 * pi) * \
        yRotationMatrix(2 * angle / 180 * pi)
    edges = [[lower(D * lift(P1)),
              lower(D * lift(P2))] for P1, P2 in cubeEdges]
    drawCube(c, edges = edges)
    z -= 0.05
    sleep(0.1)
```

Lösung 598: Den Bildwinkel kann man so visualisieren:



Die Entfernung vom „Auge“ zur roten Bildschirmfläche ist `near`. Daher gilt für das Seitenverhältnis $\tan\alpha = \texttt{top}/\texttt{near}$. Mit $\alpha = 25°$ folgt:

$$\texttt{top} = \texttt{near} \cdot \tan\alpha \approx 4.2 \cdot 0.466 \approx 1.957$$

Mit dem Seitenverhältnis von $4/3$ folgt dann:

$$\texttt{right} = 4/3 \cdot \texttt{top} \approx 1.333 \cdot 1.957 \approx 2.609$$

Wegen der Symmetrie gilt ferner $\texttt{bottom} = -\texttt{top}$ und $\texttt{left} = -\texttt{right}$.

Lösung 600: Man kann das mit partieller Integration lösen (oder sich von einem CAS helfen lassen). Eine Stammfunktion von $(\sin x)(\cos x)$ ist $-1/2 \cdot \cos^2 x$. Als bestimmtes Integral erhält man also:

$$-1/(2\pi) \cdot \cos^2 x \Big|_{-\pi}^{\pi} = 1/(2\pi) \left(\cos^2(-\pi) - \cos^2 \pi\right) = 0$$

Lösung 601: Wir haben schon diverse endliche Körper kennengelernt: $\mathbb{Z}_2$, $\mathbb{Z}_3$, $\mathbb{Z}_5$, $\mathbb{Z}_7$, und so weiter. Im nächsten Kapitel machen wir noch Bekanntschaft mit dem Körper $\mathbb{C}$ der komplexen Zahlen. Vielleicht kennen Sie den ja schon aus der Schule.

Und am Ende von Kapitel 59 lernen wir noch eine weitere Klasse von endlichen Körpern kennen, die auch für technische Anwendungen wichtig ist.

Lösung 602: Weil man nicht dividieren kann. Z.B. sind 2 und 3 ganze Zahlen, aber ihr Quotient ist keine ganze Zahl.

Lösung 603: Eigentlich ganz einfach. Man muss nur immer daran denken, dass alle arithmetischen Operationen in $\mathbb{Z}_5$ durchgeführt werden:

$$\boldsymbol{a} + \boldsymbol{b} = \begin{pmatrix} 3 \\ 2 \end{pmatrix} + \begin{pmatrix} 2 \\ 4 \end{pmatrix} = \begin{pmatrix} 3+2 \\ 2+4 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$
$$3 \cdot \boldsymbol{a} = 3 \cdot \begin{pmatrix} 3 \\ 2 \end{pmatrix} = \begin{pmatrix} 3 \cdot 3 \\ 3 \cdot 2 \end{pmatrix} = \begin{pmatrix} 4 \\ 1 \end{pmatrix}$$

Lösung 604: Auch hier geht es im Endeffekt nur darum, nicht zu vergessen, dass alle Berechnungen in $\mathbb{Z}_5$ durchgeführt werden müssen.

$$\boldsymbol{M}\boldsymbol{v} = \begin{pmatrix} 1 \cdot 2 + 2 \cdot 0 + 3 \cdot 1 \\ 3 \cdot 2 + 0 \cdot 0 + 2 \cdot 1 \\ 2 \cdot 2 + 2 \cdot 0 + 4 \cdot 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 3 \\ 3 \end{pmatrix}$$

Lösung 605: Die Matrix hat schon an der richtigen Stelle ein Pivotelement:

$$\begin{pmatrix} \textcircled{1} & 2 & 3 \\ 3 & 0 & 2 \\ 2 & 2 & 4 \end{pmatrix}$$

Wir subtrahieren nun von der zweiten Zeile das Dreifache der ersten und von der dritten das Doppelte der ersten:[63]

$$\begin{pmatrix} 1 & 2 & 3 \\ 0 & 4 & 3 \\ 0 & 3 & 3 \end{pmatrix}$$

[63] Klar, wieso z.B. in der zweiten Zeile an der dritten Stelle 3 steht? In $\mathbb{Z}_5$ gilt $2 - 3 \cdot 3 = 3$.

Nun dividieren wir die zweite Zeile durch 4, damit wir ein „schönes“ Pivotelement bekommen. Beachten Sie, dass wir als Ergebnis 2 erhalten, wenn wir in $\mathbb{Z}_5$ die Zahl 3 durch 4 teilen.

$$\begin{pmatrix} 1 & 2 & 3 \\ 0 & \textcircled{1} & 2 \\ 0 & 3 & 3 \end{pmatrix}$$

Und dann noch der letzte Schritt:

$$\begin{pmatrix} 1 & 2 & 3 \\ 0 & 1 & 2 \\ 0 & 0 & 2 \end{pmatrix}$$

Wir hätten das aber auch anders machen können. Erinnern Sie sich, dass wir immer von der modularen zur „normalen“ Arithmetik und zurück „springen“ dürfen. Wir können also die Matrix zunächst so behandeln, als würden wir in $\mathbb{Z}$ (bzw. $\mathbb{Q}$) rechnen und erst ganz am Ende alles in $\mathbb{Z}_5$ umrechnen:

$$\begin{pmatrix} \textcircled{1} & 2 & 3 \\ 3 & 0 & 2 \\ 2 & 2 & 4 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 & 3 \\ 0 & -6 & -7 \\ 0 & -2 & -2 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 & 3 \\ 0 & \textcircled{-6} & -7 \\ 0 & -6 & -6 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 & 3 \\ 0 & -6 & -7 \\ 0 & 0 & 1 \end{pmatrix}$$

Bis hierhin haben wir wie gewohnt gerechnet. Jetzt wandeln wir die (oben orange markierten) Zahlen, die zu groß oder zu klein sind, um, damit sie in den Bereich von 0 bis 4 „passen“:

$$\begin{pmatrix} 1 & 2 & 3 \\ 0 & 4 & 3 \\ 0 & 0 & 1 \end{pmatrix}$$

Das ist ein *anderes* Ergebnis als vorher, aber wir können dieses in das vorherige überführen, indem wir die zweite Zeile (in $\mathbb{Z}_5$) durch 4 dividieren. Beide Antworten sind natürlich richtig. Wir wissen ja schon, das es verschiedene Möglichkeiten gibt, eine Zeilenstufenform zu erreichen.

Lösung 606: Es gibt nur eine Lösung, nämlich $x = 3$, $y = z = 4$. Den Lösungsweg erspare ich mir. Wenn Sie die vorherigen Aufgaben gelöst haben, sollte Ihnen klar sein, was Sie tun müssen.

Lösung 607: Die Lösung sieht so aus:

$$\begin{pmatrix} 2 & 1 & 3 \\ 4 & 1 & 4 \\ 2 & 4 & 3 \end{pmatrix}$$

Lösung 608: Die Determinante von $\boldsymbol{M}$ in $\mathbb{Z}_5$ ist 3.

Lösung 612: Nein. Man bekommt eine Eins, wenn eine *ungerade* Anzahl von Fehlern aufgetreten ist, während eine Null bedeutet, dass eine *gerade* Anzahl von Fehlern aufgetreten ist. (Dazu gehört insbesondere der Fall, dass es *keine* Fehler gab.)

Der ASCII-Code kann, wie auch der ISBN-Code, nicht *alle* Fehler erkennen, sondern nur die, für die er ausgelegt wurde. In diesem Fall ging man davon aus, dass es sehr unwahrscheinlich ist, dass mehr als eins von acht Bits fehlerhaft übertragen wird.

Lösung 613: Im Prinzip keine. Das Prüfbit ist tatsächlich völlig gleichberechtigt. Aber die Bedingung besagt ja auch nur, dass die Gesamtanzahl der Einsen gerade sein muss. Wenn sie das nicht ist, dann ist ein Fehler aufgetreten; und dieser Fehler kann natürlich auch im Prüfbit aufgetreten sein. Das Prüfbit „überwacht“ also sozusagen die sieben Datenbits, aber auch sich selbst.

Lösung 614: Wenn es im roten Kreis einen Fehler gab, dann muss es an einem der Bits p_1, d_1, d_3 oder d_4 gelegen haben. Da es im blauen Kreis auch einen Fehler gab und wir davon ausgehen, dass nur ein Bit fehlerhaft ist, kommen nur noch d_3 und d_4 in Frage, denn nur die beiden liegen in beiden Kreisen. Da es im grünen Kreis aber *keinen* Fehler gab, kann es nicht an d_4 gelegen haben. Wir wissen also mit Sicherheit, dass d_3 das fehlerhafte Bit ist. Gab es nur im roten Kreis einen Fehler, so kann man analog schließen, dass p_1 das problematische Bit war.

Lösung 615: Jede Teilmenge von $\{rot, grün, blau\}$ ist ein solches Muster, also gibt es $2^3 = 8$ Möglichkeiten. Eine von denen entspricht allerdings der leeren Menge. (Das würde bedeuten, dass kein Fehler aufgetreten ist.) Wir haben somit sieben mögliche „Fehlermuster“ und sieben Bits, die jeweils für genau eines dieser Muster verantwortlich sein können. Das passt ja! (Ist natürlich kein Zufall; siehe weiter unten.)

Lösung 618: Natürlich nicht. Wir haben den Code ja gerade so konstruiert, dass jede Spalte von $\boldsymbol{H}$ einer nichtleeren Teilmenge der dreielementigen Menge $\{rot, grün, blau\}$ entspricht. Siehe auch Aufgabe 313.

Lösung 619: Wie die Parity-Check-Matrix aussehen muss, ist auf jeden Fall klar. Man muss einfach die Binärzahlen der Reihe nach als Spalten hinschreiben:

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \qquad \text{(A.8)}$$
$$\begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{matrix}$$

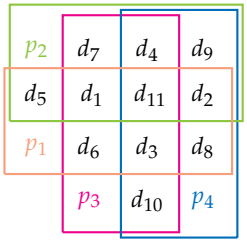
Außerdem wissen wir, dass den Prüfbits gerade die Spalten entsprechen, in denen nur eine Eins steht. Die sind oben blau markiert. Damit ist klar, dass die Datenbits an den Positionen 3, 5, 6 und 7 stehen müssen. Wir können die Generatormatrix also auch

schon zumindest teilweise hinschreiben. (Linke Matrix unten.) Der Rest funktioniert nun wie folgt: Der Matrix (A.8) entnimmt man, dass zum *ersten* Prüfbit die *letzte* Zeile gehört.[64] Dieser Zeile wiederum kann man ablesen, für welche Datenbits das Prüfbit „zuständig“ ist. Das ergibt insgesamt die erste Zeile der Generatormatrix. (Mittlere Matrix.) Uns so geht es dann natürlich weiter. (Rechte Matrix.)

$$\begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ 1 & 0 & 0 & 0 \\ \cdot & \cdot & \cdot & \cdot \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \begin{pmatrix} 1 & 1 & 0 & 1 \\ \cdot & \cdot & \cdot & \cdot \\ 1 & 0 & 0 & 0 \\ \cdot & \cdot & \cdot & \cdot \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Auch hier sollten Sie mit ein paar Beispielen überprüfen, dass die Konstruktion tatsächlich korrekt ist. Der Nachteil dieser Methode ist übrigens, dass man nun nicht mehr sofort die Datenbits erkennen kann. Es sind nicht einfach die ersten vier Bits, sondern sie sind im übertragenen Vektor „versteckt“.

Lösung 620: Zunächst mal das Diagramm. Die spezifischen Positionen wurden dabei eher willkürlich gewählt.



Daraus kann man nun die Generatormatrix ablesen:

$$\left(\begin{array}{cccccccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \hline 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{array}\right)$$

[64] Weil die Eins in der ersten Spalte an der letzten Position steht.

Und das wiederum liefert die Parity-Check-Matrix:

$$\left(\begin{array}{cccccccccc|cccc} 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{array}\right)$$

Lösung 621: Wenn $2+\mathrm{i}$ reell wäre, dann wäre $(2+\mathrm{i})+(-2)$ als Summe von zwei reellen Zahlen ebenfalls reell. Nach den üblichen Rechenregeln ist das aber i, und diese Zahl ist mit Sicherheit *nicht* reell. Ebenso kann 3i nicht reell sein, weil man in diesem Fall mit dem Produkt $(3\mathrm{i}) \cdot 1/3$ argumentieren könnte.

Lösung 622: Sie haben hoffentlich so gerechnet:

$$(5+3\mathrm{i}) - (3+2\mathrm{i}) = (5-3) + (3\mathrm{i} - 2\mathrm{i}) = 2 + (3-2)\mathrm{i} = 2+\mathrm{i}$$

Lösung 625: Meine Lösung sieht so aus:

```
def add (z1, z2):
    return (z1[0] + z2[0], z1[1] + z2[1])

def mult (z1, z2):
    return (z1[0] * z2[0] - z1[1] * z2[1],
            z1[0] * z2[1] + z1[1] * z2[0])
```

Lösung 627: Es besteht keinerlei Gefahr von Verwechslungen, weil die beiden Begriffe übereinstimmen. (Mit anderen Worten: Der Betrag für komplexe Zahlen ist eine Verallgemeinerung, im Sinne von Seite 45, des Betrages für reelle Zahlen.) Es sollte auch geometrisch klar sein, dass bereits der bekannte Betrag für reelle Zahlen den Abstand von der Null angab.

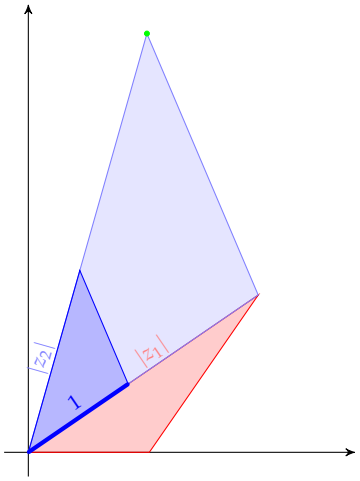
Lösung 628: Man kann das alles ganz einfach direkt nachrechnen:

$$\begin{aligned}
(\overline{a+b\mathrm{i}}) + (\overline{c+d\mathrm{i}}) &= (a-b\mathrm{i}) + (c-d\mathrm{i}) = (a+c) - (b+d)\mathrm{i} \\
\overline{(a+b\mathrm{i}) + (c+d\mathrm{i})} &= \overline{(a+c) + (b+d)\mathrm{i}} = (a+c) - (b+d)\mathrm{i} \\
(\overline{a+b\mathrm{i}}) \cdot (\overline{c+d\mathrm{i}}) &= (a-b\mathrm{i}) \cdot (c-d\mathrm{i}) \\
&= (ac-bd) - (ad+bc)\mathrm{i} \\
\overline{(a+b\mathrm{i}) \cdot (c+d\mathrm{i})} &= \overline{(ac-bd) + (ad+bc)\mathrm{i}} \\
&= (ac-bd) - (ad+bc)\mathrm{i} \\
|wz| &= \sqrt{(wz)(\overline{wz})} = \sqrt{(wz)(\overline{w}\cdot\overline{z})} = \sqrt{w\overline{w}z\overline{z}} \\
|w||z| &= \sqrt{w\overline{w}}\sqrt{z\overline{z}} = \sqrt{w\overline{w}z\overline{z}}
\end{aligned}$$

Lösung 630: Zum Beispiel so:

```
def div (z1, z2):
    divisor = z2[0] * z2[0] + z2[1] * z2[1]
    res = mult(z1, (z2[0], -z2[1]))
    return (res[0] / divisor, res[1] / divisor)
```

Lösung 631: Der für diese Frage relevante Teil der Konstruktion sieht so aus:



Die Seite des blauen Dreiecks, die ursprünglich die Länge 1 hatte, wird auf die Länge $|z_1|$ gestreckt. Das bedeutet nach unseren Vorüberlegungen über ähnliche Dreiecke, dass alle Seitenlängen dieses Dreiecks mit dem Faktor $|z_1|$ multipliziert werden. Die steil nach oben zeigende Seite des blauen Dreiecks hatte aber ursprünglich die Länge $|z_2|$, also hat sie nach der Streckung die Länge $|z_1||z_2| = |z_1 z_2|$.

Lösung 632: In dem rechtwinkligen Dreieck, das das rote Dreieck umschließt, ergibt sich die Winkelsumme von 180 Grad aus dem grünen, dem orangen und dem rechten Winkel. An der rechten Seite des Rechtecks bilden der grüne, der orange und ein rechter Winkel zusammen eine Seite und damit auch 180 Grad. Da es sich beide Male um denselben orangen Winkel handelt, muss auch der grüne Winkel in beiden Fällen gleich groß sein.

Lösung 633: Das geht so:

```
from math import atan2, sqrt

a = 5
b = -8
sqrt(a*a + b*b), atan2(-8, 5)
```

Die Antwort ist also $|z| \approx 9.43$ und $\arg(z) \approx -1.01$.

Alternativ und im Vorgriff auf eine später im Kapitel erwähnte Bibliothek hätte man es auch so machen können:

```
from cmath import phase

z = 5 - 8j
abs(z), phase(z)
```

Oder noch kürzer so:

```
from cmath import polar

polar(5-8j)
```

Lösung 634: Das kann man so ausrechnen:

```
from math import sin, cos

r = 3
phi = 0.5
r * cos(phi), r * sin(phi)
```

z ist also ungefähr $2.63 + 1.44\mathrm{i}$.

Auch hier gibt es eine Abkürzung:

```
from cmath import rect

rect(3, 0.5)
```

Lösung 635: Das erste Quadrat ist $(2\mathrm{i}) \cdot (2\mathrm{i}) = 4\mathrm{i}^2 = -4$, das zweite $(-2\mathrm{i}) \cdot (-2\mathrm{i}) = 4\mathrm{i}^2$. Wegen $(-2) \cdot (-2) = 2 \cdot 2$ sind die Ergebnisse identisch.

Das dritte Quadrat ist $(1 + \mathrm{i}) \cdot (1 + \mathrm{i}) = 1 + 2\mathrm{i} + \mathrm{i}^2 = 2\mathrm{i}$.

Lösung 636: Die erste Gleichung wird durch $5\mathrm{i}$ und $-5\mathrm{i}$ gelöst, die zweite[65] durch $\mathrm{i}\sqrt{3}$ und $-\mathrm{i}\sqrt{3}$.

Lösung 637: Zunächst berechnen wir, weil wir ihn gleich brauchen werden, den folgenden Term:

$$|c|^2 - a^2 = a^2 + b^2 - a^2 = b^2$$

Nun können wir überprüfen, ob x_1 die Gleichung löst:

$$\begin{aligned} x_1^2 &= p^2 - q^2 + 2pq\mathrm{i} \\ &= (|c| + a)/2 - (|c| - a)/2 + 2\mathrm{i}\sqrt{(|c|^2 - a^2)/4} \\ &= a + 2\mathrm{i}\sqrt{b^2/4} = a + |b|\mathrm{i} \end{aligned}$$

[65] Ich habe hier die Reihenfolge der Faktoren absichtlich vertauscht, damit man sieht, dass die Wurzel nur über der Drei steht.

Beachten Sie, dass am Ende $|b|$ dort steht, wo wir gerne b hätten. (Siehe Aufgabe 225.) x_1 ist also nur dann eine Lösung der Gleichung, wenn b nicht negativ ist. Man kann das aber offensichtlich dadurch beheben, dass man x_1 durch $p - q\mathrm{i}$ ersetzt. (Rechnen Sie es nach!)

Lösung 638: Allgemein gilt $|c| = \sqrt{a^2 + b^2} \geq \sqrt{a^2} = |a|$. Daher können, unabhängig vom Vorzeichen von a, sowohl $|c| + a$ als auch $|c| - a$ nicht negativ sein.

Lösung 639: Wenn man den vorgeschlagenen Ansatz ausmultipliziert, dann erhält man:

$$(p^2 - q^2) + 2pq\mathrm{i} = 3 + 4\mathrm{i}$$

Damit hat man zwei Gleichungen für p und q (je eine für den Real- und den Imaginärteil). Die zweite kann man z.B. nach q auflösen und erhält $q = 2/p$. Setzt man das in die erste ein, so wird daraus $p^2 - 4/p^2 = 3$. Multipliziert man mit p^2 und stellt um, so ergibt sich $(p^2)^2 - 3p^2 - 4 = 0$, also eine quadratische Gleichung für p^2 (*nicht* für p!), die man mit Schulwissen auflösen kann: $p^2 = 3/2 \pm 5/2$. Da p reell sein soll, kommt nur $p^2 = 4$ in Frage, also $p = \pm 2$ und damit $q = \pm 1$.

Lösung 640: Wir setzen zur Abkürzung $c = -7/36 - 2/3 \cdot \mathrm{i}$. Dann ist

$$|c| = \sqrt{(7/36)^2 + (2/3)^2} = \sqrt{625/1296} = 25/36$$

und $\Re(c) = -7/36$. Nun berechnen wir:

$$p = \sqrt{(25/36 - 7/36)/2} = \sqrt{9/36} = 3/6 = 1/2$$
$$q = \sqrt{(25/36 + 7/36)/2} = \sqrt{16/36} = 4/6 = 2/3$$

Da $\Im(c) = -2/3$ negativ ist, ist $x_1 = p - q\mathrm{i} = 1/2 - 2/3 \cdot \mathrm{i}$ eine Lösung. Die andere ist $x_2 = -x_1 = -1/2 + 2/3 \cdot \mathrm{i}$. (Verifizieren Sie durch Quadrieren, dass x_1 und x_2 wirklich die Lösungen sind.)

Lösung 641: Wir berechnen zuerst die Polardarstellung von $-15 - 8\mathrm{i}$, dann ziehen wir die Wurzel des Betrages und halbieren den Winkel. Daraus errechnen wir die kartesische Darstellung einer Lösung x_1.

```
r = sqrt(15 * 15 + 8 * 8)
phi = atan2(-8, -15)
sqrt(r) * cos(phi / 2), sqrt(r) * sin(phi / 2)
```

Wir wussten bereits, dass $-1 + 4\mathrm{i}$ oder $1 - 4\mathrm{i}$ herauskommen musste. Und bis auf Rundungsfehler stimmt das auch. x_2 erhalten wir nun wie üblich als $-x_1$. Oder in der Polardarstellung, indem wir zum Argument der ersten Lösung $180°$ addieren.

Lösung 642: Ist $a \neq 0$, so kann man durch a teilen und erhält die Gleichung

$$x^2 + b/a \cdot x + c/a = 0$$

vom Typ (35.3), bei der b/a und c/a die Rollen von p und q spielen. Ist $a = 0$, so hat man es mit der simplen Gleichung $bx + c = 0$ zu tun, die Sie hoffentlich lösen können.

Lösung 643: Wir formen um und ergänzen:

$$x^2 - 4x = -13$$
$$(x-2)^2 = -13 + 2^2 = -9$$

Nun substituieren wir w für $x-2$ und lösen die Gleichung $w^2 = -9$. Das führt zu $w_1 = 3\mathrm{i}$ und $w_2 = -3\mathrm{i}$. Rücksubstitution liefert dann $x_1 = w_1 + 2 = 2 + 3\mathrm{i}$ und $x_2 = w_2 + 2 = 2 - 3\mathrm{i}$.

Lösung 645: In der allgemeinsten Form würde es so aussehen: $ax^3 + bx^2 + cx + d = 0$. Das ist aber kein relevanter Unterschied. Entweder ist $a \neq 0$ und man kann die ganze Gleichung durch a dividieren; dann sieht sie aus wie (35.4). Oder es gilt $a = 0$ und man hat es dann einfach mit einer (höchstens) quadratischen Gleichung zu tun.

Lösung 646: Mit SymPy würde es so gehen:

```
def f (x):
    return x ** 3 + a * x ** 2 + b * x + c

from sympy import *

a, b, c, y = symbols("a b c y")
expand(f(y - a / 3))
```

`expand`

`expand` macht in der einfachsten Form das, was man üblicherweise mit Ausmultiplizieren bezeichnet. Das Ergebnis sieht in Python nicht ganz so schön aus, entspricht aber diesem Ausdruck:

$$y^3 + (-1/3 \cdot a^2 + b)y + 2/27 \cdot a^3 - 1/3 \cdot ab + c$$

Entscheidend ist, dass dieser Ausdruck einfacher als (35.4) ist, weil y^2 nicht mehr vorkommt. Die farbig markierten Teile hängen nur von den Koeffizienten ab und werden abkürzend im Folgenden mit p und q bezeichnet.

Lösung 647: Das ist etwas mühsam, aber eine gute Übung für den Umgang mit Wurzeln.

Zur Abkürzung setzen wir

$$D = \sqrt{(q/2)^2 - (p/3)^3}$$
$$E_+ = \sqrt[3]{q/2 + D}$$
$$E_- = \sqrt[3]{q/2 - D}$$

Mithilfe der dritten binomischen Formel ergibt sich:

$$\begin{aligned}(q/2 + D) \cdot (q/2 - D) &= (q/2)^2 - D^2 \\ &= (q/2)^2 - \left(\sqrt{(q/2)^2 - (p/3)^3}\right)^2 \\ &= (q/2)^2 - \left((q/2)^2 - (p/3)^3\right) = (p/3)^3\end{aligned}$$

Noch ein bisschen rechnen, dann hat man die gewünschte Identität:

$$\begin{aligned}
E_+^3 &= \left(\sqrt[3]{q/2+D}\right)^3 = q/2+D \\
E_-^3 &= \left(\sqrt[3]{q/2-D}\right)^3 = q/2-D \\
E_+^3 + E_-^3 &= q/2+D+q/2-D = q \\
E_+^2 \cdot E_- &= \left(\sqrt[3]{q/2+D}\right)^2 \cdot \left(\sqrt[3]{q/2-D}\right) \\
&= \left(\sqrt[3]{q/2+D}\right) \cdot \left(\sqrt[3]{(q/2+D)(q/2-D)}\right) \\
&= E_+ \cdot \sqrt[3]{(p/3)^3} = p/3 \cdot E_+ \\
E_+ \cdot E_-^2 &= \left(\sqrt[3]{q/2+D}\right) \cdot \left(\sqrt[3]{q/2-D}\right)^2 \\
&= \left(\sqrt[3]{(q/2+D)(q/2-D)}\right) \cdot \left(\sqrt[3]{q/2-D}\right) \\
&= \sqrt[3]{(p/3)^3} \cdot E_- = p/3 \cdot E_-
\end{aligned}$$

$$\begin{aligned}
y^3 &= (E_+ + E_-)^3 = E_+^3 + 3E_+^2 E_- + 3E_+ E_-^2 + E_-^3 \\
&= 3 \cdot p/3 \cdot E_+ + 3 \cdot p/3 \cdot E_- + q = p(E_+ + E_-) + q = py + q
\end{aligned}$$

Lösung 648: Wir dividieren durch 3 und erhalten $x^3 - 7x^2 + 16x - 12 = 0$. Dann setzen wir für x den Wert $y + 7/3$ ein und erhalten durch Ausmultiplizieren $y^3 = y/3 + 2/27$. Damit haben wir $p = 1/3$ und $q = 2/27$. Es ergibt sich:

$$(q/2)^2 - (p/3)^3 = (1/27)^2 - (1/9)^3 = (1/3)^6 - (1/3)^6 = 0$$

(35.6) vereinfacht sich somit zu $y = \sqrt[3]{q/2} + \sqrt[3]{q/2} = 2 \cdot \sqrt[3]{1/27} = 2/3$. Wegen unserer Ersetzung $x = y + 7/3$ ergibt sich $x = 2/3 + 7/3 = 3$. Verifizieren Sie durch Einsetzen in die ursprüngliche Gleichung, dass 3 wirklich eine Lösung ist.

Lösung 649: Mit dem vorgeschlagenen Ansatz erhält man:

$$3 = \sqrt[3]{\sqrt{325}+18} - \sqrt[3]{\sqrt{325}-18} = (a+b) - (a-b) = 2b$$

Daraus folgt $b = 3/2$. Nun können wir so fortfahren:

$$\begin{aligned}
\sqrt{325} + 18 &= (a+b)^3 = a^3 + 3a^2 b + 3ab^2 + b^3 \\
\sqrt{325} - 18 &= (a-b)^3 = a^3 - 3a^2 b + 3ab^2 - b^3
\end{aligned}$$

Subtrahiert man diese beiden Gleichungen, so ergibt sich:

$$36 = 6a^2 b + 2b^3 = 9a^2 + 27/4$$

Das ist eine quadratische Gleichung für a, die zwei Lösungen hat. Durch Einsetzen verifiziert man, dass $a = \sqrt{13}/2$ der gesuchte Wert ist, d.h. man rechnet nach: $(\sqrt{13}/2 \pm 3/2)^3 = \sqrt{325} \pm 18$.

Lösung 650: $z \mapsto \Re(z^2-1)$ bildet die komplexe Zahl $z = x+y\mathrm{i}$ auf die reelle Zahl x^2-y^2-1 ab. (Rechnen Sie nach!) Die zu zeichnende Funktion wäre also diese:

$$: \begin{cases} \mathbb{R}^2 \to \mathbb{R} \\ (x,y) \mapsto x^2-y^2-1 \end{cases}$$

Lösung 651: Mit $z = x+y\mathrm{i}$ erhält man

$$\begin{aligned} \left|z^2-3/10\right| &= \left|x^2-y^2+2xy\mathrm{i}-3/10\right| \\ &= \sqrt{(x^2-y^2-3/10)^2+(2xy)^2} \\ &= \sqrt{x^4+y^4+2x^2y^2+9/100-3/5\cdot x^2+3/5\cdot y^2} \end{aligned}$$

und ebenso

$$\left|z^2+3/10\right| = \sqrt{x^4+y^4+2x^2y^2+9/100+3/5\cdot x^2-3/5\cdot y^2}$$

Vertauscht man im ersten Term x und y, erhält man den zweiten. Und das gilt auch umgekehrt.

Lösung 652: Wir wissen schon, dass eine Multiplikation mit i einer Drehung um 90° entspricht. Dreht man umgekehrt die komplexe Zahl $x+\mathrm{i}y$ um $-90°$, so erhält man $(x+\mathrm{i}y)\cdot(-\mathrm{i}) = y-\mathrm{i}x$. Nun muss man nur noch einsetzen:

$$\begin{aligned} |f(x+\mathrm{i}y)| &= 1/|(x+\mathrm{i}y)^2-1| = 1/|x^2-y^2-1+2xy\mathrm{i}| \\ &= 1/\sqrt{(x^2-y^2-1)^2+(2xy)^2} \\ |g(y-\mathrm{i}x)| &= 1/|(y-\mathrm{i}x)^2+1| = 1/|y^2-x^2+1-2xy\mathrm{i}| \\ &= 1/\sqrt{(y^2-x^2+1)^2+(2xy)^2} \end{aligned}$$

x^2-y^2-1 und y^2-x^2+1 unterscheiden sich nur durch das Vorzeichen, was nach dem Quadrieren keine Rolle mehr spielt. Also gilt $|f(x+\mathrm{i}y)| = |g(y-\mathrm{i}x)|$.

Lösung 655: In Kurzschreibweise ist die erste Folge $(3^n)_{n\in\mathbb{N}}$ und die zweite Folge ist $((-1/2)^n)_{n\in\mathbb{N}}$. Beachten Sie das Vorzeichen![66] Die dritte Folge könnte man z.B. als $(\sum_{k=1}^{n} k)_{n\in\mathbb{N}^+}$ schreiben. Mithilfe der Gaußschen Summenformel geht das aber einfacher als $(n(n+1)/2)_{n\in\mathbb{N}^+}$.

In Python könnte es so aussehen:

```
from fractions import Fraction

def seq1 ():
    val = 1
    while True:
        yield val
        val *= 3
```

[66] Man spricht von einer *alternierenden Folge*, wenn in jedem Schritt das Vorzeichen wechselt.

```
def seq2 ():
    val = Fraction(1)
    while True:
        yield val
        val *= -Fraction(1,2)

def seq3 ():
    n = 1
    val = n
    while True:
        yield val
        n += 1
        val += n
```

Man könnte aber auch streng nach der Definition des Begriffs *Folge* vorgehen und diese als Funktionen darstellen, deren Definitionsbereich $\mathbb{N}$ (bzw. $\mathbb{N}^+$) ist:

```
def seq1 (n):
    return 3 ** n

def seq2 (n):
    return (-Fraction(1,2)) ** n

def seq3 (n):
    return n * (n + 1) // 2
```

Lösung 656: Bei der ersten Folge sind die Folgenglieder 10^9, 10^8, 10^7, etc. Am Anfang sind das sehr große Werte, aber das zehnte Folgenglied ist $10^0 = 1$ und danach sind alle weiteren Werte kleiner als 1. Also sind fast alle Folgenglieder (nämlich alle bis auf zehn) kleiner als 1.

Bei der zweiten Folge haben natürlich zunächst sehr viele Folgenglieder *nicht* eine Million verschiedene Primteiler. Aber: $3! = 1 \cdot 2 \cdot 3$ hat zwei Primteiler, nämlich 2 und 3. 5! hat drei Primteiler, nämlich 2, 3 und 5. Und allgemein hat $p_k!$ offensichtlich k verschiedene Primteiler, wenn p_k die k-te Primzahl ist. Wir müssen also „nur“ die millionste Primzahl finden. Deren Fakultät hat eine Million verschiedene Primteiler und alle Fakultäten danach „erst recht“. Die gesuchte Zahl ist übrigens 15 485 863.[67] Mehr als fünfzehn Millionen Folgenglieder haben also weniger als eine Million Primteiler. Trotzdem haben aber fast alle Folgenglieder mehr als eine Million Primteiler, weil es „nur“ endlich viele Ausnahmen gibt.

[67] Die Fakultät dieser Zahl hat mehr als hundert Millionen Dezimalstellen. Zum Vergleich: Der Teil dieses Buches, den Sie bisher gelesen haben, enthält weniger als 1 000 000 Buchstaben. Das Abdrucken der eben genannten Zahl würde mehr als die hundertfache Menge an Buchseiten benötigen!

Schließlich sind zwar *unendlich* viele natürliche Zahlen gerade, aber *nicht fast alle*, denn es gibt auch unendlich viele natürlich Zahlen, die *nicht* gerade sind.

Lösung 657: Die Folge konvergiert *nicht*. Als Grenzwerte kämen offensichtlich nur 1 und -1 in Frage. Wenn man aber z.B. das Intervall $(0.9, 1.1)$ um 1 herum betrachtet, dann liegen unendlich viele Folgenglieder (nämlich jedes zweite) nicht in diesem Intervall, also konvergiert die Folge nicht gegen 1. Analog kann man begründen, dass die Folge nicht gegen -1 konvergiert.

Lösung 658: So sollte man es sofort sehen:

$$\frac{n}{n+1} = \frac{(n+1)-1}{n+1} = 1 - \frac{1}{n+1}$$

Von 1 wird ein Wert abgezogen, der sukzessive immer kleiner wird.

Lösung 659: Ich habe es so gemacht:

```
import math

someValues(lambda n: ((-1) ** n) / n,
           [1, 2, 3, 10, 99, 100, 999, 1000])
someValues(lambda n: (2 / 3) ** n)
someValues(lambda n: (3 / 2) ** n, [1, 10, 100, 1000])
someValues(lambda n: 42 ** (1 / n))
someValues(lambda n: n ** (1 / n))
someValues(lambda n: (5 ** n) / math.factorial(n),
           [1, 2, 10, 20, 100])
someValues(lambda n: (n ** 3) / (2 ** n))
someValues(lambda n: ((n + 1) / n) ** n)
```

Bei der dritten und der sechsten Folge habe ich nicht den Standardwert für `L` genommen, da man sonst eine Fehlermeldung wegen zu großer Zahlen bekommen hätte. Man sieht aber evtl. trotzdem, wie sich diese Folgen für große n verhalten.

Bei der ersten Folge ist insbesondere zu sehen, dass eine konvergente Folge sich ihrem Grenzwert nicht unbedingt von einer Seite aus annähern muss. Die Folge alterniert, d.h. sie springt ständig vom positiven in den negativen Bereich und umgekehrt. Aber unabhängig davon kommen die Folgenglieder der Null trotzdem immer näher.

Lösung 660: Gilt $|q| = 1$ und ist q reell, so ist $q = 1$ oder $q = -1$. Im ersten Fall ist die geometrische Folge (q^n) die Folge, die konstant den Wert 1 hat. Im zweiten Fall ist (q^n) eine Folge, die abwechselnd die Werte 1 und -1 annimmt.

Lösung 661: Mit $z = 1$ bekommen Sie $1 + 1/n = (n+1)/n$.

Lösung 662: Wenn Sie das Geld einfach ein Jahr auf dem Konto lassen, haben Sie am Ende des Jahres $1.06 \cdot 42\,000 = 44\,520$ Euro. Wir schreiben das aber etwas anders, damit man besser sehen kann, was passiert:

$$(1 + 0.06) \cdot 42\,000$$

Gehen Sie schon nach sechs Monaten hin und zahlen das Geld gleich wieder ein, so bekommen Sie $1.03 \cdot 1.03 \cdot 42\,000 = 44\,557.80$ Euro.[68] Wir schreiben das so hin:

$$(1 + 0.06/2) \cdot (1 + 0.06/2) \cdot 42\,000$$

Jetzt kann man langsam erkennen, wie es weitergehen wird. Kommt man alle vier Monate (teilt man also das Jahr in drei Teile auf), so ergibt sich dies:

$$(1 + 0.06/3) \cdot (1 + 0.06/3) \cdot (1 + 0.06/3) \cdot 42\,000$$

Und teilt man das Jahr in n Teile auf, so hat man am Ende diesen Betrag erspart:

$$(1 + 0.06/n)^n \cdot 42\,000$$

Lässt man nun n immer größer werden, so kommt man dem Grenzwert $e^{0.06} \cdot 42\,000$ immer näher. Der Faktor vor dem Startkapital ist auf fünf Stellen nach dem Komma gerundet 1.06184. Sie können also so oft zur Bank gehen, wie Sie wollen, den Zinssatz werden Sie dadurch nicht höher als etwa 6.18 Prozent treiben.

Diese unrealistische Aufgabe hat übrigens einen durchaus realistischen Hintergrund: Die Eulersche Zahl taucht immer gerne auf, wenn es um Wachstumsprozesse geht.

Lösung 663: Der Bruch lässt sich folgendermaßen umstellen:

$$\frac{n^{20} \cdot 100000^n}{100001^n} = \frac{n^{20}}{(100001/100000)^n}$$

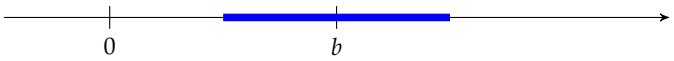
Nun sieht man, dass man es mit einer Folge der Form (n^p/b^n) zu tun hat, die gegen null konvergiert, weil $100001/100000$ (ein bisschen) größer als eins ist.

Lösung 664: Die erste Regel folgt aus der für das Produkt $(a_n b_n)$, wenn man für (b_n) die konstante Folge (λ) wählt.

Lösung 665: Rein formal darf man das tatsächlich nicht hinschreiben. Allerdings wurde in der Rechenregel vorausgesetzt, dass die Folge (b_n) gegen einen von null verschiedenen Wert b konvergiert. Wenn man nun um diesen Wert z.B. das Intervall

$$(b - b/2, b + b/2) = (b/2, 3b/2)$$

herumlegt, dann müssen fast alle Folgenglieder in diesem Intervall liegen und können darum nicht null sein.



Die Folge (b_n) hat also höchstens endlich viele Folgenglieder, die null sind und diese „wenigen“ werden einfach stillschweigend ignoriert.

Lösung 667: Die Umformung $\sqrt[n]{n^2} = \sqrt[n]{n} \cdot \sqrt[n]{n}$ zeigt, dass die Folge den Grenzwert $1 \cdot 1 = 1$ hat.

[68] Das sind tatsächlich fast 40 Euro mehr. Klingt gut, aber dadurch hat sich aufs ganze Jahr gerechnet der Zins lediglich von 6 auf 6.09 Prozent erhöht.

Lösung 668: Die Umformung

$$\left(\frac{n+1}{n\cdot\sqrt[n]{\mathrm{e}}}\right)^n = \left(\frac{n+1}{n}\right)^n \cdot \left(\frac{1}{\sqrt[n]{\mathrm{e}}}\right)^n = \left(\frac{n+1}{n}\right)^n \cdot \frac{1}{\mathrm{e}}$$

zeigt, dass der Grenzwert $\mathrm{e}\cdot 1/\mathrm{e} = 1$ ist.

Lösung 669: Bei der ersten Folge wurden die Potenzen im Nenner absichtlich unsortiert aufgeschrieben. Wenn man genau hinschaut, sieht man, dass in Zähler und Nenner die höchste Potenz n^4 ist. Der Grenzwert ist also $7/(1/6) = 42$.

Die zweite Folge konvergiert nicht, weil der Grad des Zählers (4) höher als der Grad des Nenners (3) ist.

Lösung 670: Wir können umformen, indem wir einfach ausmultiplizieren:

$$\frac{n(n-1)}{(n+2)(n+3)} = \frac{n^2-n}{n^2+5n+6}$$

Dann sieht man sofort, dass man es mit einem Quotienten zweier Polynome zu tun hat und dass der Grenzwert eins ist.

Lösung 671: Die Umkehrung gilt nicht. Ein ganz einfaches Gegenbeispiel ist die zweite Folge aus Aufgabe 655. Es handelt sich sicherlich um eine Nullfolge. Da die Folgenglieder ständig das Vorzeichen wechseln, gilt das auch für die Kehrwerte. Wenn man also die Folge der Kehrwerte bildet, so wird diese Folge zwar divergieren, aber *nicht* bestimmt, weil z.B. weder fast alle Folgenglieder größer als 42 sind noch fast alle Folgenglieder kleiner als -42.

Lösung 672: Offensichtlich werden hier alle Brüche aufgezählt, bei denen Zähler und Nenner positiv sind und der Zähler kleiner als der Nenner ist. Und zwar sortiert nach den Nennern und bei gleichem Nenner nach den Zählern. Alle Folgenglieder liegen somit zwischen 0 und 1.

Diese Folge hat die interessante Eigenschaft, dass jede rationale Zahl im Intervall $(0,1)$ unendlich oft „besucht“ wird. Nehmen wir als Beispiel den Bruch $4/7$. Der kommt zum ersten Mal vor bei den Brüchen, deren Nenner 7 ist. Er kommt aber später erneut als $8/14$ vor. Und dann noch mal als $12/21$ und immer so weiter.

Darum konvergiert diese Folge auch nicht. Wo auch immer wir ein Intervall hinlegen, das schmal genug ist, befinden sich außerhalb dieses Intervalls Brüche, die unendlich oft in der Folge vorkommen. Also können niemals fast alle Folgenglieder innerhalb des Intervalls liegen.

Diese Folge ist also divergent, aber weder ist sie bestimmt divergent, noch springt sie einfach nur immer zwischen zwei (oder drei oder sieben) Werten hin und her.

(Man kann aber sogar noch „amüsantere“ Folgen basteln. Z.B. solche, bei denen kein Wert jemals wiederholt wird, man aber trotzdem jedem Punkt in einem Intervall unendlich oft beliebig nahe kommt…)

Lösung 673: Man kann es z.B. so machen:

```
import math
from fractions import Fraction

def strangeSeq (n):
    k = int(math.sqrt(2 * n))
    if 2 * n <= k * (k + 1):
        d = k
    else:
        d = k + 1
    return Fraction(n - d * (d - 1) // 2, d + 1)
```

Zum Verständnis dieser Funktion erinnern Sie sich bitte an die Funktion `strange` von Seite 211.

Lösung 674: Nur der erste. Für alle anderen hätte man zur Darstellung im Dezimalsystem unendlich viele Nachkommastellen benötigt.[69] Und das gilt übrigens auch für die Darstellung im Binärsystem, d.h. auch intern wurden alle bis auf ein Folgenglied nicht ganz korrekt dargestellt.

Lösung 675: Im ersten Fall wurden Zähler und Nenner zunächst exakt in Langzahlarithmetik (siehe Seite 40) berechnet, bevor dividiert wurde. Das lag daran, dass alle beteiligten Operanden ganze Zahlen waren. Im zweiten Fall haben wir Python durch die Werte 20.0 und 1.00001 gezwungen, mit Fließkommazahlen zu rechnen. Das macht das Ergebnis zwar – wie wir schon wissen – potentiell ungenauer, es geht aber wesentlich schneller. Im Kapitel 39 werden wir uns damit beschäftigen, wie man vorher abschätzen kann, wie lange der Computer für eine bestimmte Berechnung braucht.

Lösung 676: Die Folge (n^3/b^n) konvergiert nur dann, wenn $|b|$ größer als eins ist (siehe Seite 442), daher wäre es falsch, wenn SymPy hier einfach mit 0 antworten würde.

Lösung 677: Damit Sie keine Fehlermeldung bekommen, müssen Sie die konstante Folge (42) so eingeben:

`Integer`

```
limit(Integer(42), n, oo)
```

Das ist notwendig, damit Python die Eingabe `42` als einen *Ausdruck* im Sinne von SymPy und nicht einfach als Zahl interpretiert. Bei zusammengesetzten Ausdrücken wie `42/n` ist das normalerweise nicht nötig; nur dann, wenn das Argument ausschließlich aus Zahlen besteht.

Lösung 678: Man kann es mit SymPy machen:

```
limit((4 ** (n + 1) + 2 ** n) / (4 ** n + 2 ** (2 * n)), n, oo)
```

[69] Siehe Aufgabe 171.

Man hätte es aber auch ohne Computerhilfe herausbekommen können. Dazu wenden wir den „Trick“ an, den wir schon bei Polynomen verwendet haben: wir dividieren Zähler und Nenner durch den Term, der am schnellsten wächst:

$$\frac{4^{n+1}+2^n}{4^n+2^{2n}} = \frac{4\cdot 4^n+2^n}{4^n+4^n} = \frac{\frac{1}{4^n}\cdot(4\cdot 4^n+2^n)}{\frac{1}{4^n}\cdot(2\cdot 4^n)} = \frac{4+(\frac{1}{2})^n}{2}$$

Da die geometrische Folge $((1/2)^n)$ gegen null konvergiert, kommt insgesamt offensichtlich als Grenzwert $(4+0)/2 = 2$ heraus.

Lösung 679: Wir haben $w - z = -5 - \mathrm{i}$ und damit $|w-z| = \sqrt{5^2+1^2} = \sqrt{26}$.

Lösung 680: Einerseits gilt:

$$|w-z| = |(a-c)+(b-d)\mathrm{i}| = \sqrt{(a-c)^2+(b-d)^2}$$

Andererseits ergibt sich (siehe Seite 359):

$$\mathrm{d}(\boldsymbol{w},\boldsymbol{z}) = \|\boldsymbol{w}-\boldsymbol{z}\| = \|(a-c,b-d)^T\| = \sqrt{(a-c)^2+(b-d)^2}$$

In beiden Fällen kommt also dasselbe heraus.

Lösung 681: Der Grenzwert ist 2. (Die Folge der Realteile ist konstant; die Folge der Imaginärteile ist die harmonische Folge, die eine Nullfolge ist.) Beachten Sie, dass alle Folgenglieder echt komplex sind, während der Grenzwert reell ist.

Lösung 682: Nein. Wenn man bei $n = 0$ startet, nimmt die Folge die Werte 1, i, -1 und $-\mathrm{i}$ an und wiederholt sich von da an zyklisch. Sie „tanzt“ sozusagen um den Nullpunkt herum.

Lösung 683: Wie bei der vorherigen Aufgabe „tanzen“ die Folgenglieder um den Nullpunkt. Gleichzeitig wird ihr Abstand zur Null aber immer geringer. Die Folge konvergiert also gegen null.

Lösung 684: Die ersten drei Folgenglieder:

$$\begin{aligned} z_0 &= 42\mathrm{i} + (4+2\mathrm{i})^0/0! = 1+42\mathrm{i} \\ z_1 &= 42\mathrm{i} + (4+2\mathrm{i})^1/1! = 4+44\mathrm{i} \\ z_2 &= 42\mathrm{i} + (4+2\mathrm{i})^2/2! = 42\mathrm{i}+6+8\mathrm{i} = 6+50\mathrm{i} \end{aligned}$$

Der Abstand aus dem Hinweis ist:

$$|z_n - 42\mathrm{i}| = \left|\frac{(4+2\mathrm{i})^n}{n!}\right| = \frac{|4+2\mathrm{i}|^n}{n!} = \frac{\sqrt{20}^n}{n!}$$

Unabhängig vom konkreten Wert von $\sqrt{20}$ wissen wir bereits, dass dieser Term gegen 0 geht, wenn n immer größer wird. Daher konvergiert die Folge gegen 42i.

Lösung 685: Man argumentiert so: Aus $(1+x)^2 \geq 1+2x$ folgt

$$(1+x)^2(1+x) \geq (1+2x)(1+x),$$

indem man beide Seiten mit $1+x$ multipliziert. Das klappt aber nur deshalb, weil dieser Faktor nicht negativ ist.[70] Und das wiederum stimmt nur, weil x nicht kleiner als -1 sein darf.

Lösung 687: Ja. Die Rolle des x spielt dabei der Term $(1-|q|)/|q| = 1/|q| - 1$. Wir dürfen die Ungleichung anwenden, wenn x nicht kleiner als -1 ist. Das ist aber der Fall, da $1/|q|$ (wegen $|q| < 1$) sogar größer als 1 ist.

Lösung 689: Das Zeichen zwischen den Termen wird „umgedreht“. Konkret gilt z.B. $23 \leq 42$. Für die Kehrwerte gilt aber $1/23 \geq 1/42$.

Lösung 690: Man muss lediglich in (17.6) das a durch 1 ersetzen, wodurch der Ausdruck einfacher wird:

$$(1+b)^n = \sum_{k=0}^{n} \binom{n}{k} b^{n-k} = \sum_{k=0}^{n} \binom{n}{k} b^k$$

Die letzte Umformung ist erlaubt, weil der Binomialkoeffizient symmetrisch ist.

Lösung 693: Mit der geometrischen Summenformel ergibt sich:

$$\sum_{k=0}^{n} 1/2^k = \frac{1/2^{n+1} - 1}{1/2 - 1} = (-2) \cdot (1/2^{n+1} - 1) = 2 - 1/2^n$$

Lösung 695: Das Ergebnis bei `aop1(20, 1000)` ist deutlich größer als bei `aop1(2, 1000)` und das gilt natürlich auch für die Zwischenergebnisse. Python wird in beiden Fällen Langzahlarithmetik (siehe Kapitel 3) verwenden. Und bei der Langzahlarithmetik hängt die Ausführungsgeschwindigkeit von simplen arithmetischen Operationen wie `+` oder `*` von der Größe der Operanden ab.

Lösung 696: Man könnte die Formel für die geometrische Summe verwenden. Wenn man jedoch „fair“ sein will, dann darf man auch in diesem Fall nicht einfach potenzieren (also `**` verwenden). Das sähe dann so aus:

```
def aop3 (x, n):
    p = 1
    for i in range(n + 1):
        p *= x
    return (1 - p) / (1 - x)
```

Mit der Terminologie, die wir in diesem Kapitel entwickeln werden, wird sich herausstellen, dass `aop3` zwar schneller als `aop2` ist, aber nicht *wesentlich* schneller.

Lösung 698: Das würde so aussehen:

```
def aop2 (x, n):
    s = 1                       # 1
    for i in range(n):          #   n: 2
        s *= x                  #      2
```

[70] Aus $\geq$ würde $\leq$ werden, wenn wir mit einem negativen Faktor multiplizieren würden.

```
        s += 1                  #       2
    return s                    # 1
```

Und das Ergebnis wäre $6n + 2$.

Lösung 699: Pro Durchlauf werden drei Zeiteinheiten benötigt, eine für den Test `i < n`, und zwei für `i += 1`, eine Kombination aus Addition und Zuweisung. Das sind $3n$ Zeiteinheiten. Hinzu kommen jedoch noch eine Zeiteinheit für `i = 0` sowie eine weitere Zeiteinheit für den *letzten* Test `i < n`, der durchgeführt wird, bevor die Schleife beendet wird. Das ergibt insgesamt $3n + 2$ Zeiteinheiten. Für eine vergleichbare `for`-Schleife berechnen wir in unserem Modell nur $2n$ Einheiten. Es wird sich aber bald herausstellen, dass solche Unterschiede mehr oder weniger irrelevant sind.

Beachten Sie, dass wir keine Zeit dafür berechnen, dass der Code nach jedem Durchlauf wieder zum Schleifenanfang „zurückspringt“. Auch hier kann man darüber streiten, ob das realistisch ist.

Lösung 700: Ohne Anspruch auf Vollständigkeit hier ein paar Gründe:

- Warum sollte z.B. eine Addition genauso viel Zeit verbrauchen wie eine Multiplikation? In manchen Prozessoren wird die Addition schneller sein. Siehe aber auch die Lösung zu Aufgabe 715.
- Außerdem ist es nicht unrealistisch, dass arithmetische Operationen mit Fließkommazahlen langsamer sind als dieselben mit ganzen Zahlen.
- Schließlich kann die Geschwindigkeit von arithmetischen Operationen auch von der Größe (also der benötigten Wortbreite) der Operanden abhängen. Selbst dann, wenn Langzahlarithmetik (siehe Aufgabe 695) keine Rolle spielt.
- Warum sollte ein Ausdruck wie `a * b` dieselbe Zeit benötigen wie `2 * b`? Im ersten Fall müssen ggf. beide Operanden aus dem Speicher geholt werden (was Zeit verbraucht), im zweiten nur einer.
- Warum sollte eine Zuweisung dieselbe Zeit benötigen wie eine arithmetische Operation?
- Wie misst man überhaupt die für eine Zuweisung benötigte Zeit? Das ist eine sehr komplexe Frage, weil die entsprechenden Werte sich sowohl in *Registern* der CPU als auch im *Hauptspeicher* befinden können. Zudem spielen beim Zugriff auf den Hauptspeicher die verschiedenen *Caches* des Rechners eine erhebliche Rolle.
- Ein Befehl wie `return` führt im Hintergrund diverse „Verwaltungsaufgaben“ durch und ist daher ggf. aufwendiger als etwa eine einfache Zuweisung. Das gilt natürlich ebenso für den Aufruf einer Funktion.
- Dass unser Berechnungsansatz für die `for`-Schleife nicht exakt richtig sein kann, deutet Aufgabe 699 an. Außerdem ist es natürlich sicher nicht korrekt, wenn man so tut, als wäre der Aufruf der Funktion `range` „kostenlos“.

- Selbst wenn Sie den Maschinencode eines Programms Schritt für Schritt analysieren, können Sie nicht jedem einzelnen Befehl eindeutig eine bestimmte Zeitdauer zuordnen, weil moderne Prozessoren den Code ggf. dadurch optimieren, dass sie die Befehle in einer anderen Reihenfolge abarbeiten.[71]
- Aktuelle Prozessoren (insbesondere solche, die in mobilen Geräten wie Laptops eingesetzt werden), können ihre Taktfrequenz in Abhängigkeit von Faktoren wie Belastung, Temperatur und Energieversorgung variieren. Dadurch wird es natürlich fast unmöglich, von festen „Zeiteinheiten“ zu reden.
- Für den speziellen Fall eines Python-Programms haben wir uns zudem keine Gedanken darüber gemacht, wie viel Zeit der Interpreter verbraucht, während die Funktion läuft. Grundsätzlich unterscheidet sich die Analyse des Laufzeitverhaltens von interpretierten Programmen von der von kompilierten.

Sie sehen: Ohne Abstraktion ist es selbst für die einfachsten Algorithmen faktisch unmöglich, ihre exakte Laufzeit analytisch zu bestimmen.

Lösung 701: Das Intervall $[-7,4]$ ist beschränkt. Als Schranke eignet sich z.B. 7. Die Menge $\mathbb{N}$ ist nicht beschränkt: Wenn C irgendeine (positive) reelle Zahl ist, dann ist $\lceil C \rceil + 1$ eine natürliche Zahl, die noch größer ist; also kann C keine Schranke sein. Die Folge $(1/n)$ ist beschränkt und $((-1)^n)$ auch. Für beide eignet sich 1 als Schranke.

Lösung 702: Wenn man es mit einer endlichen Menge A zu tun hat, dann kann man als Schranke einfach das größte Element der Menge $\{|a| : a \in A\}$ der Beträge nehmen. Im Beispiel vor Aufgabe 701 hätte man als Schranke also auch 42 wählen können.

Beachten Sie, dass unendliche Mengen ggf. kein größtes Element haben. Z.B. ist das Intervall $[-1,2)$ beschränkt, hat aber kein größtes Element.

Lösung 703: Wenn C eine Schranke der Menge $A \subseteq \mathbb{R}$ ist, dann liegen alle Elemente von A innerhalb des Intervalls $[-C,C]$.

Lösung 704: Es handelte sich um die Folge $((-1)^n)$ aus Aufgabe 701.

Lösung 705: Ja, das begründet man genau wie eben mit $2/3$ statt $3/2$. Interpretation: Die Folge $(3n)$ wächst nicht wesentlich schneller als $(2n)$, aber umgekehrt wächst $(2n)$ auch nicht wesentlich schneller als $(3n)$.

Lösung 706: Es ist genauso gemeint wie in der Mengenlehre. Mit anderen Worten: $\mathcal{O}(g(n))$ ist eine Menge; und zwar die Menge aller Funktionen, die von der Ordnung $g(n)$ sind.

Lösung 707: Die Funktion `p1` führt neun arithmetische Operationen und eine `return`-Anweisung aus, braucht also zehn Zeiteinheiten. Bei `p2` kommt man analog auf sieben Zeiteinheiten. Nun zu `test1`:

[71] Bei Interesse schlagen Sie mal unter *out-of-order execution* und *Superskalarität* nach.

```
def test1 (n):
    step = 2 / (n - 1)          # 3
    x = -1                      # 1
    for i in range(n):          #   n:  2
        y = p1(x)               #      11
        x += step               #       2
```

Das ergibt insgesamt eine Laufzeit von $f(n) = 15n + 4$ Zeiteinheiten. Analog ergibt sich für `test2` als Laufzeit $g(n) = 12n + 4$. Sowohl $(f(n)/g(n))$ als auch $(g(n)/f(n))$ konvergieren und sind daher beschränkt. Daher gilt $f(n) \in \mathcal{O}(g(n))$, aber es gilt ebenso $g(n) \in \mathcal{O}(f(n))$.

Nebenbei bemerkt konvergiert $(g(n)/f(n))$ offenbar gegen $12/15$, also gegen 0.8. Gemessen hatten wir einen Wert von etwas über 80% (siehe Seite 462). Das zeigt, dass unser sehr stark vereinfachendes Modell zumindest in diesem Fall gar nicht so sehr daneben lag.

Lösung 708: Sei $f_1(n) \in \mathcal{O}(g_1(n))$ und $f_2(n) \in \mathcal{O}(g_2(n))$ und C_1 bzw. C_2 seien Schranken, die das belegen. Für das Produkt gilt dann

$$\frac{f_1(n)f_2(n)}{g_1(n)g_2(n)} = \frac{f_1(n)}{g_1(n)} \cdot \frac{f_2(n)}{g_2(n)} \leq C_1 \cdot C_2,$$

die Schranke C_1C_2 belegt also $f_1(n)f_2(n) \in \mathcal{O}(g_1(n)g_2(n))$.

Lösung 709: Alle drei Aussagen (die beiden aus der Aufgabenstellung und die davor) sind korrekt und keine ist „korrekter“ als die anderen. Aber $\mathcal{O}(4n^3)$ würde man nicht schreiben, weil n^3 ein einfacherer Ausdruck ist. Und $\mathcal{O}(n^4)$ würde man auch nicht schreiben, weil $\mathcal{O}(n^3)$ eine bessere Abschätzung ist. (Sie schränkt die Menge der möglichen Folgen ein, denn alle Folgen, die von der Ordnung n^3 sind, sind auch von der Ordnung n^4, aber nicht umgekehrt.)

Lösung 710: Die Folge $((4n^2 + 5n + 6)/n^2)$ konvergiert (und zwar gegen 4) und ist damit insbesondere beschränkt. Die erste Aussage ist also wahr. Aber man kann das auch ohne Betrachtung des Quotienten sofort sehen, siehe die Bemerkung zu Polynomen vor Aufgabe 709.

Wegen $n^7/(2n^6) = n/2$ für alle $n > 0$ haben wir es offenbar mit einer unbeschränkten Folge zu tun. Die zweite Aussage ist somit falsch.

Wir wissen,[72] dass $(2^n/n!)$ eine Nullfolge ist. Daher muss die Folge der Kehrwerte unbeschränkt, die dritte Aussage also falsch sein.

Für $n > 0$ gilt

$$\frac{n!}{n^n} = \frac{1 \cdot 2 \cdot 3 \cdot \cdots \cdot n}{n \cdot n \cdot n \cdot \cdots \cdot n} \leq \frac{n \cdot n \cdot n \cdot \cdots \cdot n}{n \cdot n \cdot n \cdot \cdots \cdot n} = 1,$$

d.h. die entsprechende Folge ist beschränkt[73] und die vierte Aussage daher wahr.

[72] Siehe Seite 442.

[73] Sie ist sogar eine Nullfolge.

Lösung 711: Die erste Aussage gilt, denn 2^{n+1} ist ja $2 \cdot 2^n$, d.h. wir haben es nur mit einem konstanten Faktor zu tun. Die zweite Aussage stimmt nicht, denn $2^{n/2}$ ist $(\sqrt{2})^n$, d.h. wir haben es mit einer Exponentialfunktion zu tun, deren Basis kleiner als 2 ist. Präziser: Wenn man 2^n durch diesen Term dividiert, erhält man $(\sqrt{2})^n$ und die Folge dieser Werte ist nicht beschränkt.

Lösung 712: Wenn man den Quotienten der beiden Funktionen berechnet, erhält man

$$\frac{n^2 2^n}{2.1^n} = \frac{n^2}{1.05^n}$$

und aus Kapitel 37 wissen wir, dass diese Folge eine Nullfolge, also insbesondere beschränkt, ist.

Tatsächlich sieht man, dass das auch noch funktionieren würde, wenn man n^2 durch ein anderes Polynom wie n^7 ersetzen würde. Ebenfalls würde sich nichts ändern, wenn man statt 2.1 etwa 2.003 nehmen würde; die Zahl muss nur größer als 2 sein. $n^2 2^n$ wächst also (ein bisschen) schneller als 2^n, aber jede Exponentialfunktion mit einer minimal größeren Basis als 2 wächst noch schneller.

Lösung 713: Sie sollten sich aus der Schule an die Umrechnungsformel

$$\log_a(x) = 1/\log_b(a) \cdot \log_b(x)$$

erinnern.[74] Diese besagt, dass sich zwei verschiedene Logarithmen nur durch einen konstanten Faktor unterscheiden. Sie haben also nach Regel (L1) dieselbe Ordnung.

Lösung 714: Für `aop1` sieht es so aus:

```
def aop1 (x, n):
    s = 0                    #  O(1)
    for k in range(n + 1):   #  O(n):   O(1)
        p = 1                #          O(1)
        for i in range(k):   #          O(k):   O(1)
            p *= x           #                  O(1)
        s += p               #          O(1)
    return s                 #  O(1)
```

Die einzige kleine Schwierigkeit hier ist der Ausdruck $\mathcal{O}(k)$, in dem n zunächst nicht vorkommt. Dieses Problem kann man aber dadurch aus der Welt schaffen, dass k *im Durchschnitt* den Wert $n/2$ hat, so dass man $\mathcal{O}(k)$ durch $\mathcal{O}(n)$ ersetzen kann. Dann ergibt sich sofort $\mathcal{O}(n^2)$ für die gesamte Funktion.

Für `aop2` ist es noch einfacher:

```
def aop2 (x, n):
    s = 1                    #  O(1)
    for i in range(n):       #  O(n):   O(1)
```

[74] Wir gehen aber in Kapitel 44 auch noch mal darauf ein.

```
        s *= x              #          O(1)
        s += 1              #          O(1)
    return s                #  O(1)
```

Man erkennt direkt $\mathcal{O}(n)$ als Laufzeitverhalten.

Lösung 715: Es ändert sich gar nichts. Die Formeln für die tatsächlich verbrauchten Zeiteinheiten (wie z.B. $2n^2 + 7n + 7$ für `aop1`) würden sich natürlich ändern, aber nicht die Ordnungen. Eine Multiplikation würde länger dauern, hätte aber, wie eine Addition, immer noch die Ordnung 1.

Dauert eine Multiplikation in einer modernen CPU wirklich länger als eine Addition? Die Frage ist überraschenderweise schwer zu beantworten. Die Antwort hängt unter anderem davon ab, ob in der ALU oder der FPU gerechnet wird, aber auch davon, wie viele Operationen parallel durchgeführt werden und ob und wie die CPU Instruktionen dafür umordnet.

Lösung 716: Durch Umformen mit Schulwissen über den Logarithmus erhält man:

$$\begin{aligned} 1.01^n &\geq n^{1000} \\ n &\geq \log_{1.01} n^{1000} = 1000 \cdot \log_{1.01} n \end{aligned} \tag{A.9}$$

In Python sieht das so aus:

```
from math import log

n = 2
while True:
    if n >= 1000 * log(n, 1.01):
        print(n)
        break
    n += 1
```

Das liefert die Antwort 1 423 969.

Die Gleichung $1.01^n \geq n^{1000}$ lässt sich übrigens nicht symbolisch lösen, jedenfalls nicht unter ausschließlicher Verwendung der sogenannten *elementaren Funktionen*. Man braucht dafür die *lambertsche W-Funktion*, die auch *Produktlogarithmus* genannt wird.[75]

Will man (A.9) als Gleichung in SymPy nach n auflösen, so muss man zunächst so umstellen, dass auf einer Seite null steht:

$$\frac{n}{1000 \cdot \log_{1.01}} - 1 = 0$$

Dann klappt es folgendermaßen:

[75] Vereinfacht ausgedrückt sind die elementaren Funktionen die, die in der Schule besprochen werden, also rationale Funktionen, trigonometrische Funktionen, Logarithmen, Exponentialfunktionen, etc. Mehr dazu auf Seite 507.

```
from sympy import symbols, log, solve
n, a, b = symbols("n, a, b")
solve(n / (log(n, a) * b) - 1, n)
```

Die Lösung, die man erhält, sieht so aus:

```
[-b*LambertW(-log(a)/b)/log(a)]
```

Setzt man konkrete Werte ein, so erhält man die Zahl, die wir oben schon numerisch ermittelt hatten. Allerdings muss man dafür auch noch wissen, dass die W-Funktion verschiedene *Zweige* hat:

`LambertW`

```
from sympy import LambertW, N
N(-1000 * LambertW(-log(1.01) / 1000, -1) / log(1.01))
```

Das geht aber über das mathematische Niveau dieses Buchs hinaus.

Lösung 717: Wir müssen die komplexen Zahlen schwarz färben, für die die Ungleichung $|c^2 + c| \leq 2$ gilt. (Das hätte als Antwort eigentlich schon gereicht.) Setzt man $c = x + yi$, so kommt man durch Ausrechnen auf die Gleichung

$$x^2 + 2x^3 + x^4 + y^2 + 2xy^2 + 2x^2y^2 + y^4 \leq 4$$

Ersetzt man hier $\leq$ durch $=$ und löst nach y auf,[76] so erhält man insgesamt vier Lösungen:[77]

$$y = \pm\sqrt{-1 - 2x - 2x^2 \pm \sqrt{17 + 4x + 4x^2}}/\sqrt{2}$$

Zwei der Lösungen liefern echt komplexe Werte, aber die anderen beiden bilden den Umriss der oberen bzw. unteren Hälfte der Grafik für $n = 1$.

Lösung 718: Im ersten Fall ist $|c| > 2$. Wir zeigen, dass immer $|z_{c,n+1}|/|z_{c,n}| \geq |c| - 1$ gilt. Wegen $|c| - 1 > 2 - 1 = 1$ folgt damit insbesondere auch $|z_{c,n+1}| > |z_{c,n}| > |c|$. Für den Abstand von $z_{c,n+1}$ vom Nullpunkt gilt also mit derselben Argumentation, die wir auf Seite 482 angestellt haben:[78]

$$\begin{aligned} |z_{c,n+1}| &\geq |z_{c,n}^2| - |c| > |z_{c,n}| \cdot |c| - |c| \\ &= |z_{c,n}| \cdot \left(|c| - \frac{|c|}{|z_{c,n}|} \right) > |z_{c,n}| \cdot (|c| - 1) \end{aligned}$$

Wieso impliziert das die Unbeschränktheit der Folge? $|c| - 1$ ist ein konstanter Faktor, der echt größer als 1 ist und um den der Abstand vom Nullpunkt in jedem Schritt mindestens vergrößert wird.

[76] Nein, das müssen Sie nicht können...

[77] Das sind vier, weil es für beide $\pm$ jeweils zwei Möglichkeiten gibt.

[78] Rein technisch ist dies ein Beweis mittels vollständiger Induktion über n, weil wir bei der Abschätzung an zwei Stellen $|z_{c,n}| > |c|$ verwenden.

Nun zum Fall $|c| \leq 2$. Wir beginnen mit einem Folgenwert $z_{c,n}$, dessen Abstand vom Nullpunkt größer als 2 ist. Das liefert uns:

$$\begin{aligned} |z_{c,n+1}| &\geq |z_{c,n}^2| - |c| > 2 \cdot |z_{c,n}| - |c| \\ &\geq 2 \cdot |z_{c,n}| - 2 = 2 \cdot (|z_{c,n}| - 1) \end{aligned}$$

Und das wiederum impliziert $|z_{c,n+1}|/|z_{c,n}| > 2 - 2/|z_{c,n}|$. Induktiv kann man nun wieder schließen, dass auch für $m > n$ die Ungleichung $|z_{c,m+1}|/|z_{c,m}| > 2 - 2/|z_{c,n}|$ gelten muss. Wie oben haben wir also einen entsprechenden konstanten Vergrößerungsfaktor gefunden, nämlich $2 - 2/|z_{c,n}|$.

Lösung 722: Man kann es so schreiben:

```
plotFunc2D(sin, [-2*pi, 2*pi], samples=100)
```

`sin` *ist* bereits eine Funktion, die den Sinus berechnet. Sie noch mal mit `lambda` zu „verpacken“ ist nicht nur unnötig, sondern potentiell auch noch ineffizient.

Lösung 723: Mit der bekannten Parameterdarstellung sieht das so aus:

```
from math import cos, sin, pi

plotCurve2D(lambda a: (cos(a), sin(a)), [0, 2*pi], scaled=True)
```

(Ohne `scaled=True` erhalten wir allerdings eine Ellipse.)

Lösung 724: Mithilfe des Satzes von Pythagoras kann man den oberen und den unteren Halbkreis jeweils als Funktionsgraphen darstellen:

```
from math import sqrt

plotFunc2D([lambda x: sqrt(1-x*x), lambda x: -sqrt(1-x*x)],
           [-1, 1], scaled=True)
```

Siehe dazu auch Aufgabe 422.

Lösung 725: Das macht man so:

```
from math import sin, exp

plotFunc3D(lambda x, y: sin(0.5*x*y)*exp(-0.1*(x*x+y*y)), [-7,7])
```

Lösung 727: Zum Beispiel so:

```
plotSurface3D(lambda u, v: ((4 + cos(v)) * cos(u),
                            (4 + cos(v)) * sin(u),
                            0.5 * sin(v)),
              [0, 2 * pi], zRange=[-pi, pi])
```

Lösung 728: So eine Kurve nennt man *Helix*:

```
plotCurve3D(lambda t: (4 * cos(t), 4 * sin(t), t), [0, 6 * pi])
```

Lösung 729: Wir müssen dafür die Bibliothek CMATH (Kapitel 35) verwenden:

```
from cmath import sqrt

plotComplexAbs(sqrt, [-2, 2])
```

Lösung 730: Das macht man folgendermaßen:

```
from math import sqrt

def f (x, y, z):
    r = sqrt(x*x + y*y + z*z)
    return (x/r, y/r, z/r) if r != 0 else (0, 0, 0)

plotVectorField3D(f, [-4, 4])
```

Lösung 732: So könnte man es machen:

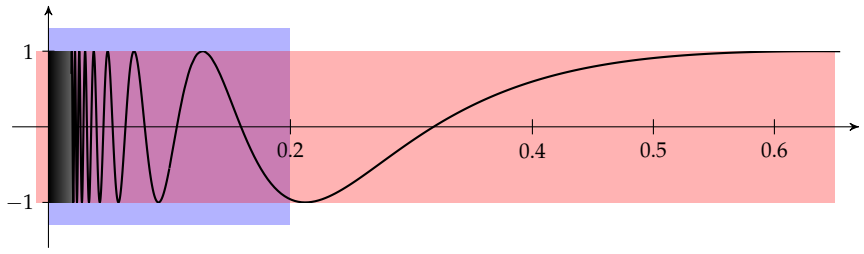
```
from math import sin

plotFunc2D(lambda x: sin(1 / x), [-1, 1], samples=400)
```

Insgesamt sieht die Grafik etwas unbefriedigend aus; mehr dazu im Text, der auf diese Aufgabe folgt. Nebenbei: Was passiert, wenn Sie `samples=401` eingeben? Können Sie das erklären?

Lösung 736: Das Ergebnis ist natürlich 42. Mit den Rechenregeln ergibt sich:

$$\lim_{x\to 7}(x^2-7) = \left(\lim_{x\to 7} x\right)\cdot\left(\lim_{x\to 7} x\right) - \left(\lim_{x\to 7} 7\right) = 7\cdot 7 - 7 = 42$$

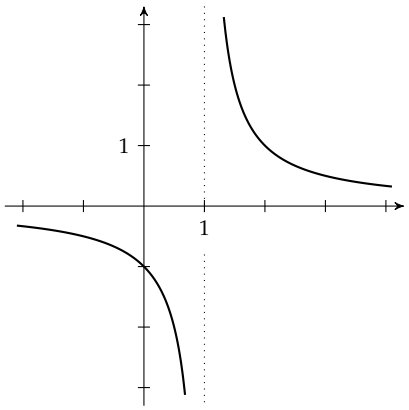
Lösung 737: Eine Grafik dazu könnte so aussehen:



Der entscheidende Punkt ist, dass der rote Bereich immer gleich bleiben würde. Wir können das blaue Rechteck schmaler und schmaler machen (die rechte Grenze würde

sich immer weiter in Richtung der Null bewegen), aber der rote Bereich würde sich *nicht* zusammenziehen.

Lösung 738: Nein. Wenn man sich von rechts nähert, divergieren die entsprechenden Folgen bestimmt gegen ∞, von links allerdings gegen $-\infty$. Nähert man sich mit Folgen, die abwechselnd Werte links und rechts von 1 annehmen, divergieren die Folgen der Funktionswerte sogar gar nicht bestimmt.[79]



Die Funktion hat an der Stelle 1 allerdings *einseitige* Grenzwerte; siehe Seite 504.

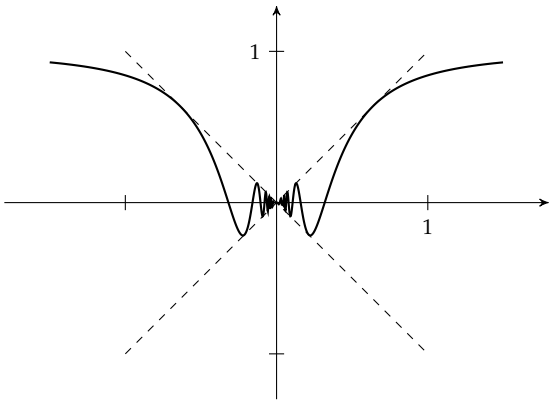
Lösung 739: Ja.

Lösung 740: Nein. Man kann das wie am Anfang des Kapitels begründen, nur mit den Kehrwerten. Nimmt man die bestimmt gegen ∞ divergierende Folge (πn), so erhält man durch Einsetzen in den Sinus immer den Wert 0. Betrachtet man stattdessen die Folge $(2\pi n + \pi/2)$, so erhält man immer den Wert 1. Damit hat man schon zwei Folgen, die nicht gegen denselben Wert konvergieren.

Anschaulich ist ohnehin klar, dass es nicht klappen kann, weil der Sinus ja ständig zwischen 1 und -1 oszilliert und sich nicht etwa einem Wert nähert, wenn man „immer weiter nach rechts“ geht.

Lösung 741: Der Grenzwert existiert: $\lim_{x\to 0} x \sin 1/x = 0$. Das liegt daran, dass der Sinus nur Werte zwischen -1 und 1 annehmen kann. Durch die Multiplikation mit x wird der Funktionswert zwischen den beiden gestrichelten Linien „eingezwängt“:

[79] Falls Ihnen die Antwort nicht sofort klar war, haben Sie sich hoffentlich die Funktion von PYTHON zeichnen lassen!

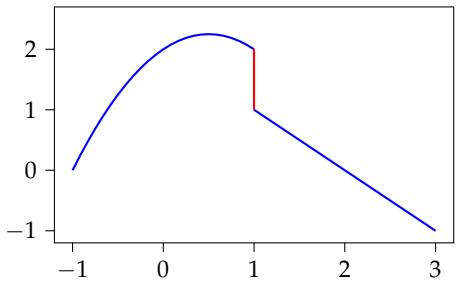


Der Funktion bleibt gar nichts anderes übrig, als gegen 0 zu konvergieren.

Lösung 743: Man könnte das z.B. so machen:

```
from plot import *

plotFunc2D(lambda x: -x**2+x+2 if x <= 1 else -x+2,
           [-1,3], samples=500)
```

Und würde in etwa dieses Ergebnis bekommen:



Das ist natürlich *nicht* der Graph einer Funktion, weil eine Funktion von $\mathbb{R}$ nach $\mathbb{R}$ ja keine senkrechten Abschnitte enthalten darf. Die falsche Grafik kommt dadurch zustande, dass der Computer einfach an verschiedenen Stellen die Funktionswerte berechnet und die so ermittelten Punkte verbindet. Bei stetigen Funktionen ist das gewollt und sieht gut aus; an Unstetigkeitsstellen wie hier kann es zu falschen Grafiken führen.[80] Erschwerend kommt hinzu, dass die von PYTHON gezeichnete Linie gar nicht wirklich senkrecht verläuft, weil die Stellen, an denen Funktionswerte berechnet werden, natürlich *nebeneinander* liegen. Ich habe daher im obigen Beispiel einen hohen Wert für `samples` gewählt, damit das Ergebnis zumindest *fast* senkrecht ist.

Lösung 744: Hier eine Beispiellösung, die eine Antwort ausgibt, die auf mindestens n Stellen nach dem Komma richtig ist – und die ohne Prüfung davon ausgeht, dass $f(a)$ und $f(b)$ tatsächlich unterschiedliche Vorzeichen haben.

[80] Manche CAS, z.B. MATHEMATICA, sind clever genug, solche Sprünge zu erkennen.

```
def zero (f, a, b, n):
    eps = 10 ** -n
    m = (a + b) / 2
    while abs(a - m) > eps:
        if f(m) == 0:
            return round(m, n)
        if f(a) * f(m) < 0:
            b = m
        else:
            a = m
        m = (a + b) / 2
    return round(m, n)
```

Dieses Verfahren nennt man *Bisektion*. (Die Funktion `round` macht das, was ihr Name vermuten lässt.)

Lösung 745: Für die erste Folge formen wir zunächst um:

$$(\log_{10} n)/n = 1/n \cdot \log_{10} n = \log_{10} n^{1/n} = \log_{10} \sqrt[n]{n}$$

Da wir bereits wissen, dass $\lim_{n\to\infty} \sqrt[n]{n} = 1$ gilt, muss die Folge gegen $\log_{10} 1 = 0$ konvergieren. Bei der zweiten Folge konvergiert der Term unter der Wurzel nach Kapitel 37 gegen $1/4$. Der Limes der Gesamtfolge ist daher $\sqrt{1/4} = 1/2$.

Man kann das auch mit Python überprüfen:

```
from sympy import *

n = symbols("n")
limit(log(n, 10) / n, n, oo)
limit(sqrt((n**3+23*n+7) / (4*n**3+2*n**2-42)), n, oo)
```

Lösung 746: Wir betrachten wie gewohnt den Quotienten, in diesem Fall also $(\log n)/n$. Wie in der vorherigen Aufgabe formt man das um und sieht, dass die Folge $((\log n)/n)$ gegen 0 konvergiert. Und da die Folge konvergiert, ist sie beschränkt.

Lösung 747: Wenn eine Funktion von $\mathbb{R}^2$ nach $\mathbb{R}$ stetig ist, dann hat der Funktionsgraph keine „Risse“ oder „Löcher“. Für ein Beispiel einer unstetigen Funktion probieren Sie mal das hier aus:

```
from plot import *
from math import sin, cos, pi
%matplotlib inline

def f (x, y):
    p = x * y
```

```
    return sin(p) if p > 0 else cos(p)
plotFunc3D(f, [-pi/2, pi/2])
```

Diese Funktion ist sowohl auf der gesamten x- als auch auf der gesamten y-Achse unstetig. Wenn Sie einen Punkt auf einer der Achsen wählen und drum herum einen kleinen Kreis, dann wird es zwischen den Funktionswerten der Punkte in diesem Kreis große Unterschiede geben. Diese Unterschiede verschwinden auch nicht, egal wie klein Sie den Kreis machen.

Lösung 748: In etwa so:

```
for n in range(1, 10):
    print(n, sum(0.5 ** k for k in range(n)))
```

Lösung 750: Der Anfang dieser Reihe sieht so aus: $1/1 + 1/4 + 1/9 + \ldots$. Die ersten drei Partialsummen sind also 1, $5/4$ und $49/36$. Das fünfte Reihenglied ist $1/25$ und der Grenzwert der Folge der Reihenglieder ist 0. Wenn Sie mit PYTHON herumspielen, werden Sie (zu Recht) vermuten, dass der Grenzwert irgendwo zwischen 1.6 und 1.7 liegt. Den tatsächlichen Grenzwert erfahren Sie so aber nicht. Wir werden auf diese Reihe aber noch zurückkommen.

Lösung 751: Als Reihe kann man das so schreiben: $\sum_{n=0}^{\infty}(-1)^n$. Die ersten vier Partialsummen sind 1, 0, 1 und 0. Und so geht es auch weiter; die Folge der Partialsummen konvergiert also offensichtlich nicht, die Reihe also definitionsgemäß auch nicht. Damit hat sich auch die Frage, welche der Klammerungen „richtig“ ist, erledigt: diese „unendliche Summe“ existiert gar nicht.

Lösung 752: Vielleicht sind Sie nicht auf die Lösung gekommen, aber wenn Sie sie sehen, wird sie Ihnen hoffentlich ganz offensichtlich vorkommen:

$$\sum_{n=0}^{\infty}(a_n - a_{n-1})$$

Dabei setzen wir $a_{-1} = 0$, damit das erste Reihenglied definiert ist. Die ersten Partialsummen sehen so aus:

$$\sum_{n=0}^{0}(a_n - a_{n-1}) = a_0 - a_{-1} = a_0$$
$$\sum_{n=0}^{1}(a_n - a_{n-1}) = (a_0 - a_{-1}) + (a_1 - a_0) = a_1$$
$$\sum_{n=0}^{2}(a_n - a_{n-1}) = (a_0 - a_{-1}) + (a_1 - a_0) + (a_2 - a_1) = a_2$$

Lösung 753: $\sum_{k=0}^{\infty} 1/(k+1)^2 = \sum_{k=1}^{\infty} 1/k^2$. Wenn Ihnen das nicht offensichtlich erscheint, schreiben Sie jeweils die ersten Partialsummen auf und vergleichen Sie.

Lösung 754: Die ersten Partialsummen sind $1/4 = 1/2 - 1/4$, $3/8 = 1/2 - 1/8$ und $7/16 = 1/2 - 1/16$. Die Folge wird offenbar gegen $1/2$ konvergieren. Ihnen ist hoffentlich klar, dass man das auch so ausdrücken kann:

$$\sum_{k=2}^{\infty} \frac{1}{2^k} = \left(\sum_{k=0}^{\infty} \frac{1}{2^k} \right) - \frac{1}{2^0} - \frac{1}{2^1} = 2 - 1 - \frac{1}{2} = \frac{1}{2}$$

(Man „überspringt“ also die ersten beiden Summanden, die daher nichts mehr zur Gesamtsumme beitragen.)

Lösung 755: Das Ergebnis muss $1/(1 - 1/3)$ sein, also $3/2$ bzw. 1.5. Das war hoffentlich auch Ihre Vermutung in Aufgabe 749.

Lösung 756: Gegen $1/(1 + 1/4) - 1 = -1/5$. Sie haben hoffentlich nicht vergessen, den ersten Summanden abzuziehen!

Lösung 757: Die erste Frage beantwortet man so:

$$\begin{aligned} 0.\overline{7} &= 7 \cdot 10^{-1} + 7 \cdot 10^{-2} + 7 \cdot 10^{-3} + 7 \cdot 10^{-4} + \ldots \\ &= 7 \cdot \sum_{k=1}^{\infty} \frac{1}{10^k} = 7 \cdot \left(\frac{1}{1 - \frac{1}{10}} - 1 \right) = \frac{7}{9} \end{aligned}$$

Für $0.\overline{9}$ ergibt sich analog $9/9$, also 1.

Lösung 758: Im Fall $q = 1$ lässt sich unsere Formel $1/(1 - q)$ nicht verwenden, weil bereits die Formel für die endliche geometrische Summe nicht mehr anwendbar ist. Es ist aber offensichtlich, dass für $q = 1$ die geometrische Reihe die Form

$$1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + \ldots$$

hat und sicher nicht konvergiert (weil die Partialsummen immer größer werden). Im Fall $q = -1$ ergibt sich die Reihe aus Aufgabe 751, die ebenfalls nicht konvergiert.

Sollten Sie bei der Frage auch an (echt) komplexe Zahlen q mit $|q| = 1$ gedacht haben:[81] auch in diesem Fall konvergiert die Reihe nicht.

Lösung 759: Dem Hinweis entsprechend erhält man, indem man beide Ausdrücke auf einen gemeinsamen Nenner bringt:

$$\frac{1}{k} - \frac{1}{k+1} = \frac{k+1}{k(k+1)} - \frac{k}{k(k+1)} = \frac{1}{k(k+1)} = \frac{1}{k^2 + k}$$

Analog zu Aufgabe 752 ergibt sich nun für die Partialsummen:

$$\sum_{k=1}^{n} 1/(k^2 + k) = \sum_{k=1}^{n} \left(\frac{1}{k} - \frac{1}{k+1} \right) = 1 - \frac{1}{n+1}$$

Wenn n immer größer wird, konvergiert dieser Term gegen eins.

[81] Bravo!

Lösung 760: Für das Ausklammern sieht das z.B. so aus: Die n-te Partialsumme ist $\sum_{k=0}^{n} \lambda \cdot a_k = \lambda \cdot \sum_{k=0}^{n} a_n$. Da man bei Folgen Faktoren ausklammern darf (siehe Seite 444), ergibt sich

$$\sum_{n=0}^{\infty} \lambda \cdot a_n = \lim_{n\to\infty} \sum_{k=0}^{n} \lambda \cdot a_k = \lim_{n\to\infty} \lambda \cdot \sum_{k=0}^{n} a_n = \lambda \cdot \lim_{n\to\infty} \sum_{k=0}^{n} a_n = \lambda \cdot \sum_{n=0}^{\infty} a_n$$

Für das „Auseinanderziehen“ von Reihen argumentiert man analog.

Lösung 761: Das geht so:

$$\begin{aligned}\sum_{k=2}^{\infty} \frac{3}{5^k} &= 3 \cdot \sum_{k=2}^{\infty} \frac{1}{5^k} = 3 \cdot \left(-1 - \frac{1}{5} + \sum_{k=0}^{\infty} \frac{1}{5^k}\right) \\ &= 3 \cdot \left(-1 - \frac{1}{5} + \frac{1}{1-1/5}\right) = \frac{3}{20} \\ \sum_{k=0}^{\infty} \left(\frac{1}{5^k} + \frac{1}{7^k}\right) &= \left(\sum_{k=0}^{\infty} \frac{1}{5^k}\right) + \left(\sum_{k=0}^{\infty} \frac{1}{7^k}\right) \\ &= \frac{1}{1-1/5} + \frac{1}{1-1/7} = \frac{29}{12}\end{aligned}$$

Lösung 762: Die harmonische Reihe, die nicht konvergiert, liefert so ein Beispiel, denn $(1/n)$ ist ja eine Nullfolge.

Lösung 763: Nein. Die Reihe selbst konvergiert zwar (als allgemeine alternierende harmonische Reihe mit $\alpha = 1$). Die „andere“ Reihe, die Reihe der Beträge, ist aber die harmonische Reihe, die divergiert.

Lösung 766: Es sollte eins herauskommen, weil es sich nur um zwei verschiedene Arten handelt, denselben Wert darzustellen. Es kommt aber nur *fast* eins heraus. Wir wissen ja schon, dass Computer nicht immer korrekte Ergebnisse liefern...

Lösung 767: Nach der Bernoullischen Ungleichung gilt $(1 + x/n)^n \geq 1 + x$. Allerdings nur dann, wenn $x/n \geq -1$ gilt. Ist $x \geq -1$, so stimmt das, und es folgt $\exp(x) \geq 1 + x$. Für $x < -1$ gilt das aber auch, weil wir ja schon wissen, dass $\exp(x) > 0$ gilt.

Außerdem gilt für $x > 0$ nach der zweiten Variante der Bernoullischen Ungleichung (siehe Aufgabe 686), dass fast alle Folgenglieder von $((1 + x/n)^n)$ größer als $1 + x$ sind. Wegen der Monotonie der Folge muss dann auch $\exp(x) > 1 + x$ gelten.

Lösung 769: Bis auf Rundungsfehler sollte in beiden Fällen wieder x herauskommen. Die Rundungsfehler gibt es aber definitiv: probieren Sie z.B. `exp(log(42))` aus.

Lösung 770: Aus $\exp(-x) = 1/\exp(x)$ wird zunächst $\exp(-x) = 1/y$. Nimmt man nun den Logarithmus, so ergibt sich $-x = \log 1/y$. Wegen $y = \exp x$, kann man nun x noch durch $\log y$ ersetzen. Insgesamt hat man dann:

$$\log \frac{1}{y} = -\log y$$

Lösung 771: Mit $y = \exp x$ haben wir $\exp(bx) = y^b$ und durch Logarithmieren ergibt sich $bx = \log y^b$. Ersetzt man nun noch x, so wird daraus:

$$\log y^b = b \log y$$

Lösung 773: Für $a > 1$ gilt $\ln a > 0$. Aus $x_1 > x_2$ folgt somit $x_1 \ln a > x_2 \ln a$. Wegen der Monotonie der Exponentialfunktion ergibt sich:

$$a^{x_1} = \exp(x_1 \ln a) > \exp(x_2 \ln a) = a^{x_2}$$

Für $a < 1$ gilt $\ln a < 0$ und damit $x_1 \ln a < x_2 \ln a$. In diesem Fall ist $x \mapsto a^x$ also streng monoton fallend.

Lösung 774: Die Antworten sind 4 und 5. Ich hoffe, das war Ihnen klar…

`log`

Lösung 775: Das geht so:

```
from math import log

log(10000, 10), log(32, 2)
```

Lösung 776: Natürlich. Das kann man sich mithilfe des eben hergeleiteten Zusammenhangs sofort überlegen:

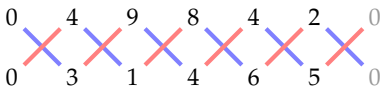
$$\begin{aligned}\log_a y_1 y_2 &= \frac{1}{\ln a} \cdot \ln y_1 y_2 = \frac{1}{\ln a} \cdot (\ln y_1 + \ln y_2) = \frac{1}{\ln a} \cdot \ln y_1 + \frac{1}{\ln a} \cdot \ln y_2 \\ &= \log_a y_1 + \log_a y_2\end{aligned}$$

Auch die Beziehungen aus den Aufgaben 770 und 771 gelten für beliebige Logarithmen.

Lösung 777: Es ergibt sich:

$$\begin{aligned}a &= \frac{1}{\sqrt{500}} \\ b &= \sqrt{5} \cdot \sqrt{500} = \sqrt{2500} = 50 \\ c &= \log_{10} 20b = \log_{10} 1000 = 3\end{aligned}$$

Lösung 779: Wir haben bereits gesehen, dass es für die Formel keine Rolle spielt, wo der Nullpunkt liegt. Wir können also z.B. die linke untere Ecke des Sechsecks $(0,0)$ nennen. Die gegen den Uhrzeigersinn nächste wäre dann $(4,3)$ und so weiter. Das ergibt insgesamt:

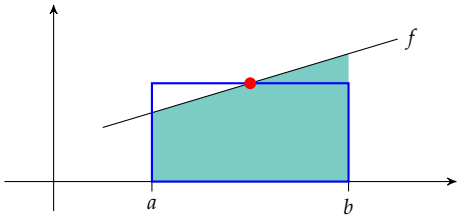


Die gesuchte Fläche ist also die Hälfte von

$$\begin{aligned}&0 \cdot 3 + 4 \cdot 1 + 9 \cdot 4 + 8 \cdot 6 + 4 \cdot 5 + 2 \cdot 0 \\ &- 4 \cdot 0 - 9 \cdot 3 - 8 \cdot 1 - 4 \cdot 4 - 2 \cdot 6 - 0 \cdot 5 = 45\end{aligned}$$

bzw. 22.5. (Siehe hierzu auch Aufgabe 833, in der wir die Fläche noch mal auf eine andere Art berechnen.)

Lösung 781: In der Skizze unten ist die Fläche unter der Kurve grün dargestellt und das von unserer PYTHON-Funktion berechnete Rechteck blau umrandet. Es wird deutlich, dass wir links genauso viel hinzufügen, wie wir rechts wegnehmen.



Lösung 782: Wir verwenden die Funktion $x \mapsto \sqrt{1-x^2}$ (siehe Aufgabe 724). Da die Fläche des Einheitskreises π ist, sollten wir für große n gute Näherungswerte für $\pi/2$ bekommen.

```
from math import sqrt, pi

2 * area(lambda x: sqrt(1 - x*x), -1, 1, 1000), pi
```

Lösung 783: Das sollte so aussehen:

```
def f (x):
    return -x*x + x + 2 if x <= 1 else -x + 2

area(f, -1, 2, 10000), \
area(lambda x: -x*x + x + 2, -1, 1, 10000) + \
area(lambda x: -x + 2, 1, 2, 10000)
```

Die Werte sind wie erwartet fast gleich. (Können Sie erklären, warum sie nicht exakt gleich sind?)

Lösung 784: Zum Berechnen der Strecke geht man so vor:

```
from math import sin, pi

def f (t):
    return 25 * sin(pi * t / 120)
area(f, 0, 120, 10000)
```

Sie sind also knapp zwei Kilometer gefahren. Und gezeichnet sieht es so aus:

```
from plot import *

plotFunc2D(f, [0, 120])
```

Wenn Sie einen Funktionsgraphen haben, der die Geschwindigkeit in Abhängigkeit von der Zeit darstellt, dann entspricht die zurückgelegte Strecke der Fläche unter der Kurve.

Lösung 785: Der Geschwindigkeit.

Lösung 786: Das geht natürlich mit

```
quad(f, 0, 120)
```

Dabei soll `f` die in der besagten Aufgabe definierte Funktion sein.

Lösung 787: Zunächst müssen wir herausfinden, wo sich f und g schneiden. Die Schnittpunkte ergeben sich als Lösungen der Gleichung $f(x) = g(x)$, aus der durch einfaches Umformen $4x^2 = 1$ bzw. $x^2 = 1/4$ wird. Die grüne Fläche liegt also zwischen $x_1 = -1/2$ und $x_2 = 1/2$. Die gesuchte Fläche erhält man nun, indem man zwischen x_1 und x_2 die Fläche unter f von der Fläche unter g subtrahiert:

$$\left(\int_{x_1}^{x_2} g(x)\,\mathrm{d}x\right) - \left(\int_{x_1}^{x_2} f(x)\,\mathrm{d}x\right)$$

Das kann man nun direkt mit zwei Aufrufen von `quad` berechnen, oder man kann es noch zu

$$\int_{x_1}^{x_2} (g(x) - f(x))\,\mathrm{d}x$$

vereinfachen.

```
quad(lambda x: -2*x*x + 0.8, -0.5, 0.5)[0] - \
quad(lambda x: 2*x*x - 0.2, -0.5, 0.5)[0], \
quad(lambda x: -4*x*x + 1, -0.5, 0.5)[0]
```

Die beiden Werte sind natürlich (bis auf Rundungsfehler) identisch. Der exakte Wert der Fläche ist $2/3$; das werden wir später in Aufgabe 832 nachweisen.

Beachten Sie, wie hilfreich die Orientierung des Volumens hier ist. Der Flächenteil von f, der oberhalb der x-Achse liegt, ist positiv und wird von der Fläche unterhalb von g (die komplett oberhalb der x-Achse liegt) abgezogen. Der Flächenteil von f, der unterhalb der x-Achse liegt, ist negativ und wird korrekterweise hinzuaddiert.

Dass das klappt, kann man sich auch dadurch klarmachen, dass der Wert des Ausdrucks $g(x) - f(x)$ sich nicht ändert, wenn man beide Funktionen um den gleichen Betrag nach oben schiebt, damit sich alles oberhalb der x-Achse abspielt.

Lösung 788: *Jede* konstante Funktion hat als Funktionsgraphen eine waagerechte Linie, ist also eine Gerade mit der Steigung null. Daher ist die Ableitung an jeder Stelle null.

Lösung 789: Mithilfe des Hinweises sollte man auf diese Umformung kommen:

$$\frac{f(x) - f(x_0)}{x - x_0} = \frac{x^3 - x_0^3}{x - x_0} = \frac{(x^2 + xx_0 + x_0^2)(x - x_0)}{x - x_0} = x^2 + xx_0 + x_0^2$$

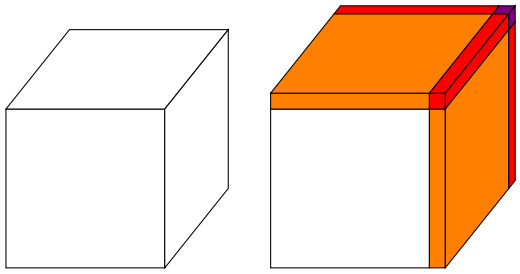
Damit ergibt sich $f'(x_0) = \lim_{x \to x_0}(x^2 + xx_0 + x_0^2) = 3x_0^2$.

Lösung 790: Aufgrund der dritten Skizze könnte einem die Betragsfunktion $x \mapsto |x|$ einfallen. Und in der Tat existiert der Grenzwert

$$\lim_{x\to 0} \frac{|x|-|0|}{x-0} = \lim_{x\to 0} \frac{|x|}{x}$$

nicht. Durch $a_n = (-1)^n/n$ ist z.B. eine Nullfolge (a_n) definiert, aber $(|a_n|/a_n)$ konvergiert nicht, sondern springt immer zwischen 1 und -1 hin und her.

Lösung 792: Die Länge x wird auf das Volumen $f(x) = x^3$ abgebildet. Wird x um $\mathrm{d}x$ vergrößert, so besteht $\mathrm{d}f$ aus den drei[82] orangen Blöcken der Größe $x^2\mathrm{d}x$. Hinzu kommen drei rote Blöcke der Größe $x(\mathrm{d}x)^2$, die man vernachlässigen kann, sowie ein noch unbedeutenderes violettes Würfelchen mit dem Volumen $(\mathrm{d}x)^3$.



Lösung 793: Der Veränderung des Kosinuswertes entspricht die waagerechte braune Kathete, die natürlich die Länge $\mathrm{d}\alpha \cdot \sin\alpha$ hat. Allerdings wird der Kosinus durch das Vergrößern des Winkels kleiner; daher haben wir:

$$\frac{\mathrm{d}}{\mathrm{d}x}\cos x = -\sin x$$

Lösung 794: So: $(\lambda f)'(x_0) = \lambda \cdot f'(x_0)$.

Lösung 795: Die Ableitungen von $x \mapsto x^2$, $x \mapsto x$ und $x \mapsto -5$ kennen wir schon. Der Rest ergibt sich aus den gerade gelernten Rechenregeln. $f'(x) = 8x + 2$. Die gesuchte Steigung ist $f'(5) = 42$.

Lösung 796: Die Ableitungen von Sinus und Kosinus kennen wir schon. Mit der Leibnizregel folgt:

$$f'(x) = (\cos x)(\cos x) + (\sin x)(-\sin x) = \cos^2 x - \sin^2 x = \cos 2x$$

Der letzte Schritt entspricht Gleichung (22.3). (Vergessen? Macht nichts...)

Lösung 797: Man kann den Faktor λ als eine konstante Funktion $g(x) = \lambda$ betrachten, deren Ableitungsfunktion natürlich $g'(x) = 0$ ist. Wendet man mit diesem g die Produktregel auf fg an, so ergibt sich (46.3).

Lösung 798: Mit der gerade hergeleiteten Regel erhält man die Ableitungsfunktionen $x \mapsto -1/x^2$ und $x \mapsto -2/x^3$.

[82] Einen davon sieht man nicht, weil er auf der „Rückseite“ des Würfels ist.

Lösung 799: Man muss nur die beiden Regeln anwenden und am Ende alles auf einen gemeinsamen Nenner bringen. Und sich dabei nicht verrechnen...

$$\left(\frac{f}{g}\right)'(x) = \left(f \cdot \frac{1}{g}\right)'(x) = f'(x) \cdot \frac{1}{g(x)} + f(x) \cdot \left(-\frac{g'(x)}{g(x)^2}\right)$$
$$= \frac{f'(x)g(x) - f(x)g'(x)}{g(x)^2}$$

Das ist die *Quotientenregel*. In Kurzform:

$$\left(\frac{f}{g}\right)' = \frac{f'g - fg'}{g^2}$$

Lösung 800: Hier wenden wir natürlich an, dass der Tangens der Quotient von Sinus und Kosinus ist. Mit der Quotientenregel ergibt sich:

$$(\tan x)' = \left(\frac{\sin x}{\cos x}\right)' = \frac{\cos^2 x + \sin^2 x}{\cos^2 x}$$

Dieses Ergebnis kann man nun noch auf zwei Arten umformen. Entweder Sie schreiben es als Summe zweier Brüche und kürzen:

$$(\tan x)' = 1 + \tan^2 x$$

Oder Sie erinnern sich, dass man den Zähler drastisch vereinfachen kann:

$$(\tan x)' = \frac{1}{\cos^2 x}$$

Beides ist richtig und kann je nach Situation hilfreich sein.

Lösung 801: Die ersten beiden Ableitungen sind $\exp(\sin x)\cos x$ und $-3\sin x\,\cos^2 x$. Im zweiten Fall ist die „äußere" Funktion $x \mapsto x^3$.

Für h_3 kann man zunächst die Ableitung der „inneren" Funktion $k(x) = \exp(x^2)$ berechnen: $k'(x) = 2x\exp(x^2)$. Nun wendet man erneut die Kettenregel an, und zwar auf $x \mapsto \tan k(x)$. Das ergibt dann insgesamt $h_3'(x) = 2x\exp(x^2) \cdot (1 + \tan^2 \exp(x^2))$.

Lösung 802: Man kann die Ableitung von $x \mapsto \cos^2 x = (\cos x)(\cos x)$ mit der Leibnizregel ausrechnen und dann diese Regel noch mal anwenden, um $h_2(x) = \cos x \cos^2 x$ zu differenzieren. Machen Sie das zur Übung ruhig mal und verifizieren Sie, dass beide Wege nach Rom führen.

Lösung 803: Die „innere Funktion" ist $x + \pi/2$ und ihre Ableitung ist einfach eins. Damit erhalten wir als Ableitung des Kosinus $\cos(x + \pi/2)$. Da der Kosinus eine gerade Funktion ist, gilt $\cos(x + \pi/2) = \cos(-x - \pi/2)$. Wenn wir erneut Kapitel 23 zurate ziehen, erhalten wir: $(\cos x)' = \cos(-x - \pi/2) = \sin(-x) = -\sin x$.

Lösung 804: Alle drei sind Umkehrfunktionen von Funktionen, deren Ableitungen wir schon kennen:

$$x \mapsto x^2 \qquad x \mapsto \exp(x) \qquad x \mapsto \sin x$$

Im nächsten Abschnitt wird erklärt, wie man Umkehrfunktionen differenziert.

Lösung 805: Für den Logarithmus ergibt sich:

$$\frac{\mathrm{d}}{\mathrm{d}x}\ln x = \frac{1}{\exp(\ln x)} = \frac{1}{x}$$

Wie bei der Ableitung des Arkustangens ist das überraschend, wenn man es zum ersten Mal sieht. Die Exponentialfunktion wird über einen Grenzwertprozess, eine unendliche Reihe, definiert, der Logarithmus ist die Umkehrfunktion davon, und dessen Ableitung ist einfach der Kehrwert!

Und wie ist das mit der Quadratwurzel? Das ist die Umkehrfunktion von $x \mapsto x^2$. Daher erhalten wir:

$$\frac{\mathrm{d}}{\mathrm{d}x}\sqrt{x} = \frac{1}{2\sqrt{x}}$$

Lösung 806: Wenn Sie auf das Vorzeichen geachtet und sich nicht verrechnet haben, dann sollte das hier herausgekommen sein:

$$\frac{\mathrm{d}}{\mathrm{d}x}\arccos x = -\frac{1}{\sqrt{1-x^2}}$$

Lösung 807: Wenn wir die Ableitungsfunktion des Arkuskosinus g nennen, so ergibt sich als Ableitung der linken Seite $\cos(\arccos x)\cdot g(x) = x\cdot g(x)$. Auf der rechten Seite erhält man $-x/\sqrt{1-x^2}$. Teilt man beide Seiten durch x, so ergibt sich dieselbe Darstellung, die wir schon in der letzten Aufgabe ermittelt haben.

Lösung 808: Wenn man sich erinnert, dass a^x als $\exp(x\ln a)$ definiert ist, ist das eine einfache Anwendung der Kettenregel, denn $\ln a$ ist ja nur ein konstanter Faktor:

$$\frac{\mathrm{d}}{\mathrm{d}x}a^x = \ln a\cdot\exp(x\ln a) = \ln a\cdot a^x$$

(Und wenn $a = \mathrm{e}$ gilt, dann ist $\ln a = 1$ und oben steht, dass e^x die Ableitung von e^x ist, was wir schon wissen.)

Lösung 809: Auch das geht mit der Kettenregel ganz einfach:

$$\frac{\mathrm{d}}{\mathrm{d}x}x^r = \frac{\mathrm{d}}{\mathrm{d}x}\exp(r\ln x) = \frac{r}{x}\cdot\exp(r\ln x) = \frac{r}{x}\cdot x^r = rx^{r-1}$$

Lösung 810: Ja, denn es gilt ja $\sqrt{x} = x^{1/2}$.

Lösung 814: Nein, siehe Aufgabe 790.

Lösung 815: Meine Lösung sieht so aus:[83]

```
def bolzanoList (a, b, A, B, n):
    if n == 0:
        return [(a, A), (b, B)]
    n = n - 1
    x1 = 0.625*a + 0.375*b
    y1 = 0.375*A + 0.625*B
    x2 = 0.5*a + 0.5*b
```

[83] Zum verschachtelten `def` siehe Kapitel 52.

```
    y2 = 0.5*A + 0.5*B
    x3 = 0.125*a + 0.875*b
    y3 = -0.125*A + 1.125*B
    return bolzanoList(a, x1, A, y1, n) + \
           bolzanoList(x1, x2, y1, y2, n) + \
           bolzanoList(x2, x3, y2, y3, n) + \
           bolzanoList(x3, b, y3, B, n)

def bolzanoFunction (n):
    L = bolzanoList(0, 2, 0, 1.5, n)
    def B (x):
        X0 = -1
        Y0 = 0
        for X, Y in L:
            if x <= X:
                return Y0 + (Y - Y0) / (X - X0) * (x - X0)
            X0, Y0 = X, Y
    return B

from plot import *
plotFunc2D([bolzanoFunction(k) for k in range(4)], [0, 2],
           samples=500)
```

Lösung 816: Die Ableitung ist $s'(t) = 25t/108$, also $s'(10) = 125/54 \approx 2.31$.

Lösung 817: Die beiden ersten Grenzwerte verschwinden. Das sollte Sie *nicht* überraschen, denn wenn sich zwei stetige Funktionen in einem Punkt treffen, dann geht der Abstand der beiden Funktionen natürlich gegen null, wenn man sich diesem Punkt nähert. Für den dritten Grenzwert erhält man hingegen:

$$\lim_{x\to 2} \frac{f(x) - s(x)}{x-2} = \lim_{x\to 2} \frac{x^2 - 2x}{x-2} = \lim_{x\to 2} \frac{x(x-2)}{x-2} = \lim_{x\to 2} x = 2$$

Entscheidend ist, dass in diesem Fall *nicht* null herauskommt. Der Abstand von f und s geht zwar gegen null, aber nicht wesentlich schneller als x gegen 2 geht. In Gleichung (46.10) ergibt sich wegen der Division durch $x - x_0$ nur dann null, wenn man die Tangente einsetzt.

Lösung 818: Ja, sie hat die Häufungspunkte 1 und -1. Es gibt zu beiden Zahlen konstante Teilfolgen.

Lösung 819: Ja, sie hat den Häufungspunkt 0, der gleichzeitig auch ihr Grenzwert ist. Beachten Sie, dass nach unserer obigen Definition insbesondere jede Folge eine Teilfolge von sich selbst ist. (Dafür setzt man einfach $n_k = k$.)

Lösung 820: Nein. Die Folge (n) hat zum Beispiel keinen Häufungspunkt. (n) selbst divergiert bestimmt gegen unendlich und jede Teilfolge auch.

Lösung 821: Nein. Seien c und d unterschiedliche Häufungspunkte der Folge (a_n) und sei ε der Abstand von c und d. Weil eine Teilfolge gegen c konvergiert, liegen dann fast alle Glieder dieser Teilfolge, also unendlich viele Glieder von (a_n), im Intervall $(c-\varepsilon/2, c+\varepsilon/2)$. Dann können aber nicht fast alle Folgenglieder von (a_n) im Intervall $(d-\varepsilon/2, d+\varepsilon/2)$ liegen. Daher kann d nicht Grenzwert von (a_n) sein. Analog, mit vertauschten Rollen, ergibt sich, dass c nicht Grenzwert von (a_n) sein kann. Und mit demselben Argument folgt dann auch, dass keine *andere* Zahl Grenzwert sein kann.

Lösung 822: Unendlich viele! Wir hatten uns ja schon überlegt, dass jede rationale Zahl im Intervall $(0,1)$ unendlich oft als Folgenglied vorkommt. Zu jeder Zahl aus der Menge $\mathbb{Q} \cap (0,1)$ gibt es also eine Teilfolge, die konstant nur diesen Wert annimmt. Tatsächlich ist sogar jede Zahl aus dem Intervall $[0,1]$ Häufungspunkt dieser Folge. (Ist Ihnen klar, warum das so ist?)

Lösung 823: Eine simple Rechenaufgabe:

$$g(a) = f(a) - (a-a) \cdot \frac{f(b)-f(a)}{b-a} = f(a)$$
$$g(b) = f(b) - (b-a) \cdot \frac{f(b)-f(a)}{b-a} = f(b) - (f(b)-f(a)) = f(a)$$

Die Werte selbst sind gar nicht so wichtig. Entscheidend ist, dass $g(a)$ und $g(b)$ identisch sind.

Lösung 824: Mit den Ableitungsregeln aus dem letzten Kapitel erhält man sofort:

$$g'(x) = f'(x) - \frac{f(b)-f(a)}{b-a}$$

Lösung 825: Wenn f und g die beiden Funktionen mit $f' = g'$ sind, dann ist $f-g$ differenzierbar und $(f-g)' = f' - g'$ die Nullfunktion. Also ist $f-g$ konstant.

Lösung 826: Das sieht man z.B. so:

```
from scipy.integrate import quad
from plot import *

def f (x):
    return -x + 2 if x <=2 else x * x
def F (x):
    return quad(f, 1, x)[0]
plotFunc2D([f, F], [1, 3], samples=300)
```

f ist nicht stetig. Die „Flächenfunktion“ F hat bei $x = 2$ einen „Knick“ (ist dort also nicht differenzierbar), aber sie ist stetig.

Lösung 827: Wir wissen, dass $-\sin x$ die Ableitung von $\cos x$ ist, also ist $\sin x$ die Ableitung von $-\cos x$ bzw. $-\cos x$ eine Stammfunktion von $\sin x$. Eine weitere Stammfunktion wäre z.B. $\pi - \cos x$.

Lösung 828: Da $(s+1)x^s$ die Ableitung von x^{s+1} ist, müssen wir einfach durch $s+1$ dividieren: $x^{s+1}/(s+1)$ ist eine Stammfunktion von x^s.

Das kann aber offenbar nicht für $s = -1$ gelten, weil man dann durch null teilen müsste. Wir wissen aber schon, dass $1/x$ die Ableitung von $\ln x$ ist. Die Stammfunktion von x^{-1} ist also der natürliche Logarithmus.

Lösung 829: Die Geschwindigkeitsfunktion v ist die Ableitung der Streckenfunktion s. Umgekehrt ist die Streckenfunktion eine Stammfunktion der Geschwindigkeitsfunktion.

Lösung 830: Der Arkustangens ist eine Stammfunktion, siehe Kapitel 46.

Lösung 831: Da e^x Stammfunktion von sich selbst ist, ist die Lösung $e - 1$.

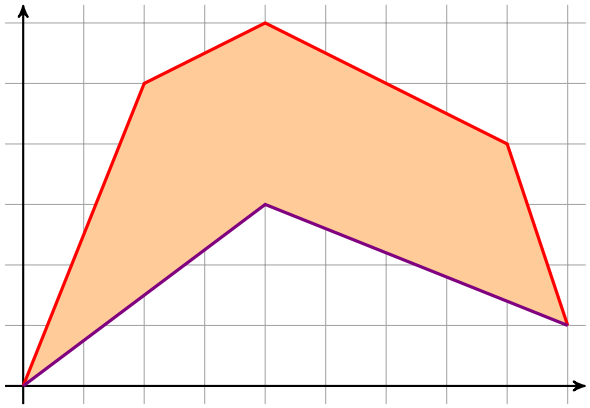
Lösung 832: Wir müssen den folgenden Wert berechnen:

$$\left(\int_{x_1}^{x_2} (g(x) - f(x))\,dx\right) = \int_{-\frac{1}{2}}^{\frac{1}{2}} (-4x^2 + 1)\,dx$$

Eine Stammfunktion von 1 ist x und die Ableitung von $-4x^3/3$ ist $-4x^2$. Die Lösung sieht also so aus:

$$\left[-\frac{4}{3}x^3 + x\right]_{-\frac{1}{2}}^{\frac{1}{2}} = \left(-\frac{4}{3}\cdot\frac{1}{8} + \frac{1}{2}\right) - \left(\frac{4}{3}\cdot\frac{1}{8} - \frac{1}{2}\right) = \frac{2}{3}$$

Lösung 833: Wir legen den linken unteren Punkt auf den Ursprung[84] und beschreiben das Sechseck als die Fläche, die von der roten und der violetten Linie begrenzt wird.



Beide Linien bestehen aus Stücken von Geraden, die man „nur noch“ ausrechnen muss. Etwas Rechenarbeit also, aber eine ganz gute Übung. Hier das Ergebnis:

$$\begin{aligned}
&\left(\int_0^2 \frac{5}{2}x\,dx + \int_2^4 \left(\frac{1}{2}x + 4\right)\,dx + \int_4^8 \left(-\frac{1}{2}x + 8\right)\,dx + \int_8^9 (-3x + 28)\,dx\right) \\
&- \left(\int_0^4 \frac{3}{4}x\,dx + \int_4^9 \left(-\frac{2}{5}x + \frac{23}{5}\right)\,dx\right) \\
&= \left(5 + 11 + 20 + \frac{5}{2}\right) - \left(6 + 10\right) = \frac{77}{2} - 16 = \frac{45}{2}
\end{aligned}$$

[84] Das muss man nicht genauso machen, aber wenn man den Rand der Figur durch Funktionen beschreiben will, muss man sich auf jeden Fall entscheiden, wo das Koordinatensystem liegen soll und welchen Maßstab es haben soll.

Lösung 834: Das macht man so:

```
integrate(sin(x) - sin(x)**2, (x, 0, pi))
```

Die Antwort ist $2 - \pi/2$.

Lösung 836: Ein Term wie `10**(-3)` wird in PYTHON als Fließkommawert berechnet. Und in dem Moment, in dem mindestens ein Argument beim Aufruf einer SYMPY-Funktion ein Fließkommawert ist, erteilen Sie SYMPY quasi die Erlaubnis, nicht mehr exakt zu rechnen.

Man hätte das in diesem Fall verhindern können, indem man `10**(-k)` durch den Ausdruck `Rational(1,10**k)` ersetzt hätte. Allerdings wollten wir hier ja Näherungswerte.

Lösung 838: Das geht so:

```
from sympy import *
from scipy.integrate import quad
import math

x = symbols("x")
sqrt(pi)/2 * erf(2) * 2
integrate(exp(-x**2), (x, -2, 2))
N(sqrt(pi)*erf(2))
quad(lambda x: math.exp(-x**2), -2, 2)
```

Lösung 839: $1/x = x^{-1}$ ist kein Polynom, denn der Exponent ist negativ, also keine natürliche Zahl. Daher ist $x^2 + x^{-2}$ wegen des Exponenten -2 auch kein Polynom. Und das gilt auch für $42\sqrt{x} = 42x^{1/2}$, weil $1/2$ auch kein Element von $\mathbb{N}$ ist. $x \mapsto \sin x$ ist offensichtlich auch kein Polynom.[85]

Wegen des x am Anfang von $x \mapsto 2xy + 4x - 3$ ist y eine Konstante. Daher ist $2xy + 4x - 3 = (2y + 4)x - 3$ ein Polynom.[86] Die anderen Funktionen sind auch alle Polynome. Z.B. kann man $(x + 1) \cdot (x - 1)$ ausmultiplizieren und erhält dann $x^2 - 1$.

Lösung 840: In der Reihenfolge von links nach rechts und von oben nach unten sind die Grade 2, 0, 2, 1 und 6. Die Leitkoeffizienten sind 1, -1, 1, $2y + 4$ und 8. Die Absolutglieder sind -3, -1, -1, -3 und 0.

Lösung 841: $2x$, 0, erneut $2x$, $2y + 4$ und $48x^5$.

[85] Wieso „offensichtlich“? Sie könnten jetzt spitzfindig argumentieren, dass es ja sein könnte, dass man die Sinusfunktion als Polynom darstellen könne. Das wäre ein cleveres Argument! Wir werden allerdings sehen, dass Polynome immer endlich viele Nullstellen haben. Der Sinus hat aber unendlich viele.

[86] Hätten wir $(x, y) \mapsto 2xy + 4x - 3$ geschrieben, dann wäre y eine Variable. Man würde dann von einem Polynom *in zwei Variablen* sprechen, aber das ist nicht Thema dieses Kapitels.

Lösung 842: Wir wissen bereits seit Aufgabe 828, dass $x^{n+1}/(n+1)$ eine Stammfunktion von x^n ist. In diesem Fall ergibt sich also $4/3 \cdot x^3 + 5/2 \cdot x^2 - 7x$.

Lösung 843: In der Reihenfolge der Aufgabenstellung sind die Antworten $-\infty$, ∞ und $-\infty$. Beachten Sie, dass der Betrag des Leitkoeffizienten keine Rolle spielt, sein Vorzeichen aber natürlich schon.

Lösung 844: Die beiden Leitkoeffizienten können sich gegenseitig auslöschen. Addieren Sie z.B. $2x^2 + 3x + 5$ und $-2x^2 + 3x + 5$. Im Allgemeinen gilt also nur:

$$\deg(p+q) \leq \max(\deg(p), \deg(q))$$

Lösung 846: Das korrekte Ergebnis ist 3. Falls Sie das nicht herausbekommen haben, haben Sie hoffentlich nicht den Fehler gemacht, den Koeffizienten 0 vor x^3 zu übersehen! So sollte es aussehen:

	2	0	−1	7	9
↓ +		−2	2	−1	−6
↗ · (−1)	2	−2	1	6	3

Lösung 847: Zum Beispiel so:[87]

```
def Horner (b, P):
    r = 0
    for a in P:
        r = b * r + a
    return r
```

Lösung 848: Im ersten Fall ergibt sich $x^4 - b^4$, im zweiten $x^6 - b^6$.

Lösung 849: Für $k = 0$ ist der Summand $a_0 \cdot (x^0 - b^0) = a_0 \cdot (1 - 1) = 0$.

Lösung 852: Man nehme einfach drei Punkte, die auf einer Gerade liegen, z.B. $(0,0)$, $(1,1)$ und $(2,2)$. Weil diese Punkte alle auf der Geraden $p(x) = x$ liegen, kann es kein weiteres Polynom q mit $\deg(q) \leq 2$ geben, das durch alle drei geht.

Lösung 853: Das ist natürlich nur eine Wiederholung des Stoffs aus Kapitel 25. Die Gerade $p(x) = 3x - 5$ geht durch die beiden Punkte. Ein Polynom kleineren Grades, das durch beide Punkte geht, kann es nicht geben, denn ein Polynom vom Grad 0 ist eine konstante Funktion.

Lösung 854: Nach den bisherigen Überlegungen in diesem Kapitel sollte klar sein, dass das folgende Polynom es tut:

$$(x+1) \cdot (x-2) \cdot (x-3) = x^3 - 4x^2 + x + 6$$

[87] Ich habe den Funktionsnamen vorsichtshalber mit einem Großbuchstaben anfangen lassen, weil SymPy bereits eine Funktion `horner` hat, wie wir noch sehen werden.

Lösung 855: Ich zeige hier nur die Lösung für p_1. Ein Polynom, das durch P_0 und P_2 geht, ist $q_1(x) = (x+1)(x-3)$. Setzt man x_1 ein, so erhält man $q_1(2) = -3$. Daher ergibt sich $p_1(x) = -1/3 \cdot q_1(x)$.

Lösung 856: Das gesuchte Polynom ist $-x^2/2 + x/2 + 4$.

Um zu prüfen, ob Ihre Lösung richtig ist, können Sie die in Kapitel 41 vorgestellte Bibliothek verwenden. Die Funktion `plotFunc2D` akzeptiert als Argumente nicht nur PYTHON-Funktionen, sondern auch Listen von Punkten:

`plotFunc2D`

```
from plot import *

plotFunc2D([lambda x: -x*x/2 + x/2 + 4,
            [(-2, 1), (-1, 3), (2, 3)]], [-2.5, 2.5])
```

Lösung 857: Man erhält diese Gleichungen:

$$\begin{array}{lcrcl} p(-2) & = & 4a - 2b + c & = & 1 \\ p(-1) & = & a - b + c & = & 3 \\ p(2) & = & 4a + 2b + c & = & 3 \end{array}$$

Das ist ein lineares Gleichungssystem für a, b und c und die eindeutige Lösung ist natürlich wie in der letzten Aufgabe $(-1/2, 1/2, 4)$.

Lösung 858: Ich habe es so gemacht:

```
from sympy import *
from plot import *

x = symbols("x")
def f (x):
    return 1 / (1 + 25*x*x)
L = [(x,f(x)) for x in [-1 + 1/5*i for i in range(11)]]
p11 = lambdify(x, interpolate(L, x))
plotFunc2D([f, p11], [-1, 1])
```

Lösung 859: $p(x) = -x^3/32 - 3x^2/4 + 3x/8 + 7/2$.

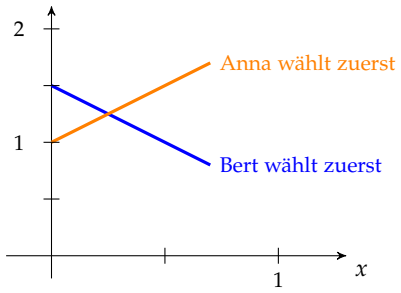
Lösung 860: Wenn Bert beim ersten Kuchen ein Teil auswählt, weiß Anna, dass sie beim zweiten Kuchen zuerst dran ist. Daher kann sie dann den zweiten Kuchen so aufteilen, dass Sie ihn (bis auf einen Krümel) komplett bekommt. Wenn Bert hingegen die erste Wahl Anna überlässt, wird sie den zweiten Kuchen natürlich in zwei genau gleiche Hälften aufteilen, weil bei diesem ja Bert wählen kann und sie ihren Anteil maximieren will.

Wenn wir das kleinere[88] der beiden Teile beim ersten Kuchen mit x bezeichnen, so ist x eine Zahl zwischen 0 und $1/2$ und der Rest ist dann $1 - x$. (Mit anderen Worten: Beide Teile zusammen ergeben 1, d.h. *einen* Kuchen.) Im ersten oben beschriebenen

[88] Wenn beide Teile gleich groß sind, nennen wir einfach eins von beiden x.

Fall (Bert wählt zuerst) bekommt Bert vom ersten Kuchen $1 - x$ (den größeren Teil) und daher vom zweiten Kuchen 0 (also den besagten Krümel). Anna erhält den Rest, d.h. $2 - (1 - x) = x + 1$.

Im zweiten Fall (Anna wählt) bekommt Bert vom ersten Kuchen x und vom zweiten $1/2$, insgesamt also $x + 1/2$, und Anna somit $2 - (x + 1/2) = -x + 3/2$. Annas Anteile $x + 1$ und $-x + 3/2$ sind zwei Geraden, von denen die erste steigt und die zweite fällt. Wenn x einen Wert hat, für den $x + 1$ und $-x + 3/2$ unterschiedlich sind, wird Bert die Strategie bzw. Gerade wählen, bei der der kleinere der beiden Werte herauskommt.



Für Anna ist die optimale Strategie also, dafür zu sorgen, dass Bert gar keine Wahl hat. Das ist der Fall, wenn sich die Geraden schneiden, wenn also $x + 1 = -x + 3/2$ gilt. Wenn man nach x auflöst, erhält man $x = 1/4$. Daher ist die beste Strategie für Anna, vom ersten Kuchen ein Viertel abzutrennen. Wählt Bert dann zuerst (also natürlich drei Viertel des Kuchens), erhält er vom zweiten nur noch einen Krümel. Lässt Bert Anna zuerst wählen, teilt sie den zweiten Kuchen in zwei gleiche Hälften, und Bert erhält ebenfalls insgesamt drei Viertel eines Kuchens. In beiden Fällen hat Anna fünf Viertel bekommen. (Darum hat Anna dieses Spiel wohl auch vorgeschlagen.) (Die optimale Strategie für Bert ist offenbar, zuerst zu wählen, wenn das kleinere Stück weniger als ein Viertel des ersten Kuchens ist. Anderenfalls lässt er Anna den Vortritt. Wir haben aber bereits gesehen, dass das Ergebnis feststeht, wenn beide optimal agieren.)

Lösung 861: Wenn ein Polynom p n-ten Grades eine Nullstelle b_1 hat, dann kann man diese als Linearfaktor abspalten, d.h. man kann p als $p(x) = (x - b_1)\, q(x)$ darstellen, wobei q ein Polynom $n - 1$-ten Grades ist. Jetzt wendet man den Satz auf q an, findet wieder eine Nullstelle, spaltet wieder einen Linearfaktor ab, etc. Am Ende bleibt ein Polynom vom Grad 0 über. Das entspricht dem Faktor a in der Formulierung des letzten Kapitels.

Lösung 862: Ganz einfach: $a^* = a/|a|$. Das entspricht dem Normieren von Vektoren.

Lösung 864: Mit folgendem Code kann man erkennen, dass ab etwa $n = 5$ der Zähler doppelt so groß wie der Nenner ist:

```
from plot import *
from math import sqrt
```

```
num = lambda n: n**4           # die Hälfte des Zählers
den = lambda n: 3*sqrt(2)*n**3 + 2*n**2 + 5*n + sqrt(5)
plotFunc2D([num, den], [0, 6])
```

Lösung 866: Das ist natürlich ein Spline. Zwei Polynome wurden an einer Stelle so „zusammengeklebt“, dass ihre Funktionswerte und Ableitungen dort übereinstimmen.

Lösung 867: Mit Ketten- und Produktregel ergibt sich $2x \sin 1/x - \cos 1/x$.

Lösung 868: Die erste Ableitung f' gibt an, wie sich die Anzahl der Arbeitslosen ändert. Ist f' positiv, so steigt die Zahl an. Wenn dieser Anstieg zurückgeht, so ist damit eigentlich gemeint, dass die zweite Ableitung f'' (also die Änderung des Anstiegs) negativ ist.

Lösung 869: Das ist die Beschleunigung, die Änderungsrate der Geschwindigkeit.

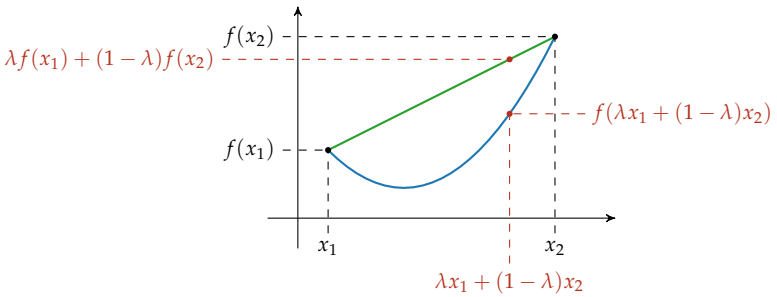
Lösung 870: Die Punkte auf der Sekante von P_1 nach P_2 kann man am einfachsten als Affinkombination (siehe Kapitel 25) der beiden Punkte darstellen. Sie haben also alle die Form

$$\lambda P_1 + (1-\lambda)P_2 = (\lambda x_1 + (1-\lambda)x_2,\ \lambda f(x_1) + (1-\lambda)f(x_2))$$

für $\lambda \in [0,1]$. Nun muss man nur noch die x-Komponente eines solchen Punktes in f einsetzen. Das liefert die gesuchte Ungleichung:

$$f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda)f(x_2)$$

In der folgenden Skizze hat λ in etwa den Wert 0.2.



Lösung 872: Das Polynom sollte so aussehen:

$$p(x) = f(x_0) + f'(x_0)(x-x_0) + \frac{f''(x_0)}{2}(x-x_0)^2 + \frac{f^{(3)}(x_0)}{6}(x-x_0)^3$$

Dabei ist Ihnen sicher aufgefallen, dass die 6 nicht „zufällig“ dort auftaucht, sondern als Produkt von 2 und 3.

Lösung 873: Dafür muss man lediglich die erste Zeile ändern:

```
def taylor (f, x, x0, n):
```

Und man muss `taylor` natürlich nun mit einem weiteren Argument aufrufen, z.B. so:

```
taylor(sqrt(y), y, 1, 2)
```

Lösung 874: Wenn Sie das getan haben, dann haben Sie bemerkt, dass die Koeffizienten für gerade Potenzen alle verschwinden:

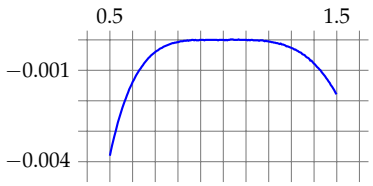
```
[diff(sin(x),x,k).subs(x,0)/factorial(k) for k in range(0,10,2)]
```

Und das war ja auch zu erwarten, weil die geraden Ableitungen des Sinus bis auf das Vorzeichen immer wieder den Sinus ergeben, der bei $x_0 = 0$ den Wert null hat.

Lösung 875: Das ist vergleichsweise simpel. Die zwölfte Ableitung des Sinus – ggf. durch SymPy berechnet – ist wieder der Sinus und der Betrag dieser Funktion wird nirgends größer als eins.[89] Wir haben somit $M = 1$ und $r = \pi$. (51.4) liefert als Abschätzung dann $\pi^{12}/12! \approx 0.00193$.

Lösung 876: Das Restglied sieht so aus:

$$\begin{aligned} R_3 f(x;1) &= f(x) - T_3 f(x;1) \\ &= \sqrt{x} - 1/16 \cdot (x^3 - 5x^2 + 15x + 5) \end{aligned}$$



Der vom Betrag her größte Wert (also der maximale Fehler) liegt offenbar am linken Rand des Intervalls und ergibt sich als:

$$R_3 f(1/2;1) = -91/128 + 1/\sqrt{2} \approx -0.00383$$

Er ist also noch deutlich kleiner als die geschätzte Obergrenze.

Bei einem komplizierteren Verlauf der Restgliedfunktion hätte man an dieser Stelle evtl. noch eine Kurvendiskussion durchführen müssen, um die Extremwerte zu bestimmen.

Lösung 877: Wir schauen uns ein paar Ableitungen von f an:

```
[diff(log(x+1), x, k) for k in range(0, 10)]
```

[89] Wir hätten die Ableitung nicht einmal ermitteln müssen, da bis auf das Vorzeichen ohnehin nur Sinus oder Kosinus infrage kamen und beide immer zwischen -1 und 1 oszillieren.

Durch scharfes Hinsehen kann man erkennen, dass das Vorzeichen alterniert und abgesehen davon im Zähler die Fakultät steht. Der Nenner ist ohnehin offensichtlich. Die n-te Ableitung (ab $n = 1$) hat also diese Form:

$$f^{(n)}(x) = \frac{(-1)^{n+1}(n-1)!}{(x+1)^n}$$

Diese Ableitungen sind im Intervall I alle monoton und werden vom Betrag her nie größer als an der Stelle $x = -0.5$:

```
plotFunc2D([lambdify(x, diff(log(x+1), x, k))
            for k in range(1, 5)], [-0.5, 0.5])
```

Als Abschätzung erhält man mithilfe von (51.4) also:[90]

$$|R_n f(x;0)| \leq \frac{n!}{(-\frac{1}{2}+1)^{n+1}} \cdot \frac{\frac{1}{2^{n+1}}}{(n+1)!} = \frac{1}{n+1}$$

Der Ansatz $\frac{1}{n+1} = 10^{-3}$ führt auf $n = 999$. Man müsste also das Taylorpolynom $T_{999}f(x;0)$ verwenden, wenn man auf der sicheren Seite sein will. (In der Realität ist diese Abschätzung viel zu grob und man kommt mit einem geringeren Grad aus.)

Lösung 878: Allgemein gilt $\ln 2y = \ln 2 + \ln y$ aufgrund der Rechengesetze für den Logarithmus. Ist x also zu groß, so kann man so lange durch 2 dividieren, bis man im Intervall $(0, 2)$ landet. Um z.B. $\ln 10$ zu berechnen, formen Sie so um:

$$\ln 10 = \ln 2 + \ln 5 = 2\ln 2 + \ln 2.5 = 3\ln 2 + \ln 1.25$$

$\ln 1.25$ kann Ihr Computer berechnen, den konstanten Wert $\ln 2$ haben Sie irgendwo gespeichert, und der Rest ist nur noch simple Multiplikation und Addition.

Lösung 879: Das ist einfach auszurechnen, da alle Ableitungen gleich sind und der Funktionswert an der Stelle $x_0 = 0$ immer 1 ist. Die ersten Taylorpolynome sind:

$$\begin{aligned}
T_0 \exp(x;0) &= 1 \\
T_1 \exp(x;0) &= 1 + x \\
T_2 \exp(x;0) &= 1 + x + x^2/2 \\
T_3 \exp(x;0) &= 1 + x + x^2/2 + x^3/6
\end{aligned}$$

Und das sind auch die ersten Partialsummen der Reihe, über die die Exponentialfunktion definiert ist. Dass das kein Zufall ist, sehen wir im nächsten Abschnitt.

Lösung 880: In Kapitel 44 wurde die Exponentialfunktion als Potenzreihe definiert.

Lösung 881: Nein. Wenn man nur den Konvergenzradius r kennt und dieser weder 0 noch ∞ ist, dann weiß man noch nichts über die Werte b, für die $|b - x_0| = r$ gilt. Das sind die Zahlen auf dem Rand des Konvergenzkreises. Das Verhalten auf diesem Rand unterscheidet sich von Reihe zu Reihe und man muss es ggf. mit speziellen Methoden untersuchen.

[90] Das kann man sich auch von SymPy berechnen lassen.

Lösung 882: Der Entwicklungspunkt dieser Reihe ist $x_0 = 0$. Wir wissen aus Kapitel 43, dass sie für $|x| < 1$ konvergiert und für $|x| > 1$ divergiert. Daher ist ihr Konvergenzradius 1.

Lösung 883: Die Koeffizienten sind die Kehrwerte der Koeffizienten der Exponentialreihe, die wir schon als Beispiel hatten. Die Folge $(|a_n/a_{n+1}|)$ ist daher die harmonische Folge $(1/(n+1))$, wodurch sich als Konvergenzradius natürlich null ergibt.

Lösung 884: Das ist ganz einfach, weil $|a_n/a_{n+1}|$ immer 1 ist. Daher konvergiert die Reihe im Intervall $(-1,1)$ um den Entwicklungspunkt 0 herum.

Lösung 885: Man muss nur so wie eben weitermachen:

$$\begin{aligned}
P'(x) &= \sum_{n=1}^{\infty} n a_n (x-x_0)^{n-1} \\
P''(x) &= \sum_{n=2}^{\infty} n(n-1) a_n (x-x_0)^{n-2} \\
P^{(3)}(x) &= \sum_{n=3}^{\infty} n(n-1)(n-2) a_n (x-x_0)^{n-3} \\
P^{(3)}(x_0) &= \sum_{n=3}^{3} n(n-1)(n-2) a_n (x-x_0)^{n-3} = 6a_3
\end{aligned}$$

Allgemein ergibt sich durch k-faches Ableiten: $P^{(k)}(x_0) = k! a_k$.

Lösung 886: Auf jeden Fall die Maclaurinsche Reihe für die Exponentialfunktion, die schon mehrfach in diesem Kapitel angesprochen wurde. Diese Taylorreihe ist schlicht und einfach die Reihe, über die die Exponentialfunktion definiert wurde.

Lösung 887: Der Tangens ist 1, wenn Ankathete und Gegenkathete gleich lang sind. Das bedeutet, dass es sich um ein gleichschenkliges rechtwinkliges Dreieck handeln muss. Also kann die Antwort nur $\pi/4 = 45°$ sein.

Lösung 888: Die Koeffizienten erhält man so:

```
from sympy import *
x = symbols("x")

[diff(atan(x),x,k).subs(x,0)/factorial(k) for k in range(0,20)]
```

Es ist nun offensichtlich, dass die Reihe so aussehen muss:

$$\arctan x = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{2n+1} = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \dots$$

Setzt man für x den Wert 1 ein, so ergibt sich:

$$\pi = 4 \arctan 1 = 4 \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} = 4 \cdot \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots\right)$$

Siehe zum Vergleich auch Programmierprojekt P10. (Dass man auf denselben Zusammenhang auch ganz anders kommen kann, zeigt das verlinkte Video.)

Lösung 890: Da am Ende noch mit vier multipliziert wird, muss der Fehler, den man beim Abbrechen der Reihe macht, kleiner als $\varepsilon/4$ sein. Der Kehrwert dieser Zahl ist 8 000 und wir haben uns gerade überlegt, dass wir auf der sicheren Seite sind, wenn wir bis zum Reihenglied $-1/7999$ addieren.

Lösung 891: Das geht so (siehe Aufgabe 23):

```
from math import log, ceil

ceil(log(leibniz(7999).numerator, 10))
```

Mit `numerator` kommt man an den Zähler eines Bruchs.

`numerator`

Lösung 892: Wenn wir an deutlich mehr als drei Stellen nach dem Komma interessiert sind, besteht die Gefahr, dass es bereits bei der Rechnung Fehler gibt, z.B. könnte es zur *Absorption* (siehe Kapitel 12) kommen. Aber selbst dann, wenn das kein Problem wäre, könnten wir natürlich auf diesem Wege nie mehr signifikante Stellen berechnen, als man mit Fließkommazahlen darstellen kann.

Lösung 894: Ich dachte an sowas:

```
def helper ():
    L = []
    def addValue (v):
        L.append(v)
    def mean ():
        if L:
            return sum(L) / len(L)
    return addValue, mean

a, m = helper()
for k in range(20):
    a(k)
m()
```

Lösung 895: Das hier klappt z.B. nicht:

```
c = convert(10, 3, third())
[next(c) for i in range(20)]
```

Vielleicht überlegen Sie mal selbst, woran es scheitert.

Lösung 896: Man kann das z.B. so machen:

```
def piprod (n):
    num, den, prod, f = 1, 3, 2, 1
    while den <= n:
```

```
        f *= Fraction(num, den)
        prod = prod + f * 2
        num += 1
        den += 2
    return prod
```

Die Konvergenzgeschwindigkeit ist deutlich höher. Die ersten drei Nachkommastellen erreicht man bereits mit `piprod(21)`.

Lösung 897: Wenn man sich nicht verrechnet, sieht es so aus:

$$\begin{aligned}\left[\begin{smallmatrix}a & b\\ c & d\end{smallmatrix}\right]\left(\left[\begin{smallmatrix}p & q\\ r & s\end{smallmatrix}\right](x)\right) &= \frac{a\cdot\left[\begin{smallmatrix}p & q\\ r & s\end{smallmatrix}\right](x)+b}{c\cdot\left[\begin{smallmatrix}p & q\\ r & s\end{smallmatrix}\right](x)+d} = \frac{a\cdot\frac{px+q}{rx+s}+b}{c\cdot\frac{px+q}{rx+s}+d}\\ &= \frac{\frac{apx+aq}{rx+s}+b}{\frac{cpx+cq}{rx+s}+d} = \frac{\frac{apx+aq}{rx+s}+\frac{brx+bs}{rx+s}}{\frac{cpx+cq}{rx+s}+\frac{drx+ds}{rx+s}}\\ &= \frac{apx+aq+brx+bs}{cpx+cq+drx+ds}\\ &= \frac{(ap+br)x+(aq+bs)}{(cp+dr)x+(cq+ds)} = \left[\begin{smallmatrix}ap+br & aq+bs\\ cp+dr & cq+ds\end{smallmatrix}\right](x)\end{aligned}$$

Die Komposition der beiden Abbildungen entspricht also dem Produkt der Matrizen, wie Sie sicher bemerkt haben.

Lösung 898: Man muss eigentlich nur darauf achten, die Drei vor dem Komma zu „entsorgen“, ansonsten ist quasi nichts zu tun:

```
p = Pi()
next(p)
c = convert(10, 2, p)
[next(c) for i in range(42)]
```

Lösung 899: Man könnte z.B. auf die Matrizen verzichten. Die sind zum Erklären ganz praktisch, aber eigentlich unnötig. Zudem erspart man sich damit den unnötigen Aufwand, die Null in der unteren linken Ecke jeder Matrix mitzuschleppen. Außerdem kann man ab und zu die entstandenen Brüche kürzen. $\left[\begin{smallmatrix}12 & -14\\ 0 & 21\end{smallmatrix}\right]$ hat z.B. denselben Effekt wie $\left[\begin{smallmatrix}60 & -70\\ 0 & 105\end{smallmatrix}\right]$. Ebenso kann man die Komponenten der Matrizen $M_1, M_2, \ldots$ erst bei Bedarf berechnen und muss dafür nur wissen, bei welchem Index man gerade ist. Schließlich sind auch die Closures ein schönes didaktisches Mittel, erzeugen aber einen gewissen Overhead, den man vermeiden kann. Eine „abgespeckte“ Version des Programms könnte so aussehen:

```
from math import gcd

def Pi():
    V1, V2, V4, num, den, prod = 1, 0, 1, 1, 3, 3
    while True:
```

```
        if 4*V1+V2-V4 < prod*V4:
            yield prod
            V1, V2, prod = 10*V1, 10*(V2-prod*V4), \
                           (10*(3*V1+V2))//V4-10*prod
        else:
            V1, V2, V4, prod = num*V1, den*(2*V1+V2), den*V4, \
                               (7*num*V1+2+den*V2)//(den*V4)
            num += 1
            den += 2
        if num % 50 == 0:
            g = gcd(V1, gcd(V2, V4))
            V1, V2, V4 = V1//g, V2//g, V4//g
```

`V1`, `V2` und `V4` repräsentieren die Matrix V, `num` und `den` sind Zähler und Nenner der Brüche $1/3, 2/5, 3/7, \ldots$ und `prod` ist die nächste zu produzierende Ziffer. Die letzten drei Zeilen sorgen dafür, dass ab und zu gekürzt wird.

Lösung 900: Trotz des Kürzens werden die beteiligten Zahlen immer größer. (Probieren Sie es aus: Lassen Sie sich nach dem Kürzen jeweils `V1`, `V2` und `V4` ausgeben.) Python muss also Langzahlarithmetik (siehe Seite 40) einsetzen und das dauert natürlich umso länger, je größer die Zahlen sind.

Kann man es prinzipiell vermeiden, dass bei einer solchen Berechnung der Speicherbedarf immer größer wird? Nein, das geht nicht! Jedes Programm, das nur eine begrenzte Menge an Speicher verwendet, muss zwangsläufig irgendwann mal in einen Zustand kommen, in dem es schon einmal war. Erzeugt so ein Programm die Nachkommastellen einer Zahl, so müssen sich diese also ab einem bestimmten Punkt periodisch wiederholen. Dann würde es sich aber um eine rationale Zahl handeln – siehe Kapitel 11.

Lösung 901: Sie kann Ihnen doch helfen, weil Sie mithilfe dieser Formel ja jede Stelle unabhängig von den anderen berechnen können. Daher können Sie die Arbeit der Berechnung aller 100 000 Stellen auf mehrere Prozessoren bzw. Computer verteilen und damit potentiell viel Zeit sparen.

Lösung 902: Das ist kein Widerspruch. Zunächst mal haben wir die Reihe nicht einfach nur umsortiert, sondern eine neue Reihe konstruiert, weil ja auch noch die Vorfaktoren dazukamen. Dass diese neue Reihe gegen den „alten“ Wert konvergiert, muss man natürlich beweisen. (Und ich habe zumindest eine Skizze des Beweises vorgeführt.)

Außerdem besagt der Umordnungssatz, dass man durch geschicktes Umordnen jeden Reihenwert erreichen kann. Er besagt *nicht*, dass *jede* Umordnung automatisch zu einem anderen Reihenwert führt.

Lösung 903: Zum Beispiel so:

```
from canvas import Canvas
from cmath import exp, pi

c = Canvas()              # in einer eigenen Zelle

c.clear()
c.drawAxes()
x = 0
step = 2 * pi / 20
while x <= 2 * pi:
    C = exp(x * 1j)
    c.drawPoint((C.real, C.imag))
    x += step
```

Lösung 904: Die Funktion `f` berechnet hier eigentlich nur $\exp(\mathrm{i}x)$. Allerdings gibt sie dann passend zum Zeichnen das Ergebnis als Tripel zurück, wobei der berechnete Funktionswert in Real- und Imaginärteil aufgespalten wurde. Die erste Zeile ist notwendig, wenn Sie die erzeugte Grafik drehen und von allen Seiten betrachten wollen.

```
%matplotlib notebook
from plot import *
from cmath import exp, pi

def f (x):
    z = exp(complex(0, x))
    return [x, z.real, z.imag]

plotCurve3D(f, [-2*pi, 2*pi])
```

Lösung 905: Zum Beispiel so:

```
from cmath import exp
from plot import *

plotComplexAbs(exp, [-2, 2])
```

Lösung 906: Das ergibt natürlich die Kosinuskurve, allerdings mit der maximalen Amplitude $\mathrm{e}^2 \approx 7.4$ statt 1.

```
from cmath import exp, pi
from plot import *

plotFunc2D(lambda x: exp(complex(2, x)).real, [-2* pi, 2*pi])
```

Lösung 907: Mit den bekannten Rechenregeln ist das ganz einfach:

$$\exp(\mathrm{i}x)^{42} = (\mathrm{e}^{\mathrm{i}\pi/14})^{42} = \mathrm{e}^{3\pi\mathrm{i}} = \mathrm{e}^{2\pi\mathrm{i}} \cdot \mathrm{e}^{\pi\mathrm{i}} = 1 \cdot (-1) = -1$$

Lösung 908: Nach der Eulerformel gilt $\mathrm{i} = \mathrm{e}^{\mathrm{i}\pi/2}$. Damit folgt:

$$\mathrm{i}^{\mathrm{i}} = \mathrm{e}^{\mathrm{i}\pi/2\cdot\mathrm{i}} = \mathrm{e}^{-\pi/2} \in \mathbb{R}$$

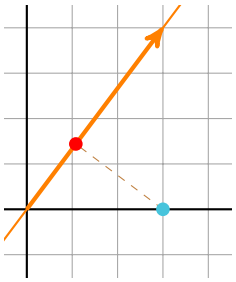
Eine Potenz, bei der sowohl Basis als auch Exponent imaginär sind, kann also durchaus einen reellen Wert haben.

Lösung 909: Wenn man $\cos(-x) = \cos x$ und $\sin(-x) = -\sin x$ beachtet, ergibt sich:

$$\begin{aligned}\mathrm{e}^{\mathrm{i}x} + \mathrm{e}^{-\mathrm{i}x} &= (\cos x + \mathrm{i}\sin x) + (\cos(-x) + \mathrm{i}\sin(-x)) \\ &= \cos x + \mathrm{i}\sin x + \cos x - \mathrm{i}\sin x = 2\cos x\end{aligned}$$

Lösung 910: Zunächst müssen wir den Richtungsvektor $\boldsymbol{v}$ normieren: Wir berechnen $\|\boldsymbol{v}\| = \sqrt{3^2 + 4^2} = 5$ und ersetzen $\boldsymbol{v}$ daher durch $\boldsymbol{v}_1 = 1/5 \cdot \boldsymbol{v}$. Die Projektion ergibt sich dann als

$$\boldsymbol{p}' = \left\langle \begin{pmatrix}3\\0\end{pmatrix}, \begin{pmatrix}3/5\\4/5\end{pmatrix} \right\rangle \cdot \begin{pmatrix}3/5\\4/5\end{pmatrix} = \frac{9}{5} \cdot \begin{pmatrix}3/5\\4/5\end{pmatrix} = \begin{pmatrix}27/25\\36/25\end{pmatrix} = \begin{pmatrix}1.08\\1.44\end{pmatrix}$$



Lösung 911: Wenn man ausnutzt, dass das Skalarprodukt von $\lambda\boldsymbol{v}$ und $\boldsymbol{p}' - \boldsymbol{p}$ null ist, erhält man:

$$\begin{aligned}\|\boldsymbol{q} - \boldsymbol{p}\|^2 &= \|\boldsymbol{p}' + \lambda\boldsymbol{v} - \boldsymbol{p}\|^2 = \|(\boldsymbol{p}' - \boldsymbol{p}) + \lambda\boldsymbol{v}\|^2 \\ &= \langle(\boldsymbol{p}' - \boldsymbol{p}) + \lambda\boldsymbol{v}, (\boldsymbol{p}' - \boldsymbol{p}) + \lambda\boldsymbol{v}\rangle \\ &= \langle\boldsymbol{p}' - \boldsymbol{p}, \boldsymbol{p}' - \boldsymbol{p}\rangle + 2\langle\lambda\boldsymbol{v}, \boldsymbol{p}' - \boldsymbol{p}\rangle + \langle\lambda\boldsymbol{v}, \lambda\boldsymbol{v}\rangle \\ &= \|\boldsymbol{p}' - \boldsymbol{p}\|^2 + \lambda^2\|\boldsymbol{v}\|^2 = \|\boldsymbol{p}' - \boldsymbol{p}\|^2 + \lambda^2\end{aligned}$$

Nach Ziehen der Wurzel ergibt sich:

$$\|\boldsymbol{q} - \boldsymbol{p}\| = \sqrt{\|\boldsymbol{p}' - \boldsymbol{p}\|^2 + \lambda^2}$$

Da λ nicht null ist, ist $\|\boldsymbol{q} - \boldsymbol{p}\|$ also auf jeden Fall größer als $\|\boldsymbol{p}' - \boldsymbol{p}\|$.

Lösung 912: Das ist reine Rechenarbeit:

$$\boldsymbol{p}' = \left\langle \begin{pmatrix}1\\1\\3\end{pmatrix}, \begin{pmatrix}1\\0\\0\end{pmatrix} \right\rangle \cdot \begin{pmatrix}1\\0\\0\end{pmatrix} + \left\langle \begin{pmatrix}1\\1\\3\end{pmatrix}, \begin{pmatrix}3/5\\4/5\\0\end{pmatrix} \right\rangle \cdot \begin{pmatrix}3/5\\4/5\\0\end{pmatrix}$$

$$= 1 \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + \frac{7}{5} \cdot \begin{pmatrix} 3/5 \\ 4/5 \\ 0 \end{pmatrix} = \begin{pmatrix} 46/25 \\ 28/25 \\ 0 \end{pmatrix}$$

Lösung 913: Es klappt:

$$\begin{aligned} \boldsymbol{p'} &= \left\langle \begin{pmatrix} 1 \\ 1 \\ 3 \end{pmatrix}, \begin{pmatrix} 4/5 \\ -3/5 \\ 0 \end{pmatrix} \right\rangle \cdot \begin{pmatrix} 4/5 \\ -3/5 \\ 0 \end{pmatrix} + \left\langle \begin{pmatrix} 1 \\ 1 \\ 3 \end{pmatrix}, \begin{pmatrix} 3/5 \\ 4/5 \\ 0 \end{pmatrix} \right\rangle \cdot \begin{pmatrix} 3/5 \\ 4/5 \\ 0 \end{pmatrix} \\ &= \frac{1}{5} \cdot \begin{pmatrix} 4/5 \\ -3/5 \\ 0 \end{pmatrix} + \frac{7}{5} \cdot \begin{pmatrix} 3/5 \\ 4/5 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \end{aligned}$$

Lösung 914: Für eine auf $[-\pi, \pi]$ definierte Funktion f erhalten wir:

$$\|f\| = \sqrt{\langle f, f \rangle} = \frac{1}{\sqrt{\pi}} \left(\int_{-\pi}^{\pi} f^2(x)\, \mathrm{d}x \right)^{\frac{1}{2}}$$

Lösung 915: Zunächst basteln wir uns eine Liste von Basisfunktionen in einer für SymPy geeigneten Form:

```
from sympy import *
x = symbols("x")

L = [cos(k * x) for k in range(4)] + [sin(k * x) for k in range(1,4)]
```

Dann definieren wir das Skalarprodukt:

```
dotProd = lambda f, g: integrate(1/pi * f * g, (x, -pi, pi))
```

Jetzt überprüfen wir, ob die Basisfunktionen paarweise orthogonal sind:

```
[dotProd(L[i], g) for i in range(len(L)) for g in L[i+1:]]
```

Und schließlich: Sind alle Basisfunktionen normiert?

```
norm = lambda f: sqrt(dotProd(f, f))
[norm(f) for f in L]
```

An der Stelle hakt es (ein bisschen): die erste Basisfunktion hat die Norm $\sqrt{2}$ und nicht 1.

Lösung 916: Wie Sie gesehen haben, hat SymPy aus dem Produkt zweier trigonometrischer Funktionen eine Summe einzelner Funktionen gemacht. Die ist wesentlich leichter zu integrieren. Unter der URL

https://github.com/sympy/sympy/blob/master/sympy/simplify/fu.py

finden Sie eine Beschreibung des FU-Moduls, das übrigens nach einem chinesischen Mathematiker benannt wurde. Dieses Modul stellt verschiedene Strategien zur Vereinfachung trigonometrischer Ausdrücke bereit, die man jedoch zum jetzigen Zeitpunkt manuell anstoßen muss.

Lösung 917: Man muss die Funktionswerte außerhalb des Intervalls so verschieben, dass sie im Intervall landen. Zum Beispiel folgendermaßen:

```
def fS (x):
    d = ceil((x - pi) / (2 * pi))
    return f(x - 2 * d * pi)

plotFunc2D([fS, fourier(f, 2)], [-7*pi, 7*pi], samples=200)
```

Lösung 918: Man kann die Aufgabe von a_0 an diesem Beispiel erkennen:

```
g = lambda x: f(x) + 42
plotFunc2D([g, fourier(g, 0)], [-pi, pi])
```

Nun haben wir $a_0 = 84$. (Denken Sie daran, dass dieser Wert ja noch durch zwei dividiert wird.) Wenn Sie sich die Fourierkoeffizienten für höhere Fourierpolynome ausgeben lassen und dabei f und g vergleichen, werden Sie sehen, dass es – bis auf Rundungsfehler – nur im Koeffizienten a_0 Unterschiede gibt. Die „0-te Schwingung“ $x \mapsto \cos 0x = 1$ ist also für das vertikale Verschieben verantwortlich.

Man kann es auch so formulieren: Nach der Formel für die Koeffizienten gilt

$$\frac{a_0}{2} = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(x)\,\mathrm{d}x$$

und damit ist $a_0/2$ das arithmetische Mittel der Funktionswerte von f im Intervall $[-\pi, \pi]$, wenn man das Integral als kontinuierliche Summe interpretiert.

Lösung 919: Zum Beispiel so:

```
f = lambda x: (x/pi) - floor(x/pi)
plotFunc2D([f, fourier(f, 30)], [-5, 5], samples=400)
```

Lösung 920: Wie üblich gibt es viele Wege, die Funktionen zu definieren. Ich habe es ähnlich wie in Aufgabe 917 gemacht:

```
def f (x):
    d = ceil((x - pi/2) / pi)
    x = x - d * pi
    if x >= 0:
        return 4*x/pi - 1
    else:
        return -4*x/pi - 1
```

```
def g (x):
    d = ceil((x - pi/2) / pi)
    return 1 if x - d * pi >= 0 else -1
```

Lösung 921: Die erste Funktion ist stetig und aus Geradenstücken, die stetig differenzierbar sind, zusammengesetzt. Daher werden die Fourierpolynome gleichmäßig, d.h. ohne Überschwinger, gegen f konvergieren. Die zweite Funktion hat Unstetigkeitsstellen. An diesen Sprüngen gibt es Überschwinger.

Lösung 923: Das sieht so aus:

$$\begin{aligned} a\mathrm{e}^{\mathrm{i}kx} &= a\cos kx + \mathrm{i}a\sin kx \\ a\mathrm{e}^{-\mathrm{i}kx} &= a\cos(-kx) - \mathrm{i}a\sin(-kx) = a\cos kx - \mathrm{i}a\sin kx \end{aligned}$$

Die Summe der beiden Ausdrücke ist $2a\cos kx$, weil die Sinusterme wegfallen. Die Differenz ist $2a\mathrm{i}\sin kx$.

Lösung 924: Um etwas Schreibarbeit zu sparen, werden die Terme vor dem Addieren mit 2 multipliziert:

$$\begin{aligned} 2c\mathrm{e}^{\mathrm{i}kx} &= (a - b\mathrm{i})(\cos kx + \mathrm{i}\sin kx) = a\cos kx - \mathrm{i}b\cos kx + \mathrm{i}a\sin kx + b\sin kx \\ 2\overline{c}\mathrm{e}^{-\mathrm{i}kx} &= (a + b\mathrm{i})(\cos kx - \mathrm{i}\sin kx) = a\cos kx + \mathrm{i}b\cos kx - \mathrm{i}a\sin kx + b\sin kx \end{aligned}$$

Nun wird addiert. Die farblich markierten Summanden neutralisieren sich:

$$2(c\mathrm{e}^{\mathrm{i}kx} + \overline{c}\mathrm{e}^{-\mathrm{i}kx}) = 2a\cos kx + 2b\sin kx$$

Zum Schluss dividiert man den Faktor 2 wieder weg:

$$c\mathrm{e}^{\mathrm{i}kx} + \overline{c}\mathrm{e}^{-\mathrm{i}kx} = a\cos kx + b\sin kx$$

Lösung 925: Damit a_0 reell ist, muss zunächst c_0 reell sein. Damit die anderen a_k reell sind, muss ihr Imaginärteil, also $\Im(c_k + c_{-k})$ verschwinden. Das bedeutet, dass $\Im(c_k) = -\Im(c_{-k})$ gelten muss. Damit die b_k auch reell sind, muss $\Im(\mathrm{i}(c_k - c_{-k})) = \Re(c_k - c_{-k})$ verschwinden.[91] Das impliziert, dass $\Re(c_k) = \Re(c_{-k})$ gelten muss. Die letzten beiden Bedingungen zusammen besagen, dass wir $c_k = \overline{c_{-k}}$ für alle $k \geq 1$ brauchen.

Lösung 926: Hier sollte man sich erinnern, dass $\Re(\overline{z}) = \Re(z)$ und $\Im(\overline{z}) = -\Im(z)$ für jede komplexe Zahl z gilt. Es folgt:

$$\begin{aligned} \int_a^b \overline{f(x)}\,\mathrm{d}x &= \int_a^b \Re(\overline{f(x)})\,\mathrm{d}x + \mathrm{i}\int_a^b \Im(\overline{f(x)})\,\mathrm{d}x \\ &= \int_a^b \Re(f(x))\,\mathrm{d}x - \mathrm{i}\int_a^b \Im(f(x))\,\mathrm{d}x \\ &= \overline{\int_a^b f(x)\,\mathrm{d}x} \end{aligned}$$

Wenn man also die konjugierte Funktion integriert, so wird einfach der Wert des Integrals konjugiert.

[91] Allgemein gilt $\Im(\mathrm{i}z) = \Re(z)$ für jede komplexe Zahl z. Rechnen Sie es nach!

Lösung 927: Nach unseren obigen Überlegungen muss $c_0 = \overline{c_0}$ gelten. Das bedeutet, dass c_0 auf jeden Fall reell ist.

Lösung 929: Das folgt aus den Rechenregeln für die Exponentialfunktion:

$$\left(\mathrm{e}^{\frac{2k\mathrm{i}\pi}{n}}\right)^n = \mathrm{e}^{\frac{2k\mathrm{i}\pi}{n}\cdot n} = \mathrm{e}^{2k\mathrm{i}\pi} = \left(\mathrm{e}^{2\mathrm{i}\pi}\right)^k = 1^k = 1$$

Lösung 930: Das ergibt ω_n^{-1}, wie man ja auch schon grafisch sieht. Und allgemein ist offenbar $\overline{\omega_n^k} = \omega_n^{-k}$.

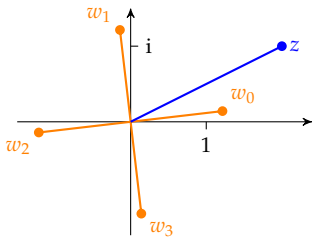
Lösung 931: ω_6 ist primitiv. Allgemein ist ω_n immer eine primitive n-te Einheitswurzel. 1 ist nicht primitiv; auch das gilt für alle n. Für $n = 6$ ist außerdem ω_6^5 primitiv, alle anderen Einheitswurzeln sind es nicht. Allgemein ist ω_n^k genau dann primitiv, wenn k und n teilerfremd sind.

Lösung 932: Wenn man $\omega_n^{k_1}$ und $\omega_n^{k_2}$ multipliziert, dann ist das Ergebnis ω_n^k, wobei $k_1 + k_2 \equiv k \pmod{n}$ gilt. Und der gleiche Zusammenhang besteht zwischen Division und modularer Subtraktion.

Lösung 933: Die Formel ist $\sqrt[n]{|z|}\,\mathrm{e}^{\mathrm{i}\arg(z)/n + 2k\mathrm{i}\pi/n}$ für $k = 0, \ldots, n-1$, denn:

$$\left(\sqrt[n]{|z|}\,\mathrm{e}^{\frac{\mathrm{i}\arg(z)+2k\mathrm{i}\pi}{n}}\right)^n = |z|\,\mathrm{e}^{\mathrm{i}\arg(z)+2k\mathrm{i}\pi} = |z|\,\mathrm{e}^{\mathrm{i}\arg(z)} = z$$

Hier z.B. die vier vierten Wurzeln von $z = 2 + \mathrm{i}$.



Der Betrag von z ist $\sqrt{5}$, der Betrag von w_0 daher $\sqrt[4]{\sqrt{5}}$. Das Argument von w_0 ist ein Viertel des Arguments von z. Die drei anderen Wurzeln entstehen, indem man w_0 um Vielfache von 90° (ein Viertel des Gesamtkreises) dreht.

Lösung 934: Wenn ich selbst nicht durcheinander gekommen bin, sieht es so aus:

$$\frac{1}{7}\cdot\begin{pmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1\\
1 & \omega_7^{-1} & \omega_7^{-2} & \omega_7^{-3} & \omega_7^{-4} & \omega_7^{-5} & \omega_7^{-6}\\
1 & \omega_7^{-2} & \omega_7^{-4} & \omega_7^{-6} & \omega_7^{-8} & \omega_7^{-10} & \omega_7^{-12}\\
1 & \omega_7^{-3} & \omega_7^{-6} & \omega_7^{-9} & \omega_7^{-12} & \omega_7^{-15} & \omega_7^{-18}\\
1 & \omega_7^{-4} & \omega_7^{-8} & \omega_7^{-12} & \omega_7^{-16} & \omega_7^{-20} & \omega_7^{-24}\\
1 & \omega_7^{-5} & \omega_7^{-10} & \omega_7^{-15} & \omega_7^{-20} & \omega_7^{-25} & \omega_7^{-30}\\
1 & \omega_7^{-6} & \omega_7^{-12} & \omega_7^{-18} & \omega_7^{-24} & \omega_7^{-30} & \omega_7^{-36}
\end{pmatrix}\cdot\begin{pmatrix}
y_0\\ \omega_7^3 y_1\\ \omega_7^6 y_2\\ \omega_7^9 y_3\\ \omega_7^{12} y_4\\ \omega_7^{15} y_5\\ \omega_7^{18} y_6
\end{pmatrix} = \begin{pmatrix}
-c_{-3}\\ c_{-2}\\ -c_{-1}\\ c_0\\ -c_1\\ c_2\\ -c_3
\end{pmatrix}$$

$$\frac{1}{6} \cdot \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \omega_6^{-1} & \omega_6^{-2} & \omega_6^{-3} & \omega_6^{-4} & \omega_6^{-5} \\ 1 & \omega_6^{-2} & \omega_6^{-4} & \omega_6^{-6} & \omega_6^{-8} & \omega_6^{-10} \\ 1 & \omega_6^{-3} & \omega_6^{-6} & \omega_6^{-9} & \omega_6^{-12} & \omega_6^{-15} \\ 1 & \omega_6^{-4} & \omega_6^{-8} & \omega_6^{-12} & \omega_6^{-16} & \omega_6^{-20} \\ 1 & \omega_6^{-5} & \omega_6^{-10} & \omega_6^{-15} & \omega_6^{-20} & \omega_6^{-25} \end{pmatrix} \cdot \begin{pmatrix} y_0 \\ \omega_6^2 y_1 \\ \omega_6^4 y_2 \\ \omega_6^6 y_3 \\ \omega_6^8 y_4 \\ \omega_6^{10} y_5 \end{pmatrix} = \begin{pmatrix} c_{-2} \\ -c_{-1} \\ c_0 \\ -c_1 \\ c_2 \\ -c_3 \end{pmatrix}$$

Überprüfen Sie bitte im Detail, ob in Ihrer Lösung auch wirklich alle Vorzeichen und Indizes korrekt sind!

Lösung 935: Für alle j gilt dann $\overline{y_j} = y_j$. Es folgt:[92]

$$\begin{aligned} \overline{c_{-2}} &= \overline{y_0 + \omega_5^2 y_1 + \omega_5^4 y_2 + \omega_5^6 y_3 + \omega_5^8 y_4} \\ &= \overline{y_0} + \overline{\omega_5^2} \cdot \overline{y_1} + \overline{\omega_5^4} \cdot \overline{y_2} + \overline{\omega_5^6} \cdot \overline{y_3} + \overline{\omega_5^8} \cdot \overline{y_4} \\ &= y_0 + \omega_5^{-2} y_1 + \omega_5^{-4} y_2 + \omega_5^{-6} y_3 + \omega_5^{-8} y_4 = c_2 \end{aligned}$$

Und ganz allgemein rechnet man leicht nach, dass unter diesen Voraussetzungen immer $\overline{c_k} = c_{-k}$ gilt. Das ist derselbe Zusammenhang wie am Ende des vorletzten Kapitels.

Lösung 936: `start` ist der Index (ohne das Minus davor) des ersten Koeffizienten, der berechnet wird. Für $n = 5$ beginnen wir z.B. mit c_{-2}, für $n = 6$ auch. In beiden Fällen hat `start` also den Wert 2. Diese Variable wird benötigt, um die Werte in der Liste `Y` mit den richtigen Potenzen von ω_n zu multiplizieren und um am Ende die Vorzeichen der berechneten Koeffizienten zu korrigieren.

Lösung 937: In SymPy würde es z.B. so gehen:

```
from sympy import *

omega = exp(2 * I * pi / 5)
M = Matrix([[omega**(-k*j) for j in range(5)] for k in range(5)])
N = Matrix([[omega**(k*j) for j in range(5)] for k in range(5)])
simplify(M * N)
```

Ohne SymPy könnte man es folgendermaßen machen:

```
from matrices import Matrix
from math import sin, cos, pi

arg = 2 * pi / 5
omega = complex(cos(arg), sin(arg))
M = Matrix([[omega**(-k*j) for j in range(5)] for k in range(5)])
N = Matrix([[omega**(k*j) for j in range(5)] for k in range(5)])
M * N
```

[92] Siehe auch Aufgabe 930.

In beiden Fällen ergibt sich, dass $\boldsymbol{M} \cdot \boldsymbol{N}$ das Fünffache der Einheitsmatrix ist.[93] Mit anderen Worten: $1/5 \cdot \boldsymbol{N}$ ist die inverse Matrix zu $\boldsymbol{M}$ bzw. $\boldsymbol{N}$ die Inverse zu $1/5 \cdot \boldsymbol{M}$. Allgemein lässt sich eine Matrix vom Typ (55.4) immer so einfach invertieren.

Lösung 938: Es gilt:

$$\mathrm{e}^{\mathrm{i}x_j} = \mathrm{e}^{\mathrm{i}(-\pi+2\pi j/5)} = \mathrm{e}^{-\mathrm{i}\pi}(\mathrm{e}^{2\mathrm{i}\pi/5})^j = -\omega_5^j$$

Aus der Forderung $y_j \stackrel{!}{=} p(x_j)$ ergibt sich dann:

$$\begin{aligned} y_j = p(x_j) &= c_{-2}\mathrm{e}^{-2\mathrm{i}x_j} + c_{-1}\mathrm{e}^{-\mathrm{i}x_j} + c_0 + c_1\mathrm{e}^{\mathrm{i}x_j} + c_2\mathrm{e}^{2\mathrm{i}x_j} \\ &= c_{-2}(-\omega_5^j)^{-2} + c_{-1}(-\omega_5^j)^{-1} + c_0 + c_1(-\omega_5^j) + c_2(-\omega_5^j)^2 \\ &= c_{-2}\omega_5^{-2j} - c_{-1}\omega_5^{-j} + c_0 - c_1\omega_5^j + c_2\omega_5^{2j} \end{aligned}$$

Und in Matrixschreibweise sieht es folgendermaßen aus:

$$\begin{pmatrix} 1 & -1 & 1 & -1 & 1 \\ \omega_5^{-2} & -\omega_5^{-1} & 1 & -\omega_5^1 & \omega_5^2 \\ \omega_5^{-4} & -\omega_5^{-2} & 1 & -\omega_5^2 & \omega_5^4 \\ \omega_5^{-6} & -\omega_5^{-3} & 1 & -\omega_5^3 & \omega_5^6 \\ \omega_5^{-8} & -\omega_5^{-4} & 1 & -\omega_5^4 & \omega_5^8 \end{pmatrix} \cdot \begin{pmatrix} c_{-2} \\ c_{-1} \\ c_0 \\ c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix}$$

Das ist eine völlig ausreichende Antwort für diese Aufgabe. Man kann aber auch hier noch etwas Arbeit investieren, um das Ergebnis ansehnlicher zu machen.

Im ersten Schritt werden wir die störenden Vorzeichen los:

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ \omega_5^{-2} & \omega_5^{-1} & 1 & \omega_5^1 & \omega_5^2 \\ \omega_5^{-4} & \omega_5^{-2} & 1 & \omega_5^2 & \omega_5^4 \\ \omega_5^{-6} & \omega_5^{-3} & 1 & \omega_5^3 & \omega_5^6 \\ \omega_5^{-8} & \omega_5^{-4} & 1 & \omega_5^4 & \omega_5^8 \end{pmatrix} \cdot \begin{pmatrix} c_{-2} \\ -c_{-1} \\ c_0 \\ -c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix}$$

Jetzt multiplizieren wir die zweite Zeile mit ω_5^2, die dritte mit ω_5^4 und so weiter:

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & \omega_5^1 & \omega_5^2 & \omega_5^3 & \omega_5^4 \\ 1 & \omega_5^2 & \omega_5^4 & \omega_5^6 & \omega_5^8 \\ 1 & \omega_5^3 & \omega_5^6 & \omega_5^9 & \omega_5^{12} \\ 1 & \omega_5^4 & \omega_5^8 & \omega_5^{12} & \omega_5^{16} \end{pmatrix} \cdot \begin{pmatrix} c_{-2} \\ -c_{-1} \\ c_0 \\ -c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} y_0 \\ \omega_5^2 y_1 \\ \omega_5^4 y_2 \\ \omega_5^6 y_3 \\ \omega_5^8 y_4 \end{pmatrix}$$

Diese Darstellung sollte Ihnen nun schon recht vertraut vorkommen. Und tatsächlich ist die obige Matrix die Inverse zu der Matrix aus Gleichung (55.4). Siehe Aufgabe 937.

Lösung 939: Die Summe $\sum_{k=0}^{5} \omega_6^k$ wird zunächst mal irgendeinen komplexen Wert z haben. Was passiert nun, wenn wir die Summe mit ω_6 multiplizieren, wenn wir also geometrisch gesehen alles drehen? Einerseits bekommen wir dann (nach Umsortieren) wieder die Summe aller sechsten Einheitswurzeln, also wieder z. Andererseits kommt offenbar $\omega_6 z$ heraus. $\omega_6 z = z$ kann aber nur stimmen, wenn $z = 0$ gilt.

[93] Wenn man es auch im zweiten Fall wegen der Rundungsfehler kaum sehen kann…

Lösung 941: Das ist ω_n^{-1}. Offensichtlich ist diese Zahl eine n-te Einheitswurzel und ihre $n/2$-te Potenz ist -1.

Lösung 942: Damit wir immer weiter durch zwei teilen können. Nachdem man die Problemgröße n halbiert hat, hat man es wieder mit einer Zweierpotenz zu tun. (Man kann die Grundidee der schnellen Fouriertransformation auch für andere n anwenden; dann wird es aber komplizierter und die Zeitersparnis ist nicht ganz so groß.)

Lösung 945: Wenn der Grad n ist, haben p und q jeweils $n+1$ Koeffizienten. Zunächst werden $(n+1)\cdot(n+1) = n^2+2n+1$ Multiplikationen durchgeführt. Die Anzahl der benötigten Additionen kann man sich z.B. so klarmachen: Jeder Eintrag in der $(n+1)\times(n+1)$-Matrix der Produkte wird zu dem Eintrag addiert, der diagonal links unter ihm steht. Die Ausnahmen bilden dabei die $n+1$ Einträge, die ganz links stehen, und die n Einträge, die ganz unten stehen und nicht schon als „ganz links“ gezählt wurden. Das ergibt

$$(n+1)\cdot(n+1) - (n+1) - n = n^2$$

Additionen. Insgesamt haben wir $2n^2+2n+1$, also $\mathcal{O}(n^2)$, Operationen.

Lösung 946: Nein. Was ist denn, wenn η dummerweise null oder eins ist? $\eta = -1$ wäre auch nicht gut. In allen drei Fällen hätten Sie nicht genügend Angaben und es gäbe unendlich viele Möglichkeiten für die a_i.

Lösung 947: Wie oben soll n größer als der Grad des Produktes pq sein. Wir führen drei schnelle Fouriertransformationen durch. Die haben aber dieselbe Ordnung wie eine, nämlich $\mathcal{O}(n\log n)$. Schritt (iii) ist offenbar $\mathcal{O}(n)$, so dass wir insgesamt $\mathcal{O}(n\log n)$ haben.

Allerdings sind wir bisher nicht vom Grad der Faktoren ausgegangen. Ist der m, so hat pq den Grad $2m$. Wenn wir Pech haben, ist die kleinste Zweierpotenz, die größer als $2m$ ist, die Zahl $4m$. Zum Glück ist aber $4m\log 4m = 4m(\log 4 + \log m)$ und das ist wieder $\mathcal{O}(m\log m)$.

Lösung 949: Die Zahl 2 erfüllt zwar die genannten Voraussetzungen für eine schnelle Fouriertransformation, aber 17 ist für unsere Zwecke nicht groß genug. Wenn man zwei Dezimalziffern multipliziert, kann deren Produkt maximal 81 sein, also muss der Restklassenring, in dem man arbeitet, mindestens 82 Elemente haben.

Lösung 950: Das sind die Zahlen 64, 659, 3438 und 4033.

Lösung 951: So zum Beispiel:

```
from matrices import Matrix
from vectors import Vector

M = Matrix([[12**(i*k) % 233 for i in range(8)]
            for k in range(8)])
A = M * Vector([1, 3, 4, 5, 0, 0, 0, 0])
B = M * Vector([2, 4, 8, 7, 0, 0, 0, 0])
```

```
C = [a*b % 233 for a, b in zip(A, B)]
12**7 % 233                          # ergibt 136
for i in range(233):
    if (i * 8) % 233 == 1:
        break
i                                    # ergibt 204, Kehrwert von 8
Minv = 204 * Matrix([[136**(i*k) % 233 for i in range(8)]
                     for k in range(8)])
D = Minv * Vector(C)
D = [d % 233 for d in D]
D
sum(d * 10**k for k, d in enumerate(D))
```

Im letzten Schritt erhalten wir die Folge der Zahlen, die wir aus (55.10) kennen. Also haben wir alles richtig gemacht.

Ein paar Dinge, die vielleicht nicht so ganz offensichtlich waren, will ich noch kurz erläutern:

- Den Kehrwert von ν bekommt man ganz einfach durch ν^7, weil ν eine achte Einheitswurzel ist.
- Sie erinnern sich hoffentlich noch, dass man für die inverse Matrix $\boldsymbol{M}^{-1}$ den Vorfaktor $1/8$ braucht.
- Den braucht man aber in $\mathbb{Z}_{233}$! Ich berechne diesen Wert hier einfach durch schlichtes Probieren. Wir wissen aus Kapitel 6 ja noch, wie man das machen müsste, wenn man es mit wesentlich größeren Zahlen als 233 zu tun hätte.
- Die Summe am Ende sollte man natürlich eigentlich mit dem Horner-Schema ausrechnen.
- Zwischendurch stehen in `A`, `B` und `D` vergleichsweise große Zahlen. Ein effizienter Algorithmus würde natürlich die ganze Zeit modular rechnen. (Ein effizienter Algorithmus würde aber auch die Matrizenmultiplikation mithilfe der schnellen Fouriertransformation durchführen.)

Lösung 953: Wenn wir davon ausgehen, dass der Kaffee (hoffentlich) wärmer als die Zimmertemperatur T_U ist, dann wird $T_K(t) - T_U$ positiv sein. Da sich der Kaffee abkühlt und nicht erhitzt, muss die Ableitung natürlich negativ sein.

Lösung 954: Nein. Es stimmt zwar, lässt sich aber aus dem, was wir bisher wissen, nicht schließen. Es könnte ja z.B. sein, dass wir durch geduldiges Raten und Ausprobieren noch andere Lösungen finden. Wir werden in diesem Kapitel aber noch präzise mathematische Aussagen über die Existenz und die Eindeutigkeit von Lösungen von Differentialgleichungen kennenlernen.

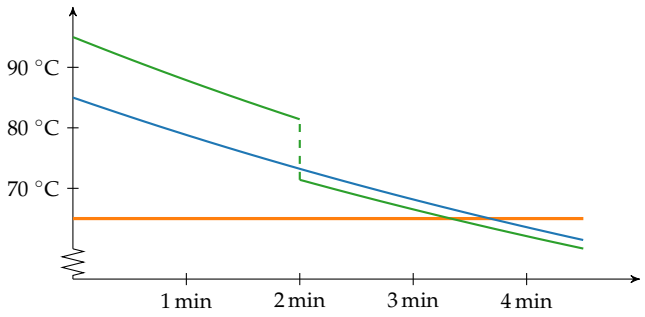
Lösung 955: Wir lassen die Einheiten ab jetzt weg, nachdem wir uns auf Minuten und Grad Celsius geeignet haben. Wir haben $T_0 = 95$ und $T_U = 20$ sowie $T_K(30) = 25$, also:

$$25 = T_K(30) = (95 - 20) \cdot \mathrm{e}^{c \cdot 30} + 20$$

Das kann man nun nach c auflösen und erhält $c = (-\ln 15)/30$. Für unsere weiteren Untersuchungen reicht uns $c \approx -0.1$ als Näherungswert.

Lösung 956: Hier ist meine Lösung. Die Funktion `Temp` liefert eine PYTHON-Funktion für den Temperaturverlauf mit der Starttemperatur `T0`.

```
from math import exp
from plot import *

TU = 20      # Zimmertemperatur
c = -0.1     # experimentell ermittelte Abkühlungskonstante
d = 10       # Temperaturverlust durch Zucker

def Temp (T0):
    return lambda t: (T0 - TU) * exp(c * t) + TU

# Temperaturverlauf bei sofortiger Zuckerzugabe
f = Temp(95 - d)
# Temperaturverlauf ohne Zuckerzugabe
g1 = Temp(95)
# Temperaturverlauf mit einer Starttemperatur,
# die sich aus einer Zuckerzugabe nach zwei Minuten ergibt
g2 = Temp(g1(2) - d)
# Kombination der beiden Verläufe
g = lambda t: g1(t) if t <= 2 else g2(t - 2)

# Vergleich mit Trinktemperatur
plotFunc2D([f, g, lambda t: 65], [0, 5], samples=400)
```



Wenn man am schnellen Abkühlen des Kaffees interessiert ist, ist es also günstiger, den Zucker erst später hinzuzufügen. Das liegt daran, dass die Temperaturkurve umso schneller fällt, je größer die Temperaturdifferenz zur Umgebung ist. Mit anderen Worten: Heißer Kaffee kühlt schneller ab.

Gibt man den Zucker sofort in den Kaffee (blaue Kurve), so dauert es etwa drei Minuten und 40 Sekunden, bis der Kaffee auf 65 Grad abgekühlt ist. Ohne Zucker

braucht der Kaffee nur ca. drei Minuten und sechs Sekunden, bis er bei 75 Grad ist. Fügen Sie dann den Zucker hinzu und trinken Sie im selben Moment, so haben Sie mehr als eine halbe Minute gespart. Morgen früh kommen Sie also garantiert rechtzeitig zur Vorlesung!

Lösung 957: Man muss in diesem Fall *zwei* Dinge wissen. Erstens braucht man natürlich die Information, wo sich der Kopf der Feder zum entsprechenden Zeitpunkt befunden hat. Das wäre $x(0)$, wenn wir den Zeitpunkt als $t = 0$ bezeichnen, und entspräche unserer Anfangsbedingung beim Kaffeeproblem. Wir müssen aber auch wissen, ob und mit welcher Geschwindigkeit sich der Kopf bewegt hat, d.h. wir brauchen auch $\dot{x}(0)$. Wir werden gleich sehen, dass es für eine Differentialgleichung zweiter Ordnung typisch ist, dass man zwei Anfangsbedingungen braucht.

Lösung 958: Das sollte hoffentlich nicht so schwer sein:

$$\begin{aligned}
F(x_1, x_2, x_3) &= x_3 - cx_2 + cT_U \\
F(x_1, x_2, x_3, x_4) &= mx_4 + cx_3 + kx_2 \\
F(x_1, \ldots, x_7) &= 42x_7 + 23x_5 - x_3 \\
F(x_1, \ldots, x_5) &= \exp(x_5 - x_2) + (5x_2 - 3)(x_3^2 + 3x_2) \\
F(x_1, x_2, x_3, x_4) &= \sin(x_4) + \frac{x_3}{3x_2 + 5} \\
F(x_1, x_2, x_3, x_4) &= x_4(x_2 + x_1) - \frac{x_1}{x_3}
\end{aligned}$$

Lösung 959: $F(x_1, x_2, x_3) = x_3 - f(x_1)$

Lösung 960: Ja. Beim Lösen der Differentialgleichung mussten wir lediglich integrieren; und wir wissen, dass Stammfunktionen bis auf die additiven Konstanten eindeutig bestimmt sind.

Lösung 961: Setzen wir $x = 2$ in (56.7) ein, so erhalten wir $y'(2) = 6 + 8 + C$. $y'(2)$ kann also nur dann den Wert 2 haben, wenn $C = -12$ gilt. Setzen wir nun $x = 2$ sowie $C = -12$ in die allgemeine Lösung ein, so ergibt sich $y(2) = 4 + 8 - 24 + D$. Damit $y(2) = 30$ gilt, muss D den Wert 42 haben. Die gesuchte spezielle Lösung ist folglich $y(x) = x^3/2 + 2x^2 - 12x + 42$.

Lösung 962: Gleichung (56.6).

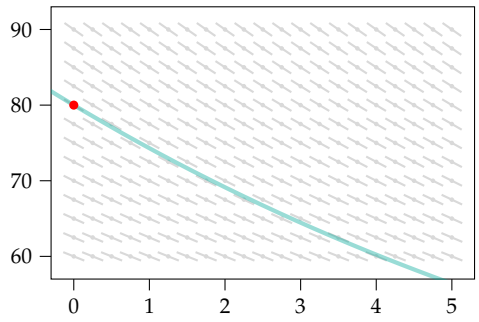
Lösung 963: Das sollte so aussehen:

$$\begin{aligned}
y' &= cy - cT_U \\
y'' &= (-cy' - ky)/m \\
y^{(5)} &= (-23y^{(3)} + y')/42 \\
y' &= 3x(3y + 5) \\
y' &= \ln(-y)
\end{aligned}$$

Da man bei der letzten Umformung den Logarithmus auf die Funktion $-y$ anwendet, muss man sich vorher darüber Gedanken machen, ob es möglich ist, dass y nichtnegative Werte annimmt; dann wäre der Ausdruck rechts vom Gleichheitszeichen nämlich

nicht definiert. In der ursprünglichen Gleichung $\exp(y') + y = 0$ ist der erste Summand aber garantiert immer positiv, also muss jede Lösung vollständig unterhalb der x-Achse verlaufen. In diesem Fall ist die Umformung also in Ordnung.

Lösung 964: Das bedeutet einfach, dass man einen bestimmten Punkt (x_0, y_0) markiert und der Lösung damit vorschreibt, dass sie durch diesen Punkt gehen muss.



Lösung 966: Jede Funktion der Form

$$x \mapsto \begin{cases} -(x-a)^2 & x \leq a \\ 0 & a \leq x < b \\ (x-b)^2 & x \geq b \end{cases}$$

ist eine Lösung, wenn $a \leq 0 \leq b$ gilt. Wählt man Werte a und b mit $a \neq -b$, so hat man eine Funktion, die nicht ungerade ist.

Lösung 967: Das geht genau wie vorher. Sie müssen nur die erste Zeile ändern.

```
res = dsolve(10*diff(y(x),x,2) + 12*diff(y(x),x) + 3*y(x))
```

Lösung 968: Da das immer nach demselben Muster abläuft, wird nur für das erste Problem ein ausführlicher Lösungsweg vorgeführt.

Zunächst lassen wir SymPy eine allgemeine Lösung berechnen:

```
from sympy import *

x = symbols("x")
y = Function("y")

res = dsolve(diff(y(x), x) - 2*x*y(x))
res
```

Daran sehen wir schon, dass die Lösung von der Form $c_1 e^{x^2}$ ist. Wir müssen nur noch nach c_1 auflösen:

```
solve(res.rhs.subs(x, 0) - 1)
```

Das liefert uns $c_1 = 1$ und damit die spezielle Lösung $y(x) = e^{x^2}$. Jetzt noch die Probe:

```
f = lambda x: exp(x**2)
f(0), 2*x*f(x), diff(f(x), x)
```

Die weiteren Lösungen sehen so aus:

$$\begin{aligned} y(x) &= -2/(2x+1) \\ y(x) &= -2 + 3e^x - 2x - x^2 \\ y(x) &= e^x - x \\ y(x) &= e^x - x - x^2/2 \\ y(x) &= \sin x \\ y(x) &= (x^3+2)/3 \end{aligned}$$

Lösung 972: Der Wert ist 5. Das kann der Computer für uns berechnen:

```
p = lambda x: 2*x**3 + 5*x**2 + 3*x + 2
p(3) % 7
```

Lösung 974: Wenn Sie es nicht selbst ausrechnen wollen, lassen Sie den Computer arbeiten:

```
from sympy import *

x = symbols("x")
p = Poly(x**2 + 1, modulus = 7)
[p(k) for k in range(7)]
```

Das (vielleicht) überraschende Ergebnis ist, dass da niemals null rauskommt. Widerspricht das nicht dem Fundamentalsatz der Algebra? Nein. Denn der sagt nur etwas über Polynome mit *komplexen* Koeffizienten aus; es handelt sich nicht um einen Satz über beliebige Polynome über irgendwelchen Körpern. Wir sehen an diesem Beispiel also, dass es durchaus möglich ist, dass ein Polynom zweiten Grades keine Nullstellen hat. Es gilt aber wie gesagt nach wie vor, dass so ein Polynom *höchstens* zwei Nullstellen hat.

Lösung 975: Die Nullstellen sind 2 und 4. Der Sinn dieser Aufgabe war, dass Sie nacheinander die Werte 1, 2, 3 in p einsetzen sollten.[94] Insbesondere sollten Sie nach dem Finden der zweiten Nullstelle aufhören und nicht auch noch $p(5)$ und $p(6)$ ausrechnen. Siehe vorherige Aufgabe.

Lösung 976: Nein! Das ist ein häufiges Missverständnis. Die Exponenten geben an, wie oft x mit sich selbst multipliziert wird. Es sind ganz normale natürliche Zahlen und keine Elemente des jeweiligen Körpers. x^3 steht z.B. *immer* für $x \cdot x \cdot x$, egal ob man in $\mathbb{R}$, in $\mathbb{Z}_3$ oder in $\mathbb{Z}_{97}$ rechnet.

[94] Dass man es nicht mit 0 zu probieren braucht, haben Sie sicher gesehen.

Lösung 977: Mit Computerhilfe:

```
from sympy import *

x = symbols("x")
p = Poly(x**3 + x + 1, modulus = 3)
q = Poly(2*x**5 + 1, modulus = 3)
[(p(k), q(k)) for k in range(3)]
```

Wir sehen, dass die Werte jeweils übereinstimmen.

Lösung 978: Eine Funktion f von $\mathbb{Z}_3$ nach $\mathbb{Z}_3$ ist eindeutig durch die Angabe der drei Funktionswerte $f(0)$, $f(1)$ und $f(2)$ festgelegt. Für jeden dieser Werte gibt es nur die drei Möglichkeiten 0, 1 und 2. Also kann es insgesamt nur $3^3 = 27$ verschiedene Funktionen geben.

Lösung 980: Ja. Sowohl 5 als auch -5 sind nun größte gemeinsame Teiler.

Lösung 981: Nur die Zahlen 1 und -1. Der Kehrwert der Eins ist eins, der Kehrwert von -1 ist -1.

Lösung 982: Alle Polynome, deren Grad null ist, also alle konstanten Polynome außer dem Nullpolynom.

`degree`

Lösung 983: Die entscheidende Änderung ist, dass es beim Vergleichen jetzt um den Grad der Polynome geht. In SymPy erhält man den mit der Funktion `degree`. Außerdem verwenden wir natürlich nun `quo` und `rem` statt `//` und `%`. Man kann es dann z.B. so machen:

```
from sympy import *

def gcdLcm (a, b, m, var = x):
    if degree(a) < degree(b):
        a, b = b, a
    c0, c = Poly(1, var, modulus=m), Poly(0, var, modulus=m)
    d0, d = Poly(0, var, modulus=m), Poly(1, var, modulus=m)
    p = 1
    while degree(b) >= 0:
        f = quo(a, b, modulus=m)
        a, b = b, rem(a, b, modulus=m)
        d0, d = d, f * d + d0
        c0, c = c, f * c + c0
        p *= -1
    return (a, c0 * Poly(p, var, modulus=m),
               d0 * Poly(-p, var, modulus=m), c, d)
```

Lösung 984: Ich habe dafür die Funktion `ktup` aus Aufgabe 301 verwendet:

```
from sympy import *

x = symbols("x")
L = []
for coeffs in ktup(list(range(3)), 6):
    if coeffs[4]:
        p = Poly(sum(c*x**k for k, c in enumerate(coeffs)),
                 x, modulus=3)
        if p.is_irreducible:
            L.append(p)
L
```

Lösung 985: Als Bitfolgen haben wir $p = \texttt{1101}$, $q = \texttt{10}$ und $r = \texttt{1000}$ sowie für die Produkte $pq = \texttt{11010}$ und $pr = \texttt{1101000}$. Offensichtlich lässt sich die Multiplikation mit einem Monom einfach als Verschiebung nach links um die entsprechende Stellenzahl realisieren.

Lösung 986: Die beiden Faktoren müssen ebenfalls Monome gewesen sein. Schauen Sie sich z.B. das folgende Produkt an:

$$(x^7 + \cdots + x^2)(x^3 + \cdots + x) = x^{10} + \cdots + x^3$$

x^{10} ergab sich hier als Produkt von x^7 und x^3, x^3 als Produkt von x^2 und x. Allgemein haben wir im Produkt eine Summe von mindestens zwei Monomen, wenn mindestens einer der Faktoren aus mehr als einem Monom besteht. (Obwohl es uns im Moment nur für $\mathbb{Z}_2$ interessiert, gilt diese Aussage für beliebige Körper, weil grundsätzlich das Produkt von zwei Koeffizienten, die beide nicht verschwinden, niemals null sein kann.)

Lösung 987: Nein. Für die mathematische Begründung muss ein Bündelfehler einfach nur von der Form $e = \texttt{1...10000}$ sein. Ob es zwischen dem ersten und dem letzten Fehler noch weitere gibt, ist irrelevant. Wie groß der Abstand zwischen Anfang und Ende des Bündels maximal sein darf, regelt der Grad von g.

Lösung 988: Wenn Sie die Zahlen 2, 1 und 7 als $(2,2,1,1,7,7)$ verschicken und es einen einzigen Fehler gibt, so könnte beim Empfänger z.B. $(2,5,1,1,7,7)$ ankommen. War die erste Zahl nun eine Zwei oder eine Fünf? Sie sehen: das bringt nichts.

Lösung 989: Nein. Wenn statt $(2,2,2,2,1,1,1,1,7,7,7,7)$ zum Beispiel die Zahlenfolge $(2,5,2,5,1,1,1,1,7,7,7,7)$ ankommt, ist nach wie vor nicht klar, was die erste Zahl sein soll. Man müsste für das verlässliche Abfangen von zwei Fehlern mit diesem Verfahren schon jede Zahl fünfmal senden.

Lösung 990: Man verwendet ein Polynom dritten Grades und sendet insgesamt acht Punkte, die auf dem Graphen des Polynoms liegen. Wenn zwei falsch sind, können die im schlimmsten Fall zusammen mit drei korrekt übertragenen Punkten eine konkurrierende Kurve bilden. Aber die richtige Kurve würde mit sechs zu fünf Punkten gewinnen.

Lösung 991: Man würde nach wie vor ein Polynom zweiten Grades nehmen, aber neun statt sieben Punkte versenden. Drei falsche Punkte könnten im Extremfall zusammen mit zwei richtigen auf einer gemeinsamen (falschen) Parabel liegen. Das wären fünf Punkte. Aber auf der richtigen Parabel lägen sechs Punkte.

Lösung 992: Wenn drei oder vier Punkte falsch sind, können die zusammen mit zwei richtigen eine andere Parabel bilden. Auf der liegen dann insgesamt fünf oder sechs Punkte, aber *nicht* alle sieben. Der Empfänger könnte also auf jeden Fall sicher sein, dass es Übertragungsfehler gegeben hat. Er wüsste nur nicht, wie viele Fehler (ein, zwei, drei oder vier) es waren. Erst bei mindestens fünf Fehlern könnte es theoretisch passieren, dass *alle* sieben Punkte auf einer Parabel liegen, die nicht die ursprünglich gesendete ist.

Es ist typisch für Reed-Solomon-Codes (und nicht nur für die), dass sie mehr Fehler *erkennen* als *korrigieren* können.

Lösung 995: Weil 12 der Kehrwert von 12 in $\mathbb{Z}_{13}$ ist. Allgemein gilt in jedem Körper $(-1)^2 = 1$, so dass wir die Matrix $\boldsymbol{M}$ immer nur mit -1 multiplizieren müssen.

Lösung 996: Das war so gemeint:

```
gcdLcm(n, r, 13)[4]
gcdLcm(n, r + Poly(x**5, modulus=13), 13)[4]
gcdLcm(n, r + Poly(4*x**5 + 2*x**3, modulus=13), 13)[4]
# etc.
```

Lösung 999: Das waren die *irreduziblen* Polynome, siehe Kapitel 57.

Lösung 1000: Man beginnt wie eben mit $\mathbb{Z}_2$ und nimmt als irreduzibles Polynom in diesem Fall $x^2 + x + 1$.

Lösung 1001: n ist der Grad des irreduziblen Polynoms, mit dem man arbeitet. Zum Glück findet man zu jedem n so ein Polynom. (Und meistens findet man mehrere desselben Grades; siehe Aufgabe 984. Die erzeugen dann aber isomorphe Körper.) Für $n = 1$ kann man das Polynom x verwenden und erhält offenbar $\mathrm{GF}(p) = \mathbb{Z}_p$.

Lösung 1002: Zunächst erstellen wir das Polynom und testen vorsichtshalber noch mal, ob es wirklich irreduzibel ist.

```
from sympy import *

x = symbols("x")
g = Poly(x**2 + x + 2, modulus=3)
g.is_irreducible
```

Dann erzeugen wir eine Liste der Körperelemente:

```
GF = [Poly(k*x + m, x, modulus=3) for k in [0,1,2] for m in [0,1,2]]
```

Außerdem habe ich aus kosmetischen Gründen noch eine kleine Hilfsfunktion geschrieben, die den Koeffizienten -1 in eine Zwei umwandelt.

```
conv = lambda x: str(2 if x == -1 else x)
```

Dann bekommen wir so die Additionstabelle:

```
A = [[p+q for p in GF] for q in GF]
[["".join(map(conv, p.all_coeffs())) for p in r] for r in A]
```

Und analog die für die Multiplikation:

```
M = [[rem(p*q, g) for p in GF] for q in GF]
[["".join(map(conv, p.all_coeffs())) for p in r] for r in M]
```

Lösung 1003: Wir fangen an wie in Aufgabe 1002. Die eigentliche Lösung dieser Aufgabe besteht lediglich aus der letzten Zeile.

```
from sympy import *

x = symbols("x")
g = Poly(x**2 + x + 2, modulus=3)
GF = [Poly(k*x + m, x, modulus=3) for k in [0,1,2] for m in [0,1,2]]
[p for p in GF if len(set(rem(p**k, g) for k in range(8))) == 8]
```

Lösung 1004: Da würde sich eine Menge wie $\Omega = \{\text{Bild}, \text{Zahl}\}$ anbieten. Man kann aber auch abstrahieren und einfach sowas wie $\Omega = \{0, 1\}$ nehmen.

Lösung 1005: Ich würde eine Menge wie $\Omega = \{0,1\}^{20}$ nehmen. Das Tupel

$$(0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0) \in \Omega$$

würde dann z.B. bedeuten, dass nur beim dritten und siebten Mal der Toast nicht auf der Butterseite gelandet ist. Ist man ausschließlich an der Anzahl der „Butterlandungen“ interessiert, so würde allerdings auch $\Omega = \{0, 1, 2, \ldots, 20\}$ als Ergebnismenge ausreichen.

Lösung 1006: Ich würde $\mathbb{N}^+$ nehmen. Dabei bedeutet das Ergebnis $n \in \mathbb{N}^+$, dass nach n Würfen zum ersten Mal „Zahl“ oben landet. Das Ereignis aus der Aufgabenstellung wäre $\{1,2,3,4\}^{\complement}$.

Lösung 1007: Um zur Vereinigung mehrerer Mengen zu gehören, reicht es, zu einer dieser Mengen zu gehören. Daher sieht die Antwort so aus:

$$\begin{aligned}\bigcup_{k=1}^{n} A_k &= A_1 \cup A_2 \cup \cdots \cup A_n \\ &= \{x : x \in A_k \text{ für mindestens ein } k \in \{1, \ldots, n\}\}\end{aligned}$$

Lösung 1008: Wenn man es ausführlich aufschreibt, so erhält man:

$$\bigcap_{n=2}^{4} A_{2n} = A_4 \cap A_6 \cap A_8$$

Es geht also um die Teiler, die die drei Zahlen 4, 6 und 8 gemeinsam haben. Das sind 1 und 2, also wäre $\{1,2\}$ die korrekte Antwort gewesen.

Lösung 1009: Das kann man in einer Zeile erledigen:

```
from functools import reduce

def union (L):
    return reduce(lambda A, B: A | B, L, set())
```

Lösung 1010: Die leere Vereinigung sollte die leere Menge sein. Das kann man sich z.B. anhand des Programms `union` klarmachen. Für den leeren Durchschnitt gibt es keinen sinnvollen Wert. Das müsste dann sowas wie „die Menge aller Objekte“ sein, aber diese Menge gibt es nicht.[95]

Lösung 1011: In diesem Durchschnitt sind die Zahlen enthalten, die Teiler von *jeder* positiven natürlichen Zahl n sind. Die einzige Zahl mit dieser Eigenschaft ist die Eins, also ist die Antwort $\{1\}$.

Lösung 1012: $A_1 = A_2 = [3,5]$, $A_3 = [4,5]$, $A_4 = A_6 = \mathbb{N}$, $A_5 = \{0\}$, $A_7 = [0,\infty)$, $A_8 = A_7 \setminus \mathbb{N}$, $A_9 = A_{10} = A_{11} = \varnothing$.

Lösung 1013: So sollte es aussehen:

$$A \setminus \bigcap_{k=1}^{n} B_k = \bigcup_{k=1}^{n} (A \setminus B_k)$$
$$A \setminus \bigcup_{k=1}^{n} B_k = \bigcap_{k=1}^{n} (A \setminus B_k)$$

Die Begründung für die Korrektheit folgt im Anschluss an diese Aufgabe.

Lösung 1015: In der Definition steht, dass eine Menge A dann zu $\mathcal{A}$ gehört, wenn ihr Durchschnitt mit $\{1,2\}$ zwei Elemente hat oder keins; mit anderen Worten: wenn entweder 1 und 2 beide zu A gehören oder beide nicht.

Lösung 1017: Nur im dritten Fall handelt es sich um eine σ-Algebra, wie man leicht überprüfen kann. Im ersten und vierten Fall gehört $\varnothing$ nicht zu $\mathcal{A}$. Im zweiten Fall sind z.B. $\{1,2\}$ und $\{2,3\}$ Elemente von $\mathcal{A}$, ihre Vereinigung ist es aber nicht. Im fünften Fall sind die Singletons $\{0\}$, $\{2\}$, $\{4\}$, $\{6\}$ und so weiter alle Elemente von $\mathcal{A}$, ihre Vereinigung ist jedoch die Menge aller geraden natürlichen Zahlen, die nicht zu $\mathcal{A}$ gehört.

[95] Darauf einzugehen würde hier zu weit führen. Bei Interesse informieren Sie sich bitte über die sogenannte *Russellsche Antinomie*.

Lösung 1018: Man könnte der Einfachheit halber $\Omega = \{1, 2, 3, \ldots, 52\}$ setzen. Ordnet man den vier Assen die Zahlen 1 bis 4 zu, so ergibt sich die gesuchte Wahrscheinlichkeit als

$$P(\{1,2,3,4\}) = \frac{|\{1,2,3,4\}|}{|\Omega|} = \frac{4}{52} = \frac{1}{13} \approx 7.7\%$$

Lösung 1019: In diesem Fall bietet es sich an,

$$\Omega = \{X \subseteq \{1, 2, 3, \ldots, 52\} : |X| = 2\}$$

zu setzen. Aus Kapitel 17 wissen wir, dass diese Menge $\binom{52}{2} = 1326$ Elemente hat. Bezeichnet man die Asse wie in Aufgabe 1018, so sieht das uns interessierende Ereignis so aus:

$$\begin{aligned} A &= \{X \subseteq \{1,2,3,4\} : |X| = 2\} \\ &= \{\{1,2\}, \{1,3\}, \{1,4\}, \{2,3\}, \{2,4\}, \{3,4\}\} \end{aligned}$$

A hat sechs Elemente. $6/1326 = 1/221 \approx 0.5\%$ ist also die gesuchte Wahrscheinlichkeit für zwei Asse.

Beachten Sie, dass in diesem Wahrscheinlichkeitsraum die Ergebnisse selbst Mengen von Zahlen sind, so dass die Ereignisse Mengen von solchen Mengen sind. Das mag etwas verwirrend sein, wenn man es zum ersten Mal sieht.

Man hätte es aber auch anders machen und mit

$$\Omega^* = \{(x_1, x_2) \in \{1, 2, 3, \ldots, 52\}^2 : x_1 \neq x_2\}$$

anfangen können. Ein Paar (x_1, x_2) steht dabei dafür, dass x_1 die erste gezogene Karte ist und x_2 die zweite. Ω^* hat die Mächtigkeit $52 \cdot 51$, weil es hier um Variationen ohne Wiederholungen geht. Das für uns relevante Ereignis würde in diesem Fall aber anders aussehen:

$$\begin{aligned} A^* = \{&(1,2), (1,3), (1,4), (2,3), (2,4), (3,4), \\ &(2,1), (3,1), (4,1), (3,2), (4,2), (4,3)\} \end{aligned}$$

Als Wahrscheinlichkeit ergibt sich $|A^*|/|\Omega^*| = 12/(52 \cdot 51)$ und das ist natürlich auch $1/221$.

Lösung 1020: Wir haben teilweise neue Mengen gebildet und P auf diese angewendet. P ist aber nur für Elemente aus $\mathcal{A}$ definiert. Daher müssen wir uns eigentlich erst versichern, dass die Mengen, über die wir sprechen, auch wirklich Ereignisse sind. Die dritte Eigenschaft kann man z.B. überhaupt nur formulieren, weil mit A auch immer $A^\complement$ zu einer σ-Algebra gehört. Bei der vierten Aussage taucht die Menge $C = B \setminus A$ auf. Dass die zu $\mathcal{A}$ gehört, kann man sich dadurch klarmachen, dass man sie in der Form $C = B \cap A^\complement$ schreibt. $A^\complement$ gehört zu $\mathcal{A}$ und damit auch C als Durchschnitt zweier Elemente von $\mathcal{A}$.

Lösung 1021: Hoffentlich an das Inklusions-Exklusions-Prinzip, das wir in Kapitel 16 besprochen haben.

Lösung 1022: Die beiden Ereignisse sind nicht disjunkt, daher kann man die beiden Wahrscheinlichkeiten nicht einfach addieren. Das ist schon wichtig. Zum Beispiel ist die Wahrscheinlichkeit, *kein* Bild zu ziehen, $1 - 3/13 = 10/13$ und die Wahrscheinlichkeit, *nicht* Herz zu ziehen, beträgt $1 - 1/4 = 3/4$. Wenn man das einfach addieren könnte, dann wäre die Wahrscheinlichkeit für eine Karte, die kein Bild *oder* nicht Herz ist, $10/13 + 3/4 = 53/52$, also größer als eins! Was sollte das bedeuten? Und wäre dann die Wahrscheinlichkeit, eine Karte wie Herz Bube zu ziehen, negativ? Das ist natürlich Unsinn.

Die richtige Antwort auf die Frage liefert die fünfte unserer kleinen Regeln. Ist A das Ereignis „Bild“ und B das Ereignis „Herz“, so wollen wir $P(A \cup B)$ berechnen und müssen daher zusätzlich zu $P(A)$ und $P(B)$ noch $P(A \cap B)$ kennen. Bei $A \cap B$ geht es um Karten, die sowohl die Farbe Herz haben als auch Bilder sind. Davon gibt es drei. Also ergibt sich:

$$\begin{aligned} P(A \cup B) &= P(A) + P(B) - P(A \cap B) \\ &= 12/52 + 13/52 - 3/52 = 22/52 \approx 42\% \end{aligned}$$

Lösung 1023: Wir modellieren das als Laplace-Experiment mit $\Omega = X^3$, wobei X die Menge $\{1,2,3,4,5,6\}$ sein soll. Man könnte sich nun mühsam überlegen, wie viele verschiedene Möglichkeiten es gibt, eine bestimmte Augenzahl k ab fünf zu erreichen, und diese Möglichkeiten dann alle addieren. Zum Beispiel gibt es für fünf Augen die $M_5 = 6$ Möglichkeiten: $(1,1,3)$, $(1,3,1)$, $(3,1,1)$, $(1,2,2)$, $(2,1,2)$ und $(2,2,1)$. Das ergäbe diese Tabelle:

k	5	6	7	8	9	10	11	12	$\cdots$	17	18
M_k	6	10	15	21	25	27	27	25	$\cdots$	3	1

Als Summe ergibt sich 212 und damit als Wahrscheinlichkeit $212/6^3 \approx 98\%$.

Ein häufig sinnvoller „Trick“ ist jedoch, sich zu überlegen, ob die „umgekehrte“ Wahrscheinlichkeit nicht einfacher zu berechnen ist. Wenn die Gesamtaugenzahl *nicht* mindestens fünf ist, dann kann sie nur drei oder vier sein. Dafür gibt es vier Möglichkeiten, nämlich $(1,1,1)$, $(1,1,2)$, $(1,2,1)$ und $(2,1,1)$. Also ergibt sich die gesuchte Wahrscheinlichkeit als $1 - 4/6^3$. Das ist natürlich viel einfacher.

Lösung 1024: Das Modell ist richtig. Man kann sich das z.B. dadurch klarmachen, dass man mit einem Würfel drei Würfe macht statt mit drei Würfeln einen. Oder man stellt sich vor, die drei Würfel hätten unterschiedliche Farben. Dann wäre der eine Wurf $(1,4,5)$ und der andere $(1,5,4)$.

Lösung 1025: Betrachtet man die beiden Alternativen (i) und (ii) als Ereignisse A und B, so gilt offensichtlich $B \subseteq A$ und damit $P(B) \leq P(A)$. Die zweite Alternative kann also unmöglich *wahrscheinlicher* sein als die erste. (Wenn Sie daran interessiert sind, wie Kahneman diesen Trugschluss erklärt, informieren Sie sich über *Repräsentativitätsheuristik*.)

Lösung 1026: Am Beispiel $k = 5$ will ich das noch mal kurz erläutern. Wenn Sie fünf Würfe machen, dann gibt es 2^5 mögliche Ergebnisse, z.B. $ZBZBZ$ oder $ZBBZZ$. (2^5

sind es, weil es sich hier um Variationen mit Wiederholungen handelt.) Jedes von denen hat dieselbe Wahrscheinlichkeit, aber nur eines entspricht in unserem Wahrscheinlichkeitsraum dem Ereignis $\{5\}$, nämlich $ZZZZB$.

Lösung 1027: Die gesuchte Wahrscheinlichkeit ist:

$$P(\{1,2,3\}) = P(\{1\}) + P(\{2\}) + P(\{3\}) = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} = \frac{7}{8}$$

Allgemein ergibt sich mit der Formel für die geometrische Summe:

$$P(\{1,2,\ldots,k\}) = \sum_{k=1}^{n} P(\{k\}) = \sum_{k=1}^{n} 2^{-k} = \frac{1-2^{-n-1}}{1-2^{-1}} - 1 = 1 - 2^{-n}$$

Lösung 1028: Wir haben es mit einer geometrischen Reihe zu tun:

$$\sum_{k=1}^{\infty} P(\{k\}) = \sum_{k=1}^{\infty} 2^{-k} = \frac{1}{1-2^{-1}} - 1 = 1$$

Und da *muss* ja auch eins herauskommen, weil wir gerade $P(\Omega)$ ausgerechnet haben!

Lösung 1029: Das Ereignis, das uns interessiert, ist die Menge $A = \{2,4,6,8,\ldots\}$, die man als $\{2\} \cup \{4\} \cup \{6\} \cup \ldots$ darstellen kann:

$$P(A) = \sum_{k=1}^{\infty} P(\{2k\}) = \sum_{k=1}^{\infty} 2^{-2k} = \sum_{k=1}^{\infty} 4^{-k} = \frac{1}{3}$$

Falls Sie bei solchen Umformungen oder der damit verbundenen Bruchrechnung unsicher sind, vergessen Sie nicht, dass Ihr Computer Ihnen helfen kann:

```
from sympy import symbols, Sum, oo
k = symbols("k")

Sum(2**(-2*k), (k, 1, oo)).doit()
```

Lösung 1030: Weil π die Fläche des Kreises Ω ist und P normiert sein muss.

Lösung 1031: Das ist ganz einfach. Nehmen Sie das Ereignis, dass der Pfeil irgendeinen Punkt *außer* ω trifft. Die Wahrscheinlichkeit dafür ist

$$P(\Omega \setminus \{\omega\}) = 1 - 0 = 1,$$

aber das bedeutet nicht, dass dieses Ereignis auf jeden Fall eintreten wird. Das einzige Ereignis, das in diesem Modell mit Sicherheit eintreten wird, ist Ω.

Lösung 1032: Man kann sich das z.B. so vorstellen, dass man die Scheibe in n gleich große Stücke unterteilt, wobei die Zahl n durch die zur Verfügung stehende Messgenauigkeit determiniert wird. Das würde ein simples Laplace-Experiment ergeben, bei dem jedem Stück die Wahrscheinlichkeit $1/n$ zugeordnet wird. Erhöht man die Messgenauigkeit, so wird n größer und die Wahrscheinlichkeit, eins der n Stücke zu treffen, kleiner. Das Modell, über das wir gerade gesprochen haben, entspricht einer unbegrenzten Messgenauigkeit: wir lassen n gegen unendlich gehen.

Lösung 1033: Mein Programm sieht so aus:

```
from random import sample
from sympy import isprime

def oneRun ():
    blueInRed = sum(sample([0] * 6 + [1] * 8, 5))
    return not isprime(blueInRed + 6 - (5 - blueInRed))

def manyRuns (n):
    return sum(oneRun() for i in range(n)) / n
```

Hierbei wird ausgenutzt, dass man boolesche Werte addieren kann: `True` verhält sich dann wie die Eins, `False` wie die Null.

Zur exakten Lösung zunächst eine Vorüberlegung. Ist b die Anzahl der blauen Kugeln im roten Behälter, so befinden sich dort außerdem noch $5-b$ rote Kugeln. Damit verbleiben für den blauen Behälter $6-(5-b)=b+1$ rote Kugeln. Die Summe, um die es in der Aufgabenstellung geht, ist also $2b+1$ und hängt nur von b ab. Es gibt sechs Möglichkeiten:

b	0	1	2	3	4	5
$2b+1$	1	3	5	7	9	11

Nur in den beiden markierten Fällen ist die Summe keine Primzahl. Wir brauchen also die Wahrscheinlichkeiten dafür, dass gar keine blaue Kugel im roten Behälter landet oder dass es genau vier sind.

Zum Modellieren setzen wir[96]

$$X = \{1,2,3,4,5,6,7,8,9,10,11,12,13,14\}$$

und $\Omega = \{Y \subseteq X : |Y| = 5\}$. Jedes Element von X steht für eine der vierzehn Kugeln; die ersten sechs Zahlen repräsentieren die roten Kugeln. Jedes Element von Ω steht für eine mögliche Befüllung des roten Behälters und alle haben dieselbe Wahrscheinlichkeit. Aus der Kombinatorik wissen wir, dass $|\Omega| = \binom{14}{5} = 2002$ gilt.

Das Ereignis „keine blaue Kugel im roten Behälter“ sieht so aus:

$$\begin{aligned} A_0 &= \{Y \in \Omega : Y \subseteq \{1,2,\ldots,6\}\} \\ &= \{Y \subseteq \{1,2,\ldots,6\} : |Y| = 5\} \end{aligned}$$

Es gilt $|A_0| = \binom{6}{5} = 6$ und damit $P(A_0) = 6/2002 = 3/1001$.

Das Ereignis „vier blaue Kugeln im roten Behälter“ sieht so aus:

$$\begin{aligned} A_4 &= \{Y \in \Omega : |Y \cap \{1,2,\ldots,6\}| = 1\} \\ &= \{Y_r \cup Y_b : Y_r \subseteq \{1,2,\ldots,6\},\ Y_b \subseteq \{7,8,\ldots,14\},\ |Y_r| = 1,\ |Y_b| = 4\} \end{aligned}$$

Damit folgt $|A_4| = \binom{6}{1} \cdot \binom{8}{4} = 420$, also $P(A_4) = 420/2002 = 30/143$.

Insgesamt ist die gesuchte Wahrscheinlichkeit $P(A_0) + P(A_4) = 213/1001 \approx 21\%$.

[96] Es gibt allerdings auch noch andere Möglichkeiten, das zu modellieren.

Lösung 1034: Das Analogon zum MAPLE-Programm im Video:

```
def P (n):
    prod = 1
    for i in range(n):
        prod *= 365 - i
    return (365**n - prod) / 365**n
```

Lösung 1035: Zum Beispiel so:

$$A_1 = \{(x_1, x_2, x_3) \in \Omega : x_1 + x_2 + x_3 \leq 9\}$$
$$A_2 = \{(x_1, x_2, x_3) \in \Omega : |\{x_1, x_2, x_3\}| < 3\}$$

Lösung 1036: `Omega` ist ein Iterator, der nur einmal durchlaufen werden kann. (Und warum klappt das dann mit `range`? `range` ist in PYTHON-Termini *iterable*, aber rein technisch kein Iterator. Für weitere Details schauen Sie bitte in der Dokumentation nach.)

Lösung 1037: Für PYTHON ist die Zahl 42 „wahr“. Und jede andere Zahl auch, außer der Null.

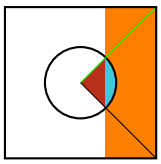
Lösung 1038: Ich habe es so gemacht:

```
Omega = list(filter(lambda L: 1 in L, Omega))
P(lambda L: sum(L) <= 9), P(lambda L: len(set(L)) < 3)
```

Jetzt ist es besser, sich für Antwort (I) zu entscheiden. Die Wahrscheinlichkeit ist fast doppelt so hoch wie bei der zweiten Antwort!

Lösung 1039: Um möglichst einfache Maße zu erhalten, setzen wir fest, dass die vier Ecken des Quadrats $(1,1)$, $(-1,1)$, $(-1,-1)$ und $(1,-1)$ sind. Die Fläche des Quadrats ist dann 4, die Länge der Diagonalen nach Pythagoras $2\sqrt{2}$ und damit der Kreisradius $\sqrt{2}/3$ und die Kreisfläche $2\pi/9$. Die Wahrscheinlichkeit, den Kreis zu treffen, ist damit $\pi/18 \approx 17\%$.

Wir ziehen eine Strecke (in der Skizze unten grün) vom Kreismittelpunkt zur rechten oberen Ecke des Quadrats. Aufgrund der vorgegebenen Größe des Kreises muss der Rand des Kreises diese Linie im Verhältnis $1:2$ durchschneiden. Aufgrund der Breite des orangen Streifens muss dessen linke Seite die Linie aber ebenfalls in diesem Verhältnis durchschneiden. Der Rand des Kreises, die linke Seite des Rechtecks und die Diagonale des Quadrats treffen sich also in einem Punkt.



Die beiden Diagonalen in der Skizze schneiden also genau ein Viertel des Kreises (mit der Fläche $\pi/18$) heraus, das in der Skizze aus dem dunkelroten Dreieck und dem

hellblauen Kreissegment besteht. Das Dreieck ist gleichseitig mit bekannten Winkeln und einer bekannten Seitenlänge. Damit kann man seine Fläche berechnen: $1/9$. Die hellblaue Fläche ist also $(\pi/2-1)/9$. Da die orange Fläche $4/3$ ist, ergibt sich als bedingte Wahrscheinlichkeit für das Treffen des Kreises unter der Bedingung, dass der Treffer im orangen Rechteck liegt, der Wert $(\pi/2-1)/12 \approx 5\%$.

Lösung 1040: Wir haben es mit einem Laplace-Experiment zu tun. Ich notiere ohne weitere Kommentare einfach die notwendigen Schritte. (Man hätte das natürlich auch in Python machen können.)

$$\begin{aligned}
W &= \{1,2,3,4,5,6\} \\
\Omega &= W^2 \\
|\Omega| &= 6^2 = 36 \\
A &= \{(x_1,x_2) \in \Omega : x_1 \neq x_2\} = \Omega - \{(x_1,x_1) : x_1 \in W\} \\
|A| &= 36-6 = 30 \\
B &= \{(x_1,x_2) \in \Omega : x_1 + x_2 = 6\} \\
&= \{(1,5),(2,4),(3,3),(4,2),(5,1)\} \\
|B| &= 5 \\
A \cap B &= \{(1,5),(2,4),(4,2),(5,1)\} \\
|A \cap B| &= 4
\end{aligned}$$

Also ergeben sich $P(A \mid B) = 4/5 = 80\%$ und $P(B \mid A) = 4/30 \approx 13\%$. Natürlich sind $P(A \mid B)$ und $P(B \mid A)$ im Allgemeinen nicht identisch.

Lösung 1041: Die folgende Übersicht zeigt alle 16 Möglichkeiten. Die Ergebnisse, die zu A gehören, sind blau umrandet, die zu B gehörigen orange hinterlegt.

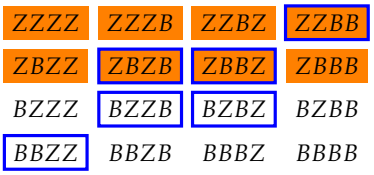
ZZZZ	*ZZZB*	*ZZBZ*	*ZZBB*
ZBZZ	*ZBZB*	*ZBBZ*	*ZBBB*
BZZZ	*BZZB*	*BZBZ*	*BZBB*
BBZZ	*BBZB*	*BBBZ*	*BBBB*

Man erhält also:

$$\begin{aligned}
P(A) &= 4/16 = 1/4 = 25\% & \qquad P(B) &= 5/16 \approx 31\% \\
P(A \mid B) &= 1/5 = 20\% & \qquad P(B \mid A) &= 1/4 = 25\%
\end{aligned}$$

Lösung 1042: Das wird dann und nur dann der Fall sein, wenn der Faktor $P(A)/P(B)$ in der Formel von Bayes nahe bei eins ist. Das ist aber genau dann der Fall, wenn wiederum $P(A)$ und $P(B)$ ähnlich groß sind.

Lösung 1043: Wir verwenden dieselbe Systematik wie in der Lösung zu Aufgabe 1041.



Man erhält $P(A) = 6/16 = 3/8$ und $P(A \mid B) = 3/8$, in beiden Fällen kommt also derselbe Wert heraus.

Lösung 1044: Das Laplace-Experiment, das sich anbietet, ist $\Omega = X^4$ mit der Menge $X = \{1, 2, 3, 4, 5, 6\}$, wobei für ein Tupel $(x_1, x_2, x_3, x_4) \in \Omega$ die Komponente x_i natürlich die Augenzahl des i-ten Wurfes sein soll. In diesem Fall geht es um die folgenden Ereignisse:

$$A = \{(x_1, x_2, x_3, x_4) \in \Omega : x_4 \neq 6\}$$
$$B = \{(6, 6, 6, x_4) : x_4 \in X\}$$

Ihr Kommilitone glaubt, dass $P(A \mid B) > P(A)$ gilt. Man rechnet jedoch nach:

$$\begin{aligned}
|\Omega| &= 6^4 \\
|A| &= 6^3 \cdot 5 \\
P(A) &= |A|/|\Omega| = 5/6 \\
B &= \{(6,6,6,1), (6,6,6,2), (6,6,6,3), (6,6,6,4), (6,6,6,5), (6,6,6,6)\} \\
A \cap B &= \{(6,6,6,1), (6,6,6,2), (6,6,6,3), (6,6,6,4), (6,6,6,5)\} \\
P(A \mid B) &= |A \cap B|/|B| = 5/6
\end{aligned}$$

Das ist aber kein Beweis dafür, dass Ihr Kommilitone im „wirklichen Leben“ nicht recht hat. Siehe die Diskussion im folgenden Text. Auch im Zusammenhang mit dem *Gesetz der großen Zahlen* werden wir darauf noch einmal zurückkommen.

Lösung 1045: Da A und B disjunkt sind, gilt $P(A \cap B) = 0$. Nach (61.1) können A und B unter dieser Voraussetzung nur dann unabhängig sein, wenn $P(A) = 0$ oder $P(B) = 0$ gilt. Zwei sich gegenseitig ausschließende Ereignisse sind also nur dann unabhängig, wenn (mindestens) eines von beiden fast unmöglich ist.

Beim Roulette schließen sich z.B. die Ereignisse „Kugel fällt auf 23“ (A) und „Kugel fällt *nicht* auf 23“ ($A^{\complement}$) gegenseitig aus, sind aber offensichtlich *nicht* unabhängig: $P(A \mid A^{\complement})$ ist null, $P(A)$ aber nicht. Bei einer geometrischen Wahrscheinlichkeit wie im Dartbeispiel sind jedoch das Ereignis A, einen ganz bestimmten Punkt zu treffen, und das Ereignis $A^{\complement}$, ihn *nicht* zu treffen, nach unserer Definition unabhängig – auch wenn das unserer Intuition von „unabhängig“ widersprechen mag. A ist *fast* unmöglich, während $A \cap A^{\complement}$ die leere Menge, also „absolut unmöglich“ ist. In Zahlen ist jedoch die Wahrscheinlichkeit für beide Ereignisse einfach null.

Lösung 1046: Nein. Beachten Sie, dass es 38 mögliche Ergebnisse (Null, Doppelnull und die Zahlen von 1 bis 36) gibt. Nennen wir die Ereignisse aus der Aufgabenstellung A und B, so gilt $P(A) = P(B) = 12/38$ und $P(A \cap B) = 4/38 \neq P(A) \cdot P(B)$.

Ignoriert man wie in Aufgabe 1048 die Nullen, so sind die Ereignisse allerdings unabhängig.

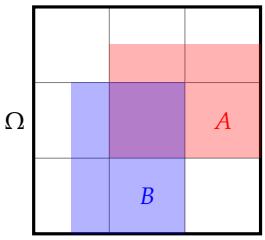
Lösung 1047: Es gilt $P(A) = P(B) = 1/2$ und $P(A \cap B) = 1/4 = P(A)P(B)$. Die Ereignisse sind also unabhängig. Wenn Sie sich darüber wundern, dann liegt es daran, dass Sie *kausale* Abhängigkeit und *stochastische* Abhängigkeit durcheinanderbringen. Die Gesamtaugenzahl hängt natürlich vom Ergebnis des ersten Wurfs (und auch von dem des zweiten) ab. Aber das Wissen darüber, ob das Ereignis A eingetreten ist, liefert Ihnen keine Informationen über die Wahrscheinlichkeit des Eintretens von B, die Sie nicht vorher schon hatten.

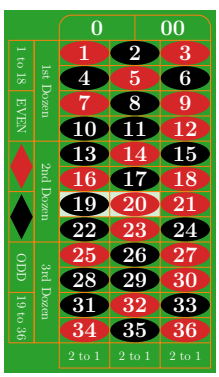
Lösung 1048: Die drei Ereignisse haben alle die Wahrscheinlichkeit $1/2$, denn es gilt $|A| = |B| = |C| = 18$. Außerdem haben wir:

$$\begin{aligned} A \cap B &= \{1,3,5,7,9,12,14,16,18\} \\ A \cap C &= \{2,4,6,8,10,12,14,16,18\} \\ B \cap C &= \{12,14,16,18,30,32,34,36\} \\ |A \cap B| = |A \cap C| &= 9 \\ |B \cap C| &= 8 \\ P(A)P(B) = P(A)P(C) = P(B)P(C) &= 1/4 \\ P(A \cap B) = P(A \cap C) &= 1/4 \\ P(B \cap C) &= 2/9 \end{aligned}$$

Stochastische Unabhängigkeit ist also nicht transitiv. Obwohl A und B sowie A und C jeweils unabhängig sind, sind B und C *nicht* unabhängig.

Lösung 1049: Es gilt $P(A) = P(B) = 3/9 = 1/3$, denn die Rechtecke haben jeweils die Fläche 3. Damit A und B unabhängig sind, muss $P(A \cap B) = 1/9$ gelten, $A \cap B$ also die Fläche 1 haben. Das heißt, dass die Rechtecke so liegen müssen, dass ihr Durchschnitt die Fläche 1 hat; zum Beispiel so:





Lösung 1050: Man muss einfach eine ungerade rote Zahl in der zweiten Hälfte (19 bis 36) schwarz färben und eine gerade schwarze Zahl dafür rot. Zum Beispiel könnte man die Farben von 19 und 20 tauschen.

Lösung 1051: Nein. Es gilt:

$$P(A \cap B \cap C) = P(\{12,14,16,18\}) = 4/36 = 1/9$$

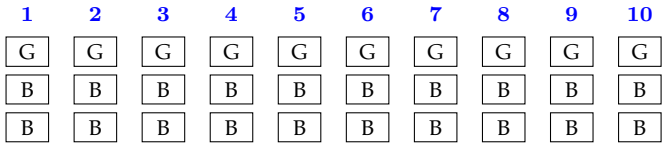
Andererseits ist aber $P(A)P(B)P(C) = 1/8$.

Lösung 1052: Die Wahrscheinlichkeit für einen Defekt bei einer *Gut*-Festplatte innerhalb der ersten fünf Jahre ist $P(G^{\complement}) = 1 - P(G) = 1 - 2P(B)$. Die Wahrscheinlichkeit dafür, dass es bei beiden *Billig*-Platten im selben Zeitraum einen Defekt gibt, ist:

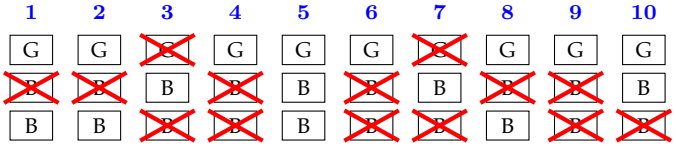
$$P(B^{\complement}) \cdot P(B^{\complement}) = (1 - P(B))^2 = 1 - 2P(B) + P(B)^2$$

Diese Wahrscheinlichkeit ist um $P(B)^2$ größer. Wenn wir also davon ausgehen, dass die Festplatten nicht perfekt sind, dass also $P(B)$ größer als null ist, dann ist es besser, nur eine *Gut*-Platte zu kaufen.

Wenn das Ihrer Intuition widerspricht, stellen Sie sich ein Experiment vor, in dem Sie in zehn verschiedenen Räumen je eine *Gut*- und zwei *Billig*-Festplatten installieren und diese fünf Jahre lang beobachten.



Wenn wir nun von einer konkreten Wahrscheinlichkeit wie $P(G) = 0.8$ ausgehen und die Ausfallquote exakt der Wahrscheinlichkeit entspricht, so sind nach fünf Jahren noch acht *Gut*- und acht *Billig*-Platten in Ordnung. Da die Platten aber unabhängig voneinander ausfallen, könnte das so aussehen:



Es gibt acht Räume mit funktionierenden *Gut*-Festplatten, aber nur sieben Räume, in denen man die Daten auf den *Billig*-Platten noch lesen kann. (Dafür sind in Raum 5 noch beide *Billig*-Platten in Ordnung, aber das ändert nichts daran, dass es mehr Räume gibt, in denen die *Gut*-Platten die Daten bewahrt haben.)

Lösung 1053: Sei $A_{i,k}$ das Ereignis „Super-GAU im i-ten Kraftwerk im Jahr k“, wobei wir der Einfachheit halber den heutigen Tag als den ersten Tag des Jahres $k = 1$ definieren. Nach Aufgabenstellung gehen wir von $P(A_{i,k}^{\complement}) = 1 - 1/10\,000$ aus. Mit B_i bezeichnen wir das Ereignis „Super-Gau im Kraftwerk i in den nächsten zwanzig Jahren“. Da wir davon ausgehen, dass die Ereignisse $A_{i,k}$ unabhängig sind, ergibt sich:

$$P(B_i^{\complement}) = P\left(\bigcap_{k=1}^{20} A_{i,k}^{\complement}\right) = \prod_{k=1}^{20} P\left(A_{i,k}^{\complement}\right) = \left(\frac{9\,999}{10\,000}\right)^{20}$$

$$P\left(\bigcap_{i=1}^{500} B_i^{\complement}\right) = \prod_{i=1}^{500} P\left(B_i^{\complement}\right) = \left(\frac{9\,999}{10\,000}\right)^{10\,000} \approx 37\%$$

Die Wahrscheinlichkeit dafür, dass es in den kommenden zwanzig Jahren einen Super-GAU gibt, liegt also bei ca. 63%. Das klingt etwas beunruhigender als 10 000 Jahre...

Lösung 1054: Geben wir den Ereignissen Namen:

T	Testergebnis ist positiv
H	Patient ist mit HIV infiziert

Wir wollen $P(H \mid T)$ wissen. Nach der Bayes-Formel ist das $P(T \mid H)P(H)/P(T)$. Allerdings kennen wir $P(T)$ nicht. Wir können diesen Wert jedoch mit dem Gesetz der totalen Wahrscheinlichkeit ermitteln:

$$\begin{aligned} P(T) &= P(T \mid H)P(H) + P(T \mid H^{\complement})P(H^{\complement}) \\ &= \frac{999}{1\,000} \cdot \frac{1}{10\,000} + \frac{3}{1\,000} \cdot \frac{9\,999}{10\,000} = \frac{7\,749}{2\,500\,000} \approx 0.003 \end{aligned}$$

Damit ergibt sich nun:

$$\begin{aligned} P(H \mid T) &= P(T \mid H)P(H)/P(T) \\ &= \frac{999}{1\,000} \cdot \frac{1}{10\,000} \cdot \frac{2\,500\,000}{7\,749} = \frac{37}{1\,148} \approx 0.032 \end{aligned}$$

Lösung 1055: Die relevanten Ereignisse sind:

B	Berliner stammt von Bäckerei Brandt.	$P(B) = 150/350 = 3/7$
$B^{\complement}$	Berliner stammt von Bäckerei Albrecht.	$P(B^{\complement}) = 200/350 = 4/7$
S	Berliner ist mit Senf gefüllt.	

Wir wollen $P(B \mid S)$ wissen, brauchen aber erst mal $P(S)$. Den Wert erhalten wir mithilfe der totalen Wahrscheinlichkeit:

$$\begin{aligned} P(S) &= P(S \mid B) \cdot P(B) + P(S \mid B^{\complement})P(B^{\complement}) \\ &= 1/6 \cdot 3/7 + 1/20 \cdot 4/7 = 1/10 \end{aligned}$$

Nun können wir die Bayes-Formel anwenden:

$$P(B \mid S) = P(S \mid B)P(B)/P(S) = 1/6 \cdot 3/7 \cdot 10/1 = 5/7 \approx 71\%$$

In diesem einfachen Fall hätte man natürlich auch anders auf das Ergebnis kommen können. Bäckerei Albrecht liefert zehn Senf-Berliner, Bäckerei Brandt 25. Insgesamt sind es also 35 Senf-Berliner. Und wenn man 25 durch 35 teilt, kommt auch $5/7$ heraus. Das ist aber nicht ganz die gleiche Überlegung, weil man hier von genauen Stückzahlen ausgeht, während in der Aufgabenstellung nur stand, wie die Berliner *im Durchschnitt* befüllt werden. Der Ansatz mit der Bayes-Formel ist allgemeiner.

Lösung 1057: Wir haben es mit einem Laplace-Experiment zu tun. Ist A das Ereignis, eine Zahl zu würfeln, die kleiner als 6 ist, so ist $P(A) = 5/12$. Das ist Ihre Gewinnwahrscheinlichkeit, wenn Sie keine weiteren Informationen haben. Bezeichnen wir mit B das Ereignis, keine Sieben zu würfeln, so ist $P(B) = 11/12$. Außerdem gilt offenbar $A \cap B = A$. Die gesuchte bedingte Wahrscheinlichkeit ist also $P(A \mid B) = P(A \cap B)/P(B) = 5/11$.

Lösung 1058: Da die beiden Würfe sicher unabhängig sind, erhalten wir für die Wurfkombinationen „Bild"-„Zahl" und „Zahl"-„Bild" die Wahrscheinlichkeiten $p(1-p)$ und $(1-p)p$, die natürlich identisch sind.

Lösung 1060: Das ist eine Wiederholung von Aufgabe 22. (Lang, lang, ist's her...) Die maximale Größe für X ist $2^n - 1$, im konkreten Beispiel also 1 099 511 627 775.

Lösung 1061: Im heute üblichen Sprachgebrauch entspricht ein Gigabyte einer Milliarde Byte und ein Byte besteht aus acht Bits. Die Datei besteht also aus $n = 16 \cdot 10^{10}$ Bits. Zur Berechnung der Anzahl der Dezimalstellen muss man $\log_{10} 2^n$ ermitteln, also $n \log_{10} 2$. Man braucht demnach bis zu 48 164 799 307 Dezimalstellen für X. Ausgedruckt würde das einen Papierstapel von mehreren Hundert Meter Höhe ergeben!

Lösung 1062: Wegen $p \leq n^2$ braucht man für p maximal $\lceil \log_2 n^2 \rceil \leq 2\lceil \log_2 n \rceil$ Bits. X mod p ist kleiner als p, also braucht man für diese Zahl auch nicht mehr Bits. Man kommt also auf jeden Fall mit $4\lceil \log_2 n \rceil$ Bits aus. Für $n = 16 \cdot 10^{10}$ sind das maximal 152 Bits statt zwanzig Gigabyte!

Lösung 1063: Die Anzahl der *verschiedenen* Primfaktoren (und nur um die geht es ja) von $|X - Y|$ wird deutlich kleiner als n sein. Zudem können wir bei geringen Unterschieden zwischen den beiden Dateien davon ausgehen, dass $|X - Y|$ wesentlich kleiner als 2^n ist.

Außerdem gehen wir in unserer Abschätzung davon aus, dass die Primfaktoren von $|X - Y|$ alle unterhalb von n^2 liegen. Das muss natürlich auch nicht der Fall sein, weil 2^n ja viel größer als n^2 ist.

Lösung 1064: Das ergibt eine maximale Fehlerwahrscheinlichkeit von etwa 16%.

Lösung 1065: Nein! Die Wahrscheinlichkeit bezieht sich ja nur auf die Fälle, in denen die Dateien verschieden sind. Wenn es also der Normalzustand ist, dass die Dateiinhalte identisch sind und wenn Abweichungen selten sind, dann werden durch das Verfahren bedingte Fehler noch viel seltener auftreten.

Lösung 1066: Das ist leicht auszurechnen:

$$\frac{\pi(2k)}{\pi(2)} \approx \frac{2k/\ln 2k}{k/\ln k} = 2 \cdot \frac{\ln k}{\ln k + \ln 2}$$

Für große k ist dieser Wert offenbar nahe bei 2. Mit anderen Worten: Wenn m groß genug ist, liegt fast die Hälfte der Primzahlen unter m zwischen $m/2$ und m.

Lösung 1067: An der korrekten Abschätzung der Fehlerwahrscheinlichkeit ändert sich dadurch nichts. Wenn n groß ist und p zufällig ausgewählt wird, ist die Wahrscheinlichkeit für die Auswahl einer kleinen Primzahl sehr gering. Nichtsdestotrotz kann man sich z.B. darauf beschränken, nur Primzahlen zwischen $n^2/2$ und n^2 auszuwählen. Die letzte Aufgabe zeigt, dass man damit die theoretische Fehlerwahrscheinlichkeit nur verdoppeln würde.

Lösung 1068: Wenn man davon ausgeht, dass die Primzahlen stochastisch unabhängig ausgewählt werden, dann reicht es natürlich, das Verfahren mehrfach anzuwenden.

Lösung 1069: Am einfachsten ist es wohl, $\Omega = \{0,1\}^3$ zu setzen, wobei die Eins für „Bild" steht. Dann kann man X durch $(x_1, x_2, x_3) \mapsto x_1 + x_2 + x_3$ definieren. Der Wertebereich von X ist $\{0,1,2,3\}$.

Lösung 1070: Das Ereignis sieht so aus $\{(0,0,0),(1,0,0),(0,1,0),(0,0,1)\}$ und hat vier Elemente. Da es sich um ein Laplace-Experiment handelt, ist die gesuchte Wahrscheinlichkeit $4/8 = 1/2$.

Lösung 1071: Sinnvollerweise sollte $\Omega = \{0,1,2,3,\ldots,36\}$ die Ergebnismenge sein. Die Zufallsvariable für *Manque* sieht dann so aus:

$$X_M : \begin{cases} \Omega \to \mathbb{R} \\ \omega \mapsto \begin{cases} 5 & 1 \le x \le 18 \\ -5 & x = 0 \text{ oder } x > 18 \end{cases} \end{cases}$$

Und für *Les trois premiers* so:

$$X_L : \begin{cases} \Omega \to \mathbb{R} \\ \omega \mapsto \begin{cases} 55 & x \le 2 \\ -5 & x > 2 \end{cases} \end{cases}$$

Lösung 1072: Damit der Wert von X größer als 10 ist, muss eines der Ereignisse $(6,6)$, $(5,6)$ oder $(6,5)$ eintreten. Da wir davon ausgehen können, dass der erste und der zweite Wurf unabhängig voneinander sind, ergibt sich:

$$\begin{aligned} P(X \ge 11) &= P(X = 11) + P(X = 12) = P(\{(5,6),(6,5)\}) + P(\{(6,6)\}) \\ &= 2 \cdot \frac{4}{25} \cdot \frac{1}{5} + \frac{1}{5} \cdot \frac{1}{5} = \frac{13}{125} = 10.4\% \end{aligned}$$

Bei einem fairen Würfel wären das nur etwa 8.3% gewesen.

Lösung 1073: Wenn man das Beispiel „wörtlich“ überträgt, sieht es so aus:

```
from itertools import product
from plot import *

Omega = list(product(range(1,7), repeat=2))
X = lambda w: w[0] + w[1]
FX = lambda x: len(list(filter(lambda w: X(w) <= x, Omega))) \
               / len(Omega)
plotFunc2D(FX, [-1,14], samples=500)
```

Man kann es aber noch etwas effizienter machen, wenn man ausnutzt, dass die booleschen Werte `True` und `False` beim Rechnen wie 1 und 0 behandelt werden:

```
FX = lambda x: sum(X(w)<=x for w in Omega) / len(Omega)
```

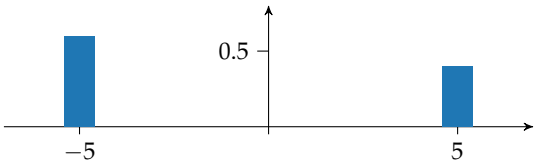
Lösung 1074: $X_1 = 1$ kann man als $X_1 \in \{1\}$ schreiben, $X < 4$ als $X \in \{2,3\}$. $X_1 = 1$ und $X < 4$ haben die Wahrscheinlichkeiten $1/2$ und $1/12$. Der Durchschnitt der beiden Ereignisse ist im üblichen Modell $\{(1,1),(1,2)\}$ mit der Wahrscheinlichkeit $1/18$. Und das ist natürlich nicht dasselbe wie das Produkt von $1/2$ und $1/12$. Es ist auch intuitiv klar, dass die Wahrscheinlichkeit für eine kleine Gesamtaugenzahl sich ändert, wenn man weiß, dass der erste Wurf eine Eins war.

Lösung 1075: Das geht so:

$$P(2 < X \leq 3) = P(X \leq 3) - P(X \leq 2) = F_X(3) - F_X(2)$$

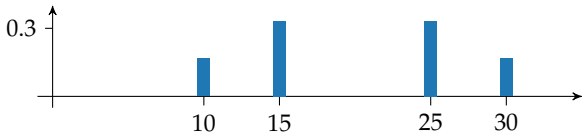
Und $P(X = 3)$ könnte man als $P(X \leq 3) - P(X < 3)$ darstellen. Wir werden aber im weiteren Verlauf sehen, dass man solche Werte wie $P(X = 3)$ entweder kennt oder nicht benötigt.

Lösung 1076: Der Wertebereich von X ist $\{-5, 5\}$, weil der Spieler entweder fünf Euro verliert oder fünf Euro gewinnt. X ist eine diskrete Zufallsvariable, weil der Wertebereich endlich (und damit insbesondere abzählbar) ist. Mit $x_1 = -5$ und $x_2 = 5$ erhalten wir $p_1 = P(X = x_1) = 0.6$ und $p_2 = 0.4$.



Lösung 1077: Der Wertebereich von X ist $\{10, 15, 25, 30\}$. X ist wieder eine diskrete Zufallsvariable. Wahrscheinlichkeiten und Stabdiagramm sehen so aus:

i	1	2	3	4
x_i	10	15	25	30
p_i	1/6	1/3	1/3	1/6



Lösung 1078: Mit der folgenden Funktion erhalte ich mit `test(1000000)` konsistent Werte, die nahe bei $-2/9$ liegen.

```
from random import randrange

def X ():
    sum = randrange(1,7) + randrange(1,7)
    return 20 if sum % 4 == 0 else -sum

def test (n):
    return sum(X() for i in range(n)) / n
```

Gebe ich aber z.B. mehrfach `test(100)` ein, so erhalte ich sehr unterschiedliche Ergebnisse. Es ist halt ein Glücksspiel...

Lösung 1079: $\mathrm{E}(X) = 0.6 \cdot (-5) + 0.4 \cdot 5 = -1$.

Lösung 1080: $\mathrm{E}(Y) = 0.6 \cdot (-50) + 0.4 \cdot 50 = -10 = 10 \cdot \mathrm{E}(X)$. Hatten Sie diesen Zusammenhang vermutet?

Lösung 1081: Wir ergänzen die Tabelle, die wir bereits erstellt hatten:

i	1	2	3	4
x_i	10	15	25	30
p_i	1/6	1/3	1/3	1/6
$p_i x_i$	5/3	5	25/3	5

Als Summe über die letzte Zeile ergibt sich 20. Das hätte man sich anhand des Stabdiagramms natürlich schon denken können, weil es symmetrisch um diese Zahl herum verteilt ist.

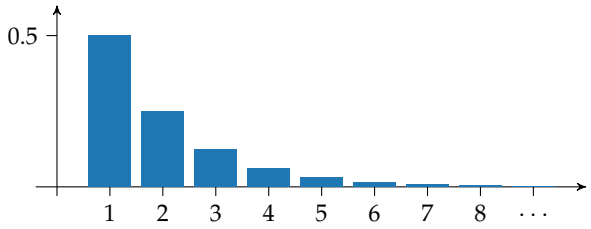
Lösung 1082: Wir wissen bereits, dass $P(X = k) = 2^{-k}$ gilt. Damit ergibt sich:

$$\mathrm{E}(X) = \sum_{k=1}^{\infty} k \cdot P(X = k) = \sum_{k=1}^{\infty} \frac{k}{2^k}$$

In Python geben wir nun dies ein:

```
from sympy import *

k = symbols("k")
Sum(k/2**k, (k,1,oo)).doit()
```

Damit ergibt sich $\mathrm{E}(X) = 2$. Wenn wir das Spiel oft spielen, können wir also davon ausgehen, dass wir im Durchschnitt pro Spiel etwa zwei Würfe brauchen werden.



Sollte Sie das nicht überzeugen, so hilft evtl. dieser experimentelle Zugang:

```
from random import randrange

def X ():
    c = 1
    while randrange(2) == 0:
        c += 1
    return c

def test (n):
    return sum(X() for i in range(n)) / n
```

Und falls Sie sich fragen, wieso sich als Summe der Reihe 2 ergibt, hier eine grafische Begründung:[97]

$$\begin{aligned}
\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\frac{1}{16}+\frac{1}{32}+\ldots &= 1\\
\frac{1}{4}+\frac{1}{8}+\frac{1}{16}+\frac{1}{32}+\ldots &= \frac{1}{2}\\
\frac{1}{8}+\frac{1}{16}+\frac{1}{32}+\ldots &= \frac{1}{4}\\
\frac{1}{16}+\frac{1}{32}+\ldots &= \frac{1}{8}\\
&\vdots
\end{aligned}$$

Der Reihenwert ergibt sich, indem man entweder alle linken Seiten oder alle rechten Seiten addiert.

Lösung 1083: Y hat die Realisierungen $y_1 = 0$, $y_2 = 1$ und $y_3 = 2$. Wir müssen nun die 36 möglichen Fälle durchgehen und erhalten:

i	1	2	3
y_i	0	1	2
$P(Y = y_i)$	1/4	1/2	1/4
$P(Y = y_i)y_i$	0	1/2	1/2

Der Erwartungswert von Y ist also 1. Damit haben wir $\mathrm{E}(X)\,\mathrm{E}(Y) = \mathrm{E}(X) = 7$. Für $Z = XY$ wird es ein bisschen aufwendiger:

i	1	2	3	4	5	6	7	8	9	10	11
z_i	0	3	5	7	8	9	11	12	16	20	24
$P(Z = z_i)$	1/4	1/18	1/9	1/6	1/36	1/9	1/18	1/18	1/12	1/18	1/36
$P(Z = z_i)z_i$	0	1/6	5/9	7/6	2/9	1	11/18	2/3	4/3	10/9	2/3

Als Summe über die letzte Zeile ergibt sich $\mathrm{E}(XY) = 15/2$.

Lösung 1084: Wir haben solche Wahrscheinlichkeiten in Aufgabe 1033 schon mal ausgerechnet. Es gibt insgesamt $\binom{10}{4} = 210$ Möglichkeiten, vier Kugeln zu entnehmen. Genau zwei rote Kugeln erhält man zum Beispiel, indem man zwei der sechs roten und zwei der vier schwarzen Kugeln entnimmt, wofür es $\binom{6}{2} \cdot \binom{4}{2} = 15 \cdot 6 = 90$ Möglichkeiten gibt. Die Wahrscheinlichkeit, genau zwei rote Kugeln zu erwischen, ist also $90/210 = 3/7$. Allgemein ergibt sich:

$$P(X = x) = \frac{1}{210} \cdot \binom{6}{x} \cdot \binom{4}{4-x}$$

i	1	2	3	4	5
x_i	0	1	2	3	4
p_i	1/210	4/35	3/7	8/21	1/14
$p_i x_i$	0	4/35	6/7	8/7	2/7

[97] Siehe dazu auch das Ende von Kapitel 52.

Als Erwartungswert (Summe über die letzte Zeile) erhalten wir $12/5 = 2.4$; im Mittel entnimmt man also mehr als zwei rote Kugeln. Das entspricht sicher auch Ihrer Intuition, weil im Behälter ja mehr rote als schwarze Kugeln sind. Siehe auch Aufgabe 1086.

Lösung 1085: Es ergibt sich:

$$\begin{aligned} \mathrm{E}(aX) &= p_1 \cdot ax_1 + p_2 \cdot ax_2 + \cdots + p_n \cdot ax_n \\ a\,\mathrm{E}(X) &= a(p_1 \cdot x_1 + p_2 \cdot x_2 + \cdots + p_n \cdot x_n) \end{aligned}$$

Klammert man im ersten Ausdruck a aus, erhält man den zweiten. Und das Ausklammern von konstanten Faktoren ist bekanntlich auch bei Reihen erlaubt.

Lösung 1086: Kann man. Es gilt $P(X_i = 1) = 3/5$ für alle i und damit $\mathrm{E}(X_i) = 0.6$. Da der Erwartungswert linear ist, folgt:

$$\begin{aligned} \mathrm{E}(X) &= \mathrm{E}(X_1 + X_2 + X_3 + X_4) \\ &= \mathrm{E}(X_1) + \mathrm{E}(X_2) + \mathrm{E}(X_3) + \mathrm{E}(X_4) = 4 \cdot 0.6 = 2.4 \end{aligned}$$

Einfach, oder? Aber Sie haben sich vielleicht gefragt, wieso $P(X_i = 1) = 3/5$ für alle i gelten soll. Hier eine Begründung: Dass $P(X_1 = 1) = 3/5$ gilt, dürfte klar sein, denn drei Fünftel der Kugeln sind rot. Nun stellen Sie sich vor, dass Sie die vier Kugeln „blind“ entnehmen und jeweils in einen geschlossenen Behälter stecken. Wenn Sie noch wissen, welchen Behälter Sie zuerst befüllt haben, dann ist die Wahrscheinlichkeit, dass in diesem Behälter eine rote Kugel steckt, also $3/5$. Aber nehmen Sie an, die Behälter sähen alle gleich aus und Sie würden diese erst nach mehreren Tagen öffnen – nachdem Sie vergessen haben, in welcher Reihenfolge sie befüllt wurden. Gibt es irgendeinen Grund für die Annahme, dass einer der vier sich anders „verhält“ als die anderen? Nein. Und da die Wahrscheinlichkeit für einen der vier $3/5$ ist, muss sie daher für alle $3/5$ sein. (Aber beachten Sie auch die nächste Aufgabe!)

Lösung 1087: Natürlich nicht. Wenn man z.B. weiß, dass X_1 den Wert 1 hat, dass also die erste Kugel rot war, dann ändert sich dadurch die Wahrscheinlichkeit dafür, dass die zweite Kugel ebenfalls rot ist, von $3/5$ zu $5/9$. Zum Glück ist es für die Gültigkeit der Formel $\mathrm{E}(X + Y) = \mathrm{E}(X) + \mathrm{E}(Y)$ irrelevant, ob X und Y unabhängig sind.

Lösung 1088: Für die Zufallsvariablen aus der Lösung zu Aufgabe 1071 ergibt sich $\mathrm{E}(X_M) = \mathrm{E}(X_L) = -5/37$ und es gilt offenbar $X = X_M + 2X_L$. Damit folgt sofort $\mathrm{E}(X) = -5/37 + 2 \cdot (-5/37) = -15/37 \approx -0.4$. Das Ergebnis hat nichts damit zu tun, ob die beiden am selben Tisch spielen, weil man die Formel $\mathrm{E}(X + Y) = \mathrm{E}(X) + \mathrm{E}(Y)$ auch dann anwenden kann, wenn X und Y nicht unabhängig sind.

Lösung 1089: Sei X_i die Augenzahl des i-ten Würfelwurfs. Wir wissen bereits dass, $\mathrm{E}(X_1) = \mathrm{E}(X_2) = 7/2$ gilt. Weil X_1 und X_2 unabhängig sind und $Y = X_1 X_2$ gilt, können wir die gesuchte Zahl ganz einfach ausrechnen: $\mathrm{E}(Y) = \mathrm{E}(X_1)\,\mathrm{E}(X_2) = 49/4$.

Lösung 1090: Durch $\omega \mapsto \mathrm{E}(X)$ wird eine konstante Funktion definiert, die nach Definition aber auch eine Zufallsvariable ist. Ihr Erwartungswert ist offensichtlich $\mathrm{E}(X)$. Nach den Rechenregeln für Erwartungswerte folgt daher $\mathrm{E}(X - \mathrm{E}(X)) = 0$.

Lösung 1091: Mit $\mu_X = 20$ ergänzen wir die Tabelle aus Aufgabe 1077:

i	1	2	3	4
x_i	10	15	25	30
p_i	1/6	1/3	1/3	1/6
$(x_i - \mu_X)^2$	100	25	25	100
$p_i(x_i - \mu_X)^2$	50/3	25/3	25/3	50/3

Damit ergibt sich $\mathrm{Var}(X) = 50$ als Summe über die letzte Reihe und $\sigma_X = \sqrt{50} \approx 7.1$.

Bei Aufgabe 1084 gehen wir ebenso vor. Hier war $\mu_X = 12/5$.

i	1	2	3	4	5
x_i	0	1	2	3	4
p_i	1/210	4/35	3/7	8/21	1/14
$(x_i - \mu_X)^2$	144/25	49/25	4/25	9/25	64/25
$p_i(x_i - \mu_X)^2$	24/875	28/125	12/175	24/175	32/175

Wir erhalten $\mathrm{Var}(X) = 16/25 = 0.64$ und $\sigma_X = 4/5 = 0.8$.

Im letzten Beispiel erhalten wir als Varianz 2 und als Standardabweichung $\sqrt{2} \approx 1.4$; und zwar so:

$$\mathrm{Var}(X) = \sum_{k=1}^{\infty} (k-2)^2 \cdot P(X = k) = \sum_{k=1}^{\infty} \frac{(k-2)^2}{2^k}$$

```
from sympy import *

k = symbols("k")
Sum((k-2)**2 / 2**k, (k, 1, oo)).doit()
```

Lösung 1092: Man kann diese Aufgabe lösen, indem man wieder eine Tabelle aufstellt. Man kann es aber auch etwas cleverer machen. Wir wissen schon, dass $2\mu_X$ der Erwartungswert von $2X$ ist. Mit den Rechenregeln für Erwartungswerte folgt daher:

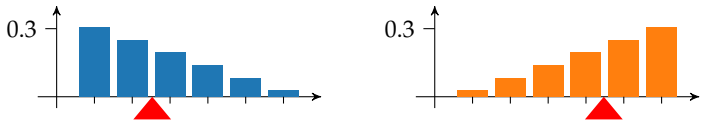
$$\begin{aligned}\mathrm{Var}(2X) &= \mathrm{E}((2X - \mu_{2X})^2) = \mathrm{E}((2X - 2\mu_X)^2) \\ &= \mathrm{E}(4(X - \mu_X)^2) = 4\,\mathrm{E}((X - \mu_X)^2) = 4\,\mathrm{Var}(X)\end{aligned}$$

Die gesuchte Varianz ist also $4 \cdot 35/6 = 70/3 \approx 23$ und die Standardabweichung ist wegen

$$\sigma_{2X} = \sqrt{\mathrm{Var}(2X)} = \sqrt{4\,\mathrm{Var}(X)} = 2\sqrt{\mathrm{Var}(X)} = 2\sigma_X$$

genau doppelt so groß wie die ursprüngliche.

Lösung 1093: Wie üblich kann man das mit Tabellen berechnen, was ich hier jetzt nicht noch mal vorführe. Das ergibt $\mathrm{E}(X_m) = 91/36 \approx 2.53$, $\mathrm{E}(X_M) = 161/36 \approx 4.47$ und $\mathrm{Var}(X_m) = \mathrm{Var}(X_M) = 2555/1296 \approx 1.97$. Dass X_m und X_M dieselbe Varianz haben, ist nicht weiter erstaunlich, wenn man sich die beiden Stabdiagramme anschaut, die Spiegelbilder voneinander sind.



Die Summe von X_m und X_M ist natürlich einfach die Summe der Augenzahlen, also die Zufallsvariable, die wir in den vorherigen Beispielen X genannt haben. Wie wir bereits wissen, gilt:

$$\mathrm{E}(X_m + X_M) = \mathrm{E}(X) = 7 = 91/36 + 161/36 = \mathrm{E}(X_m) + \mathrm{E}(X_M)$$

Allerdings gilt eine analoge Gleichheit *nicht* für die Varianz:

$$\mathrm{Var}(X_m + X_M) = \mathrm{Var}(X) = 35/6 \neq 2555/648 = \mathrm{Var}(X_m) + \mathrm{Var}(X_M)$$

Zur Übung hier noch mal eine Umsetzung dieser Aufgabe in Python, bei der wir ausnutzen, dass es sich um ein Laplace-Experiment handelt:

```
from itertools import product
from matplotlib.pyplot import bar

L = list(range(1,7))
Om = list(product(L, repeat=2))

# die Zufallsvariable Xm, XM und X:
Xmin = lambda w: w[0] if w[0] <= w[1] else w[1]
Xmax = lambda w: w[1] if w[0] <= w[1] else w[0]
X = lambda w: w[0]+w[1]

# berechnet P(X = x)
def P (X, x):
    return sum(map(lambda w: X(w) == x, Om)) / len(Om)
bar(L, list(P(Xmin, x) for x in L))
bar(L, list(P(Xmax, x) for x in L))

# berechnet E(X)
def E (X):
    rng = set(X(w) for w in Om)          # Wertebereich von X
    return sum(x*P(X, x) for x in rng)
E(Xmin), E(Xmax), E(Xmin)+E(Xmax), E(X)

# berechnet Var(X)
def Var (X):
    mu = E(X)
    return E(lambda w: (X(w) - mu)**2)
Var(Xmin), Var(Xmax), Var(Xmin)+Var(Xmax), Var(X)
```

Lösung 1094: Direkte Berechnung liefert $\mathrm{E}(W) = 7/2$ und

$$\mathrm{Var}(W) = \sum_{k=1}^{6} 1/6 \cdot (k - 7/2)^2 = 35/12 \approx 2.9$$

Andererseits wissen wir aber auch, dass sich die Gesamtaugenzahl von zwei Würfen als Summe der Augenzahlen von zwei *unabhängigen* Einzelwürfen ergibt. Daher muss die bereits berechnete Varianz $35/6$ die Summe zweier gleicher Varianzen sein, was ebenfalls auf den Wert $35/12$ führt.

Lösung 1095: X_i sei die Bilanz des i-ten Spiels, d.h. $X = X_1 + X_2 + X_3$. X_i kann offenbar die Werte -1 und 1 mit den Wahrscheinlichkeiten $19/37$ bzw. $18/37$ annehmen. Daraus folgt :

$$\begin{aligned}\mathrm{E}(X_i) &= -1 \cdot 19/37 + 1 \cdot 18/37 = -1/37\\ \mathrm{Var}(X_i) &= (-1 + 1/37)^2 \cdot 19/37 + (1 + 1/37)^2 \cdot 18/37 = 1368/1369\end{aligned}$$

Da die drei Spiele unabhängig sind, ergibt sich:

$$\sigma_X = \sqrt{\mathrm{Var}(X)} = \sqrt{3 \cdot \mathrm{Var}(X_1)} = 6/37 \cdot \sqrt{114} \approx 1.73$$

Lösung 1096: Für den Fall, dass Sie die Lösung nicht ohnehin sofort sehen, kommen Sie mit diesem Ansatz weiter:

$$\begin{aligned}a \cdot 1 + b &\stackrel{!}{=} 3\\ a \cdot 0 + b &\stackrel{!}{=} 11\end{aligned}$$

Das liefert $a = -8$ und $b = 11$; die Variable $Y = -8X + 11$ hat also das gewünschte Verhalten. Es ergibt sich nun:

$$\begin{aligned}\mathrm{E}(Y) &= \mathrm{E}(-8X + 11) = -8\,\mathrm{E}(X) + 11 = -8p + 11\\ \mathrm{Var}(Y) &= \mathrm{Var}(-8X + 11) = (-8)^2\,\mathrm{Var}(X) = 64p(1-p)\end{aligned}$$

Lösung 1097: Aus dem letzten Kapitel wissen wir, dass wir einfach die Summen der Erwartungswerte bzw. der Varianzen der X_k bilden müssen:

$$\begin{aligned}\mathrm{E}(X) &= \sum_{k=1}^{n} \mathrm{E}(X_k) = np\\ \mathrm{Var}(X) &= \sum_{k=1}^{n} \mathrm{Var}(X_k) = np(1-p)\end{aligned}$$

Lösung 1098: Die Wahrscheinlichkeit, mit *einem* Wurf eine Drei oder eine Vier zu würfeln, ist $2/6 = 1/3$. Die Antwort erhält man also so:

```
from scipy.stats import binom

b = binom(20, 1/3)
sum(b.pmf(k) for k in range(7, 21))
```

Oder so:

```
1 - b.cdf(6)
```

Lösung 1099: Ich hatte an folgenden Einzeiler gedacht:

```
def createPdf (b):
    return lambda n: sum(b.pmf(k) for k in range(int(n) + 1))
```

Lösung 1100: Die Wahrscheinlichkeit, eine rote Kugel zu ziehen, ist $3/10$. Also ergibt sich die Antwort so:

```
from scipy.stats import binom

binom(5, 3/10).pmf(2)
```

Lösung 1101: Das berechnet man so:

```
from scipy.stats import hypergeom

h = hypergeom(10, 3, 5)
h.expect(), h.var()
```

Lösung 1102: Für $k = 20$ kann man das z.B. so machen:

```
from scipy.stats import binom, hypergeom
from plot import *

k = 20
b = binom(k, 0.3)
h = hypergeom(100, 30, k)
plotFunc2D([b.cdf, h.cdf], [-0.5, k + 0.5], samples=500)
```

Ihnen sollte auffallen, dass sich für kleiner werdendes k die beiden Kurven immer ähnlicher werden. Das ist anschaulich auch klar: Ist bei der hypergeometrischen Verteilung n klein im Vergleich zu N, so spielt es keine große Rolle, ob das entnommene Objekt nach der Entnahme zurückgelegt wird oder nicht.

Lösung 1103: Man kann höchstens n Objekte entnehmen und von denen können höchstens M die gesuchte Eigenschaft haben. Der höchstmögliche Wert, den X annehmen kann, ist also $\min\{M, n\}$, d.h. er hängt davon ab, welche der beiden Zahlen kleiner ist. Wenn $N - M$ kleiner als n ist, so kann es sein, dass man bei jeder Entnahme zwangsläufig Objekte mit der gesuchten Eigenschaft entnimmt. (Hätte man in unserem einführenden Beispiel zur hypergeometrischen Verteilung acht Kugeln entnommen, wäre immer mindestens eine rote dabei gewesen.) Der minimale Wert, den X annehmen kann, ist somit nicht notwendig null. Insgesamt sieht der Wertebereich daher so aus:

$$\{\max\{0, n - (N - M)\}, \ldots, \min\{M, n\}\}$$

Lösung 1104: X sei die Zufallsvariable für die Anzahl der roten Kugeln, die Frau X entnommen hat, Y sei entsprechend definiert. Dann haben wir $Z = X + Y$. Offenbar gilt $X \sim \mathrm{Hyp}_{10,5,3}$ und $Y \sim \mathrm{Hyp}_{12,7,5}$. In erster Linie ging es mir bei dieser Aufgabe darum, dass Sie erkennen, dass man *nicht* einfach

$$Z = X + Y \sim \mathrm{Hyp}_{10+12,\,5+7,\,3+5} \tag{A.10}$$

erhält. Damit Z z.B. den Wert 8 annimmt, muss Frau X drei rote Kugeln entnehmen und Herr Y fünf. Da die Ereignisse unabhängig sind, ist die Wahrscheinlichkeit dafür:

$$P(Z=8) = P(X=3)\cdot P(Y=5) = \frac{\binom{5}{3}\binom{5}{0}}{\binom{10}{3}} \cdot \frac{\binom{7}{5}\binom{5}{0}}{\binom{12}{5}} \tag{A.11}$$

Das ergibt 7/3168:

```
from sympy import binomial

binomial(5,3)/binomial(10,3) * binomial(7,5)/binomial(12,5)
```

Mit der falschen Verteilung (A.10) hätte sich aber ein *anderer* Wert ergeben:

```
binomial(12,8)/binomial(22,8)
```

Gleichung (A.11) gibt uns jedoch gleich eine Idee dafür, wie man die gesuchte Verteilung korrekt formulieren könnte. Was muss z.B. passieren, damit Z den Wert vier annimmt? Dafür gibt es die Möglichkeit $X = 1$ und $Y = 4 - 1 = 3$, aber auch $X = 2$ und $Y = 4 - 2 = 2$ und noch weitere. Wir können ruhig auch unmögliche Kombinationen hinzunehmen, z.B. $X = -1$ und $Y = 4 - (-1) = 5$. Das ergibt als Summe auch vier, wird aber wegen $P(X = -1)$ nicht gezählt. Insgesamt:

$$P(Z=4) = \sum_{k\in\mathbb{Z}} P(X=k)\cdot P(Y=4-k) \tag{A.12}$$

Das sieht zunächst nach einer unendlichen Reihe aus, aber es ergeben sich nur wenige von null verschiedene Summanden:

$$\begin{aligned} P(Z=4) = {} & P(X=0)P(Y=4) + P(X=1)P(Y=3) + \\ & P(X=2)P(Y=2) + P(X=3)P(Y=1) \end{aligned}$$

Aus (A.12) wird allgemein die folgende Formel:

$$\begin{aligned} P(Z=z) &= \sum_{k\in\mathbb{Z}} P(X=k)\cdot P(Y=z-k) \\ &= \sum_{k\in\mathbb{Z}} P(X=z-k)\cdot P(Y=k) \end{aligned}$$

für beliebige $z \in \mathbb{Z}$. Tatsächlich funktioniert das immer so, wenn X und Y irgendwelche *unabhängigen* diskreten Zufallsvariablen mit ganzzahligen[98] Realisierungen sind.

[98] Es klappt auch dann noch, wenn die Wertebereiche keine Teilmengen von $\mathbb{Z}$ sind, aber dann wird das Aufschreiben etwas umständlicher.

Der einzige Wermutstropfen ist, dass sich diese Formel typischerweise nicht signifikant vereinfachen lässt. Das gelingt nur bei bestimmten Verteilungen, siehe z.B. Aufgabe 1109 weiter hinten.

Lösung 1105: Sie hatten hoffentlich die Idee, die Wahrscheinlichkeit für einen Dreier mithilfe der hypergeometrischen Verteilung auszurechnen:

```
from scipy.stats import hypergeom, geom

p = hypergeom(49,6,6).pmf(3)
p, geom(p).cdf(52)
```

Lösung 1106: Für den ersten Teil gilt $P(X \geq 3) = 1 - P(X < 3) = 1 - F_X(2)$, also:

```
from scipy.stats import geom

1 - geom(0.2).cdf(2)
```

Das ergibt 64 Prozent.

Die allgemeine Formel erhält man z.B. mithilfe der geometrischen Reihe:

$$\begin{aligned} P(X \geq k) &= \sum_{i=k}^{\infty} P(X = i) = \sum_{i=k}^{\infty} p(1-p)^{i-1} \\ &= p(1-p)^{k-1} \sum_{i=0}^{\infty} (1-p)^i \\ &= p(1-p)^{k-1} \cdot \frac{1}{1-(1-p)} = (1-p)^{k-1} \end{aligned}$$

Man kann sich allerdings die Rechnerei auch sparen und sich klarmachen, dass das Ergebnis $X \geq k$ genau dann eintritt, wenn die ersten $k-1$ Versuche Misserfolge waren. Die Wahrscheinlichkeit dafür ist $(1-p)^{k-1}$.

Lösung 1107: Für $X \sim \mathrm{B}_{n,p}$ gilt $\mathrm{Var}(X) = np(1-p) = (1-p)\,\mathrm{E}(X)$. Wenn nun n immer größer wird, wird p immer kleiner und damit konvergiert $1-p$ gegen 1. Im Grenzwert, also für $Y \sim \mathrm{P}_{np}$, folgt $\mathrm{Var}(Y) = \mathrm{E}(Y)$.

Lösung 1108: Ein Quadratzentimeter ist der zehntausendste Teil eines Quadratmeters. Im Durchschnitt ist also mit drei Sandkörnern pro Quadratzentimeter zu rechnen. Die Antwort erhält man somit folgendermaßen:

```
from scipy.stats import poisson

poisson(3).cdf(2)
```

Die Wahrscheinlichkeit liegt bei etwa 42%.

Lösung 1109: Nach Voraussetzung gilt $X \sim \mathrm{P}_7$ und $X \sim \mathrm{P}_5$. Wir setzen $Z = X + Y$.

Ich zeige die Lösung anhand des Beispiels $z = 4$. Mit (64.1) ergibt sich:

$$\begin{aligned}
P(Z=4) &= \sum_{k\in\mathbb{Z}} P(X=k)\cdot P(Y=4-k) \\
&= \sum_{k=0}^{4} P(X=k)\cdot P(Y=4-k) \\
&= \sum_{k=0}^{4} \frac{7^k}{k!}\cdot \mathrm{e}^{-7}\cdot \frac{5^{4-k}}{(4-k)!}\cdot \mathrm{e}^{-5} = \mathrm{e}^{-7-5}\cdot \sum_{k=0}^{4} \frac{7^k}{k!}\cdot \frac{5^{4-k}}{(4-k)!} \\
&= \frac{\mathrm{e}^{-12}}{4!}\cdot \sum_{k=0}^{4} \frac{4!}{k!(4-k)!}\cdot 7^k\cdot 5^{4-k} = \frac{\mathrm{e}^{-12}}{4!}\cdot \sum_{k=0}^{4} \binom{4}{k}\cdot 7^k\cdot 5^{4-k} \\
&= \frac{\mathrm{e}^{-12}}{4!}\cdot (7+5)^4 = \frac{12^4}{4!}\cdot \mathrm{e}^{-12}
\end{aligned}$$

Dabei haben wir zum Schluss den binomischen Lehrsatz (Kapitel 17) verwendet. Offenbar gilt also, wenn man das verallgemeinert, $Z \sim \mathrm{P}_{7+5}$. Die Poisson-Verteilung ist somit ein Beispiel dafür, dass das Summieren von unabhängigen Zufallsvariablen in bestimmten Fällen zu einfachen Verteilungen führen kann.

Falls Sie jetzt denken, dass Ihnen intuitiv ohnehin klar war, was hier herauskommen musste, dann gebe ich Ihnen sofort recht. Wenn bei der einen Firma im Schnitt sieben Anrufe pro Stunde ankommen und bei der anderen fünf, dann kommen insgesamt im Schnitt ja wohl zwölf an. Aber erstens kann einen die Intuition in der Stochastik auch mal in die Irre führen[99] und zweitens ging es mir aus didaktischen Gründen eher darum, dass Sie sich noch einmal Gedanken über die Summen von Zufallsvariablen machen.

Lösung 1110: Offenbar gibt es von jeder der sechs Farben bis auf gelb 17 Bärchen. Also gibt es 34 rote Bärchen. Damit erhält man die Antwort mithilfe der hypergeometrischen Verteilung:

```
from scipy.stats import hypergeom

hypergeom(103, 34, 10).pmf(0)
```

Lösung 1111: Bei jedem Anruf beträgt die Wahrscheinlichkeit, einen Analogfotografen zu erwischen, nach Voraussetzung 0.03. Das kann man mit der geometrischen Verteilung modellieren:

```
from scipy.stats import geom

1 - geom(0.03).cdf(40)
```

(Mit einer Binomialverteilung würde es übrigens auch gehen. Wie?)

Lösung 1112: Die Lottoziehung lässt sich (siehe Aufgabe 1105) mit einer hypergeometrischen Verteilung modellieren; sechs der 49 Zahlen haben die gesuchte Eigenschaft

[99] Sind z.B. für Z immer noch alle Voraussetzungen für eine Poisson-Verteilung erfüllt?

und es werden sechs Zahlen angekreuzt. Gesucht ist in diesem Fall der Erwartungswert:

```
from scipy.stats import hypergeom

hypergeom(49, 6, 6).expect()
```

Lösung 1113: Wir arbeiten hier mit der Poisson-Verteilung. In fünf Sekunden kommen im Durchschnitt $87/12 = 7.25$ Teilchen an.

```
from scipy.stats import poisson

1 - poisson(7.25).cdf(10)
```

Lösung 1114: Das kann man mit einer Binomialverteilung machen:

```
from scipy.stats import binom

1 - binom(15, 0.0001).cdf(1)
```

Lösung 1115: Die Wahrscheinlichkeit, ein Ass zu ziehen, ist $1/13$. Also:

```
from scipy.stats import binom

k = 1
while 1 - binom(k, 1/13).cdf(1) <= 0.9:
    k += 1
k
```

Lösung 1116: Das ist wieder ein Fall für die hypergeometrische Verteilung:

```
from scipy.stats import hypergeom

hypergeom(1000, 5, 2).pmf(2)
```

Lösung 1117: Sie fangen immer wieder von vorne an und jedesmal ist die Wahrscheinlichkeit $1/7$:

```
from scipy.stats import geom

geom(1/7).cdf(6)
```

Lösung 1118: Die Wahrscheinlichkeit ist

$$\begin{aligned} P(1 < X \leq 2) &= F_X(2) - F_X(1) = (1 - \mathrm{e}^{-2\lambda}) - (1 - \mathrm{e}^{-\lambda}) \\ &= \mathrm{e}^{-\lambda} - \mathrm{e}^{-2\lambda} = \mathrm{e}^{-0.2} - \mathrm{e}^{-0.4} \approx 14\% \end{aligned}$$

Lösung 1119: Es gilt immer $P(X \leq 0) = 0$. Für negative t würde sich in (65.1) eine negative Wahrscheinlichkeit ergeben, was nicht sein darf. Damit man es wirklich mit einer Verteilungsfunktion zu tun hat, muss man zwangsläufig $P(X \leq t) = 0$ für $t < 0$ setzen.

Lösung 1120: Die erste Wahrscheinlichkeit ist einfach $P(X \geq 5) = 1 - F_X(5)$. Für die zweite gilt nach Definition der bedingten Wahrscheinlichkeit:

$$P(X \geq 8 \,|\, X \geq 3) = \frac{P(X \geq 8, X \geq 3)}{P(X \geq 3)} = \frac{P(X \geq 8)}{P(X \geq 3)} = \frac{1 - F_X(8)}{1 - F_X(3)}$$

In Python:

```
from scipy.stats import expon

ex = expon(scale=1/0.3)
1 - ex.cdf(5), (1 - ex.cdf(8)) / (1 - ex.cdf(3))
```

Die beiden Werte sind gleich. Ein Bauteil, das bereits drei Jahre „durchgehalten“ hat, hat also eine genauso hohe Wahrscheinlichkeit, die nächsten fünf Jahre zu überstehen, wie ein fabrikneues.

Lösung 1121: Wie in der Aufgabenstellung schon angekündigt ist das in erster Linie Rechenarbeit (und daher eine gute Übung). Auch die geometrische Reihe kommt vor:

$$\begin{aligned} P(X = k + n \,|\, X > n) &= \frac{P(X = k + n)}{P(X > n)} = \frac{(1-p)^{k+n-1} p}{\sum_{i=n+1}^{\infty} (1-p)^{i-1} p} \\ &= \frac{(1-p)^{k+n-1} p}{(1-p)^n p \sum_{i=0}^{\infty} (1-p)^i} \\ &= \frac{(1-p)^{k-1}}{\sum_{i=0}^{\infty} (1-p)^i} = \frac{(1-p)^{k-1}}{1/(1-(1-p))} = \frac{(1-p)^{k-1}}{1/p} \\ &= p(1-p)^{k-1} = P(X = k) \end{aligned}$$

Lösung 1122: Einen grafischen Vergleich erhält man so:

```
from scipy.stats import expon, geom
from plot import *

ex = expon(scale=1/0.2)
ge = geom(0.2)
plotFunc2D([[(k,v.cdf(k)) for k in range(1,10)]
            for v in [ex,ge]], [1, 10])
```

Lösung 1123: Wir ersetzen hier den konkreten Wert 0.2 durch den allgemeineren λ. Zwischendurch verwenden wir die Formel für die geometrische Summe aus Kapitel 16:

$$P(Z_n \leq k) = P(1/n \cdot Y_n \leq k) = P(Y_n \leq kn)$$

$$
\begin{aligned}
&= P(Y_n = 1) + P(Y_n = 2) + \cdots + P(Y_n = kn) \\
&= \sum_{i=1}^{kn} P(Y_n = i) = \sum_{i=1}^{kn} \frac{\lambda}{n} \left(1 - \frac{\lambda}{n}\right)^{i-1} \\
&= \frac{\lambda}{n} \cdot \sum_{i=0}^{kn-1} \left(1 - \frac{\lambda}{n}\right)^{i} = \frac{\lambda}{n} \cdot \frac{1 - \left(1 - \frac{\lambda}{n}\right)^{kn}}{1 - \left(1 - \frac{\lambda}{n}\right)} \\
&= 1 - \left(1 - \frac{\lambda}{n}\right)^{kn} = 1 - \left(\left(1 - \frac{\lambda}{n}\right)^{n}\right)^{k}
\end{aligned}
$$

Am Ende sehen wir die Exponentialfolge. Also ergibt sich:

$$\lim_{n\to\infty} P(Z_n \leq k) = 1 - \mathrm{e}^{-k\lambda}$$

Und das ist natürlich die Exponentialverteilung.

Lösung 1124: Wir müssen $\int_0^\infty x\lambda \mathrm{e}^{-\lambda x}\,\mathrm{d}x$ berechnen. (Das Integral beginnt bei 0, weil der Teil links davon keine Rolle spielt.)

Mit SymPy kann man den Wert auf Wunsch exakt ausrechnen:

```
from sympy import integrate, symbols, exp, oo
from scipy.stats import expon
from fractions import Fraction

x = symbols("x", real=True)
L = Fraction(2, 10)
integrate(x*L*exp(-L*x), (x,0,oo)), expon(scale=1/0.2).expect()
```

Lösung 1125: Die Ableitung ist $-\lambda x \mathrm{e}^{-\lambda x}$, also bis auf das Vorzeichen der Integrand des zu berechnenden Integrals. Mit SymPy hätte man das so herausbekommen:

```
from sympy import symbols, diff, exp, simplify

x = symbols("x")
L = symbols("L", positive=True)
simplify(diff((1 + L*x)*exp(-L*x)/L, x))
```

Da wir nun eine Stammfunktion kennen, können wir das Integral exakt berechnen:

$$
\begin{aligned}
\int_0^\infty x\lambda \mathrm{e}^{-\lambda x}\,\mathrm{d}x &= \lim_{b\to\infty} \left[-(1+\lambda x)\mathrm{e}^{-\lambda x}/\lambda\right]_0^b \\
&= 1/\lambda - \lim_{b\to\infty} (1+\lambda b)\mathrm{e}^{-\lambda b}/\lambda = 1/\lambda
\end{aligned}
$$

Falls man nicht sofort sieht, dass der Grenzwert null ist, kann man ihn auch vom Computer berechnen lassen:[100]

[100] Beachten Sie aber, dass dafür der Zusatz `positive=True` von oben notwendig ist.

```
from sympy import limit, oo

limit((1 + L*x)*exp(-L*x)/L, x, oo)
```

Übrigens hätte SYMPY das Integral auch direkt berechnen können:

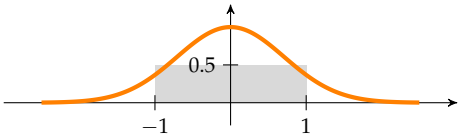
```
from sympy import integrate

integrate(x*L*exp(-L*x), (x,0,oo))
```

Lösung 1126: Die durchschnittliche Dauer zwischen zwei Anrufen ist ein Siebtel einer Stunde. Wir modellieren das also mit einer Zufallsvariablen X mit $X \sim \text{Exp}_7$:

```
from scipy.stats import expon

1 - expon(scale=1/7).cdf(0.5)
```

Die Wahrscheinlichkeit liegt demnach bei etwa drei Prozent.

Lösung 1128: Man kann der Grafik sofort ansehen, dass die Fläche unter der Kurve zu groß ist.



Das eingezeichnete Rechteck hat die Fläche $2 \cdot 0.5 = 1$ und passt fast komplett unter die Kurve, aber bei einer Dichte muss die *Gesamtfläche* von $-\infty$ bis ∞ eins sein.

Lösung 1129: Das kann man so machen:

```
from math import exp, pi, sqrt

def f (x):
    return exp(-x*x/2) / sqrt(2*pi)
plotFunc2D([f, lambda x: f(x - 2)], [-3, 5])
```

Durch die Subtraktion von 3 wird die Kurve entsprechend nach rechts verschoben.

Lösung 1130: Das geht folgendermaßen:

```
from sympy import integrate, exp, pi, sqrt, symbols, oo
```

```
x = symbols("x")
(integrate(exp(-x**2/2)   / sqrt(2*pi), (x, -oo, oo)),
 integrate(exp(-x**2/2/5) / sqrt(2*pi), (x, -oo, oo)))
```

Der Divisor 5 sorgt dafür, dass $\sqrt{5}$ statt 1 herauskommt.

Lösung 1131: So zum Beispiel:

```
from scipy.integrate import quad
from math import exp, pi, sqrt
from plot import *

f = lambda x: exp(-x*x/2) / sqrt(2*pi)
plotFunc2D(lambda x: quad(f, -100, x)[0], [-4, 4])
```

Die Kurve entspricht im Wesentlichen der in Kapitel 48 erwähnten Fehlerfunktion. Damit die beiden Graphen identisch sind, muss man ein paar Anpassungen vornehmen:

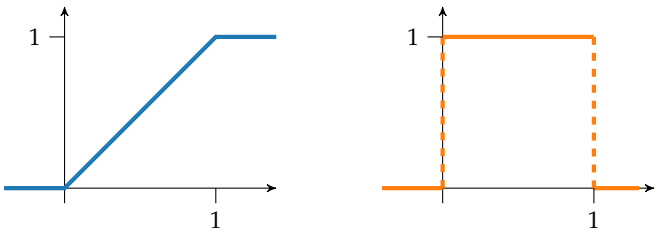
```
from math import erf

plotFunc2D(lambda x: (1 + erf(x / sqrt(2))) / 2, [-4, 4])
```

Lösung 1132: Nichts leichter als das:

```
from scipy.stats import norm

1 - norm(loc=100, scale=15).cdf(138)
```

Lösung 1133: Nein, das wäre keine gute Idee, weil wir ja in Kapitel 12 gelernt haben, dass die Maschinenzahlen in der Nähe der Null dichter liegen. Hätten alle dieselbe Wahrscheinlichkeit, so würden wir häufiger sehr kleine Werte erhalten. Das ist typischerweise nicht das, was man möchte. Nicht jede Programmiersprache geht mit diesem Problem gleich um. Wenn Sie es genau wissen wollen, müssen Sie in der jeweiligen Dokumentation nachschlagen (und hoffen, dass Sie dort eine Antwort finden).

Lösung 1134: Mit Beschriftung sieht es so aus:



Links haben wir die Verteilungsfunktion F_X und rechts die Dichte f_X.

$$F_X : \begin{cases} \mathbb{R} \to \mathbb{R} \\ x \mapsto \begin{cases} 0 & x < 0 \\ x & 0 \le x \le 1 \\ 1 & x > 1 \end{cases} \end{cases} \qquad f_X : \begin{cases} \mathbb{R} \to \mathbb{R} \\ x \mapsto \begin{cases} 0 & x < 0 \\ 1 & 0 \le x \le 1 \\ 0 & x > 1 \end{cases} \end{cases}$$

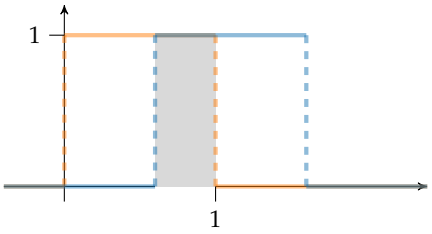
Vielleicht überlegen Sie sich zur Übung noch, wie das für $X \sim \mathcal{U}_{a,b}$ aussehen müsste, wenn a und b irgendwelche reellen Zahlen mit $a < b$ sind.

Lösung 1135: Die Faltung zweier auf $\mathbb{R}$ definierter Funktionen f und g ist folgendermaßen definiert:[101]

$$(f * g)(x) = \int_{-\infty}^{\infty} f(t) \cdot g(x - t)\, \mathrm{d}t$$

Für die Stochastik heißt das: Sind X und Y unabhängige Zufallsvariablen mit den Dichten f_X und f_Y, so hat die Zufallsvariable $X + Y$ die Dichte $(f_X * f_Y)$.

Lösung 1136: Seien X, Y Zufallsvariablen mit $X, Y \sim \mathcal{U}_{0,1}$ und seien f_X und f_Y ihre Dichten. Zeichnet man f_X und die Funktion $t \mapsto f_Y(1.6 - t)$ gemeinsam in eine Skizze, ergibt sich das folgende Bild:



Das Produkt der beiden Funktionswerte ist nur dort nicht null, wo beide Funktionen den Wert eins haben. Die Faltung $f_X * f_Y$ berechnet für das Argument 1.6 also die graue Fläche 0.4, die sich durch $1 - (1.6 - 1) = 2 - 1.6$ ergibt. Zeichnen Sie das entsprechende Bild für z.B. 0.3 statt 1.6, so werden Sie auf die Fläche 0.3 kommen. Allgemein sieht man ohne Rechnung, dass die Faltung so aussehen muss:

$$(f_X * f_Y)(z) = \begin{cases} 0 & x < 0 \text{ oder } x > 2 \\ x & 0 \le x \le 1 \\ 2 - x & 1 < x \le 2 \end{cases}$$

Man kann zeigen, dass die Summe Z_n von n unabhängigen und auf $[0, 1]$ gleichverteilten Zufallsvariablen der sogenannten *Irwin-Hall-Verteilung* folgt, die diese Verteilungsfunktion hat:

$$P(Z_n \le z) = \frac{1}{n!} \cdot \sum_{k=0}^{\lfloor z \rfloor} (-1)^k \binom{n}{k} (z - k)^n \tag{A.13}$$

Lösung 1137: Die Funktionsweise von `F` lässt sich folgendermaßen beschreiben. Sei $X_1, X_2, X_3, \ldots$ eine Folge von unabhängigen Zufallsvariablen, für die $X_n \sim \mathcal{U}_{0,1}$ gilt.

[101] Wobei das natürlich nur dann geht, wenn das entsprechende Integral existiert.

Der n-te Aufruf von `random` in `F` liefert eine Realisierung von X_n. Analog zur letzten Aufgabe schreiben wir Z_n für $X_1 + \cdots + X_n$. Gleichung (A.13) mag etwas furchterregend aussehen, aber der folgende Wert ist ganz einfach zu berechnen:

$$\begin{aligned} P(Z_n < 1) = P(Z_n \leq 1) &= \frac{1}{n!} \cdot \sum_{k=0}^{1} (-1)^k \binom{n}{k} (1-k)^n \\ &= \frac{1}{n!} \cdot \left((-1)^0 \binom{n}{0} (1-0)^0 + (-1)^1 \binom{n}{1} (1-1)^1 \right) = \frac{1}{n!} \end{aligned}$$

Wenn `F` z.B. den Wert 4 zurückgibt, dann bedeutet das, dass Z_3 eine Realisierung hatte, die kleiner als 1 war, während die Realisierung von Z_4 größer als 1 (oder zumindest genauso groß) war. Wir haben also für $n \geq 2$:

$$P(F = n) = P(Z_n \geq 1) - P(Z_{n-1} \geq 1) = \left(1 - \frac{1}{n!}\right) - \left(1 - \frac{1}{(n-1)!}\right) = \frac{n-1}{n!}$$

Außerdem gilt natürlich:

$$P(F = 1) = P(Z_1 \geq 1) = P(X_1 \geq 1) = 0$$

Damit können wir nun den Erwartungswert berechnen, der vielleicht etwas überraschend ist:

$$\begin{aligned} \mathrm{E}(F) &= \sum_{n=1}^{\infty} n \cdot P(F = n) = \sum_{n=2}^{\infty} n \cdot P(F = n) \\ &= \sum_{n=2}^{\infty} n \cdot \frac{n-1}{n!} = \sum_{n=2}^{\infty} \frac{1}{(n-2)!} = \sum_{n=0}^{\infty} \frac{1}{n!} = \mathrm{e} \end{aligned}$$

Lösung 1138: Wir müssen die Faltung der Dichten von X und Y berechnen:

```
from sympy import *
x, t = symbols("x t")

def f (x, mu = 0, sigma = 1):
    return exp(-(x-mu)**2 / 2 / sigma**2) / sqrt(2*pi*sigma**2)

simplify(integrate(f(t) * f(x-t), (t,-oo,oo)))
```

Man sieht an diesem Ergebnis, dass $X + Y$ wieder normalverteilt ist, und zwar mit Erwartungswert 0 und Varianz $1 + 1 = 2$. Beim zweiten Aufgabenteil ergibt sich ebenfalls wieder eine Normalverteilung:

```
simplify(integrate(f(t, mu=1) * f(x-t, mu=2), (t,-oo,oo)))
```

In diesem Fall ist der Erwartungswert $1 + 2 = 3$. Das kann man so verdeutlichen:

```
expand(-(x-3)**2 / 2 / 2)
```

Im dritten Teil, Sie ahnten es schon, kommt auch eine Normalverteilung heraus. Hier ist die Varianz $3^2 + 2^2 = 13$.

```
simplify(integrate(f(t, sigma=3) * f(x-t, sigma=2), (t,-oo,oo)))
```

Ganz allgemein ist die Summe von normalverteilten Zufallsvariablen immer normalverteilt. Das werden wir in Kapitel 67 noch verwenden.

Lösung 1140: Für Y ergeben sich nach Kapitel 64 der Erwartungswert $\mathrm{E}(Y) = 40 \cdot 0.4$ und die Varianz $\mathrm{Var}(Y) = 40 \cdot 0.4 \cdot (1 - 0.4)$. Damit folgen für $\overline{X}^n = Y/40$ dann die Werte $\mathrm{E}(\overline{X}^n) = 0.4$ und $\mathrm{Var}(\overline{X}^n) = 0.4 \cdot (1 - 0.4)/40 = 0.006$. Nach der Faustregel können wir also *ungefähr* $\overline{X}^n \sim \mathcal{N}_{0.4,0.006}$ annehmen. Wegen $Y = 40\overline{X}^n$ ist Y auch normalverteilt mit 40-mal so großem Erwartungswert $40 \cdot 0.4 = 16$ und mit Varianz $40^2 \cdot 0.006 = 9.6$.

Lösung 1141: Doch, sie ist auch hier anwendbar. Die Faustregel besagt das Folgende: Wenn Sie sich zufällig z.B. hundert Menschen (egal ob männlich oder weiblich) herausgreifen und deren *Durchschnittsgewicht* bestimmen und wenn Sie diesen Vorgang sehr oft wiederholen und als Stabdiagramm visualisieren, dann ergibt das eine „richtige“ Glockenkurve.

Lösung 1142: Ich nehme an, Ihr „gesunder Menschenverstand“ rät Ihnen, es mit dem *arithmetischen Mittel* zu versuchen:

$$(x_1 + x_2 + \cdots + x_{10})/10 = 4.2$$

Wir werden im Verlaufe des Kapitels sehen, dass dies auch aus Sicht der Stochastik die beste Lösung ist. Insbesondere werden wir auch sehen, *warum* das so ist.

Lösung 1143: Zum Beispiel könnte sich durch die Durchführung eines Experimentes das Labor aufheizen. Andererseits könnte die Anfangstemperatur das Ergebnis beeinflussen. Daher muss vor jedem Versuch die Raumtemperatur kontrolliert und das Labor ggf. abgekühlt werden. Vergisst man das, sind die Versuche nicht mehr unabhängig.

Lösung 1144: Ich nehme an, dass Ihr Vorschlag $(X_1 + \cdots + X_n)/n$ lautet. Das ist wieder das arithmetische Mittel, wird in diesem Fall aber anders interpretiert, weil wir sozusagen die durchschnittliche Anzahl der Erfolge berechnen.

Lösung 1145: Dass das arithmetische Mittel auch ein konsistenter Schätzer für den Erwartungswert ist, besagt das Gesetz der großen Zahlen.[102]

Lösung 1146: Man könnte es z.B. so machen:

```
def mean (dist, n):
    return sum(dist.rvs(n)) / n
def unbiased (dist, n, m):
    return sum(mean(dist, n) for i in range(m)) / m
```

[102] Und das Theorem von Bernoulli liefert uns übrigens einen konsistenten Schätzer für Einzelwahrscheinlichkeiten, von dem man auch leicht zeigen kann, dass er erwartungstreu ist. (Siehe Aufgabe 1144.)

Um die Erwartungstreue des arithmetischen Mittels von zehn binomialverteilten Zufallsvariablen zu prüfen, gibt man dies ein:

```
from scipy.stats import binom

bi = binom(30,0.3)
bi.expect(), unbiased(bi, 10, 10000)
```

Und so geht man vor, um die Konsistenz des arithmetischen Mittels von geometrisch verteilten Variablen zu prüfen:

```
from scipy.stats import geom

ge = geom(0.42)
ge.expect(), [mean(ge, 10**k) for k in range(2, 8)]
```

Lösung 1147: So zum Beispiel:

```
from numpy import std

std([test1(1000000) for i in range(50)])
std([test2(1000000) for i in range(50)])
```

Auf meinem Rechner haben die Ergebnisse von `test2` eine geringere Standardabweichung.

Lösung 1149: Für die Summe gilt dann $X_1 + X_2 \sim \mathcal{N}_{\mu_1+\mu_2,\sigma_1^2+\sigma_2^2}$, d.h. Erwartungswert und Varianz werden einfach addiert. Das haben wir uns bereits in Kapitel 63 überlegt.

Lösung 1150: Nein, können wir nicht. Man kann nur Werte einsetzen, die sich als arithmetisches Mittel von zehn entnommenen Tüten ergeben haben!

Lösung 1151: Ja. Der Wert liegt innerhalb der folgendermaßen ermittelten Intervallgrenzen:

```
from scipy.stats import norm

norm(loc=500, scale=5/sqrt(20)).interval(0.95)
```

Lösung 1152: Ich habe das so gemacht:

```
from scipy.stats import norm, t
from math import sqrt
from numpy import mean, std

def test ():
```

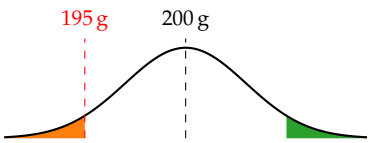
```
    M, n, p = 42000, 200, 0.99
    no = norm(loc=M, scale=100)
    L = no.rvs(n)
    m, s = mean(L), std(L, ddof=1)
    d = t(n-1).interval(p)[1] * s / sqrt(n)
    return abs(M - m) <= d

m = 10000
sum(test() for i in range(m)) / m
```

Das Ergebnis sollte natürlich in etwa bei 0.99 liegen. Das ist ja gerade der Sinn des Konfidenzniveaus.

Lösung 1153: Man müsste die Kurve schmaler machen. Und wir wissen ja schon, wie das geht: Man braucht dafür größere Stichproben. Sie können fast immer davon ausgehen, dass die Sensitivität eines Tests sich dadurch verbessern lässt, dass man mehr Daten zur Verfügung hat. Siehe auch die Lösung von Aufgabe 1156.

Lösung 1154: Die Nullhypothese wäre, dass die Wahrscheinlichkeit für eine Landung auf der Butterseite 50% beträgt.

Lösung 1155: Wir erinnern uns an das Beispiel zur Berechnung des p-Wertes von Seite 856:



Die grüne Fläche brauchten wir, weil es ein zweiseitiger Test war, und als p-Wert ergab sich das Doppelte der orangen Fläche: $2 \cdot 4.7\%$. Für einen *einseitigen* Test können wir den von `ttest_1samp` zurückgegebenen p-Wert also einfach halbieren, denn wir brauchen in diesem Fall nur die orange Fläche. (Allerdings müssen wir auch noch prüfen, auf welcher Seite des $\boldsymbol{H_0}$-Erwartungswertes unsere Stichprobe liegt. Was zu tun ist, wenn das Stichprobengewicht größer als 200 g ist, überlegen Sie vielleicht mal selbst.)

Lösung 1156: Sie werfen den Würfel mehrfach und notieren die Ergebnisse. Sie zählen, wie oft jede Augenzahl vorkommt. Das könnte z.B. so aussehen:

Augenzahl	1	2	3	4	5	6
Häufigkeit	14	16	20	22	17	31

In Python geben Sie nun dies ein:[103]

[103] In diesem speziellen Fall hätten Sie beim Aufruf von `chisquare` übrigens das zweite Argument weglassen können. SciPy nimmt dann an, dass die erwarteten Häufigkeiten alle gleich sind.

```
from scipy.stats import chisquare

chisquare([14, 16, 20, 22, 17, 31], [20, 20, 20, 20, 20, 20])
```

Der p-Wert liegt bei etwa 9.7%. Bei einem üblichen Signifikanzniveau von 5% müssten Sie also trotz des „auffälligen“ Wertes von 31 Sechsen davon ausgehen, dass mit dem Würfel wohl alles in Ordnung ist.

An diesem Beispiel kann man übrigens gut erkennen, wie die Größe der Stichprobe die Sensitivität des Tests beeinflusst. Nehmen wir an, der Würfel sei gefälscht und die Wahrscheinlichkeit für eine Sechs betrüge $1/4$ statt $1/6$, während die anderen Augenzahlen alle dieselbe Wahrscheinlichkeit hätten. Die obigen Häufigkeiten würden dazu „passen“ und wir hätten es mit einem Fehler 2. Art zu tun. Geben Sie stattdessen das Zehnfache der obigen Werte ein, so ist die Aussage des Tests eindeutig:

```
chisquare([140, 160, 200, 220, 170, 310])
```

Lösung 1157: Das ist ganz einfach:

```
from scipy.stats import ttest_rel

V = [3.8, 3.4, 3.5, 2.6, 3.3, 2.9, 3.9, 4.2, 1.8, 0.9]
N = [3.9, 4.4, 3.4, 3.9, 7.7, 4.5, 3.2, 2.9, 3.7, 4.1]
ttest_rel(V, N)
```

Das ergibt einen p-Wert von knapp sieben Prozent. Wäre Ihre Alternativhypothese allerdings gewesen, dass die Spielzeit sich durch die Erhöhung des Schwierigkeitsgrades *erhöht*, so hätte es sich um einen einseitigen Test gehandelt. Der p-Wert wäre dann nur halb so groß gewesen (siehe Aufgabe 1155), da die durchschnittliche Spieldauer ja in der Tat angestiegen ist.

Lösung 1158: Man rechnet die Paardifferenzen explizit aus und vergleicht mit dem Wert null, der der Nullhypothese entspricht:

```
from scipy.stats import ttest_1samp

ttest_1samp([n - v for n, v in zip(N, V)], 0)
```

Lösung 1159: Das ginge so:

```
from scipy.stats import chisquare

chisquare([42, 58])
```

Da der p-Wert größer als 10% ist, müssen Sie davon ausgehen, dass Murphys Gesetz nicht gilt.

Lösung 1160: Wenn danach der Küchenfußboden ordentlich gesäubert wird, spricht nichts dagegen. Die Ergebnisse sind fast identisch.

Lösung 1161: Bei *B* würde es so aussehen:

```
from scipy.stats import ttest_1samp

L = [1] * 58 + [0] * 42
ttest_1samp(L, 0.5)
```

Bei *A* könnte `L` z.B. die Form `[0, 1] * 42 + [1] * 16` haben. Das darf natürlich *nicht* zu unterschiedlichen Ergebnissen führen, weil sich der Test nur für den Erwartungswert „interessiert“ und nicht für die Reihenfolge.

Abgesehen davon: Betrachten wir das Experiment als eine Abfolge von hundert unabhängigen Würfen, dann haben beide Ergebnisse (die von *A* protokollierte Wurffolge und die von *B*) dieselbe Wahrscheinlichkeit.

Lösung 1162: Pro Pixel werden drei Farbinformationen (rot, grün und blau) gespeichert und pro Farbe werden 8 Bit benötigt. Das ergibt drei Byte pro Pixel, also

$$1\,800 \cdot 1\,200 \cdot 3 = 6\,480\,000$$

Byte für das gesamte Foto. Die Datei ist allerdings noch 54 Byte größer. Das sind Verwaltungsdaten am Anfang der Datei.

Lösung 1165: Ja, es geht. So, wie die Frage formuliert war, müssten Sie einfach ein Programm schreiben, das gar nichts macht. Mit anderen Worten wäre f die Identität: die „komprimierte“ Version einer Datei D wäre immer einfach D selbst. Das würde zwar niemandem helfen, aber der Anforderung, dass komprimierte Dateien niemals größer als das Original sein dürfen, wäre Genüge getan.

Lösung 1166: Das Kompressionsverfahren f muss injektiv sein, damit es *umkehrbar* ist – siehe Kapitel 19. Wäre es nicht umkehrbar, so könnte man D aus $f(D)$ nicht immer mit Sicherheit rekonstruieren. Man würde dann nicht von „verlustfrei“ sprechen.

Lösung 1167: Es gibt mehr als vier Billionen 42-Bit-Dateien bzw. genauer: 2^{42}. Siehe Kapitel 17. Die Antwort auf die zweite Frage ist $2^0 + 2^1 + 2^2 + \cdot + 2^{42} = 2^{43} - 1$. Dabei haben wir die „leere Datei“ mitgerechnet, die aus null Bits besteht. Wenn Sie das stört, müssen Sie von diesem Ergebnis noch eins subtrahieren.

Lösung 1168: Es muss sich offenbar um eine Binomialverteilung mit den Parametern 24 und $1/2$ handeln. Wenn Sie sich das Stabdiagramm der zugehörigen Wahrscheinlichkeitsfunktion zeichnen lassen, werden Sie bis auf die Beschriftung der y-Achse keinen Unterschied zur Verteilung der Bits erkennen können.

Lösung 1169: Die einzelnen Zeichen werden *nicht* unabhängig voneinander „gesendet“. Die Wahrscheinlichkeit für ein *H* ist z.B. wesentlich größer als üblich, wenn der letzte Buchstabe ein *C* war. Auch der *vorletzte* Buchstabe kann Auswirkungen auf die Wahrscheinlichkeit haben, und natürlich spielen in Texten auch *Wörter* und nicht nur *Buchstaben* eine Rolle.

Obwohl dieses Modell nicht immer angemessen ist, eignet es sich doch oft als gute Näherung. Außerdem arbeitet man in der Informationstheorie natürlich auch mit komplexeren Modellen; Quellen müssen dort nicht notwendig *gedächtnislos* sein. Für unsere Zwecke reicht aber diese vereinfachte Darstellung.

Lösung 1170: Das ist die Funktionalgleichung des Logarithmus; siehe Kapitel 44.

Lösung 1171: Uns interessiert die Funktion auf dem Intervall $(0, 1]$ und dort ist der Logarithmus negativ. Da (i) negative Werte verbietet, müssen wir mit einem negativen Wert multiplizieren. (Rein technisch könnten wir auch $a = 0$ nehmen und damit alle Forderungen erfüllen. Es ist aber wohl offensichtlich, dass das keine Funktion ergibt, die uns weiterhilft.)

Lösung 1172: Ungefähr 0.88. Das berechnet man so:

```
from math import log

-0.3*log(0.3, 2) - 0.7*log(0.7, 2)
```

Lösung 1173: Am einfachsten kann man sich das wohl dadurch klarmachen, dass wir bei der Beschreibung von Shannons Grundmodell einen Wahrscheinlichkeitsraum mit der Ergebnismenge Σ definiert haben. (Zur Erinnerung: Σ ist das *Alphabet*.) Die Ereignismenge ist $\wp(\Sigma)$ und das Wahrscheinlichkeitsmaß ist durch $P(x_i) = p_i$ gegeben. Da I eine Abbildung ist, deren Definitionsbereich Σ und deren Wertebereich eine Menge von reellen Zahlen ist, handelt es sich um eine Zufallsvariable nach der Definition aus Kapitel 63. Die Bedingung (63.1) muss in diesem Fall nicht überprüft werden.

Lösung 1174: Im Morsealphabet haben die häufiger vorkommenden Buchstaben (also die mit geringerem Informationsgehalt) kürzere Codes als die seltenen. Ein T wird z.B. nur durch — kodiert, ein Q hingegen durch — — · —.

Lösung 1175: Fünf Bit bräuchte man. Damit kann man bis zu $2^5 = 32$ verschiedene Zeichen darstellen. Man „verschenkt“ dann aber sechs „freie Plätze“ für Zeichen, die gar nicht benötigt werden. Die vom Quellencodierungstheorem prognostizierte Rate von kaum mehr als 4.06 Bit pro Zeichen erreicht man also nicht, wenn alle Zeichen gleich lang sind.

Lösung 1176: Nein, definitiv nicht, weil es viel mehr große Dateien als kurze Programme gibt. Und da das die letzte Aufgabe im Buch war, überlasse ich Ihnen die detaillierte Ausarbeitung der Antwort…

Lösung P1: Die Antwort ist 6171. Man kann das z.B. so lösen:

```
def f (n):
    return n // 2 if n % 2 == 0 else 3 * n + 1

def countSteps (n):
    counter = 0
```

```
    while n != 1:
        n = f(n)
        counter += 1
    return counter

def findStart ():
    n = 1
    start = -1
    maxSteps = -1
    while n <= 10000:
        steps = countSteps(n)
        if steps > maxSteps:
            maxSteps = steps
            start = n
        n += 1
    return start
```

Lösung P2: Die ersten vier vollkommenen Zahlen sind 6, 28, 496 und 8128. Hier ist ein Lösungsvorschlag:

```
def divisorSum (n):
    sum = 0
    candidate = 1
    while candidate < n:
        if n % candidate == 0:
            sum += candidate
        candidate += 1
    return sum

def findPerfectNumbers (m):
    result = []
    n = 1
    while len(result) < m:
        if n == divisorSum(n):
            result.append(n)
        n += 1
    return result

findPerfectNumbers(4)
```

Sie werden bemerken, dass das schon recht lange dauert. Ruft man die Funktion mit `5` statt `4` auf, dauert es sogar *sehr* lange. Man kann die Funktion mit einfachen Mitteln ein bisschen schneller machen. Probieren Sie z.B. den folgenden Code aus. (Warum funktioniert das?)

```
def divisorSum (n):
    sum = 0
    candidate = 1
    m = n // 2
    while candidate <= m:
        if n % candidate == 0:
            sum += candidate
        candidate += 1
    return sum
```

Trotzdem bringt das nicht sehr viel. Schneller wird es, wenn man statt einer interpretierten Sprache wie PYTHON eine kompilierte wie C benutzt. Aber auch dann wird dieser Algorithmus zu langsam sein, um wirklich große vollkommene Zahlen zu finden. Um noch mehr zu erreichen, muss man mit aufwendigeren mathematischen Methoden arbeiten; die Informatik allein wird einem nicht helfen.

Lösung P3: Man kann es z.B. so machen:

```
def isPrimitive (a, n):
    i = 2
    power = a
    powers = [a]
    while i < n:
        power = (power * a) % n
        if power in powers:
            return False
        powers.append(power)
        i += 1
    return True

def hasPrimitive (n):
    a = 1
    while a < n:
        if isPrimitive(a, n):
            return True
        a += 1
    return False

def showPrimitives (m):
    result = []
    n = 2
    while len(result) < m:
        if hasPrimitive(n):
            result.append(n)
        n += 1
```

```
    return result

showPrimitives(20)
```

Das Ergebnis ist die Liste der ersten 20 Primzahlen. Tatsächlich hat $\mathbb{Z}/n\mathbb{Z}$ dann und nur dann eine primitive Einheitswurzel, wenn n eine Primzahl ist. (Warum das so ist, sollte klar werden, wenn Sie die entsprechenden Kapitel des Buches gelesen haben.)

Beachten Sie, dass im obigen Code nie explizit eine Potenz (`**` in PYTHON) berechnet wird. Für große n bringt das potentiell Geschwindigkeitsvorteile, weil weniger multipliziert werden muss.

Lösung P4: Zunächst die einfache Funktion:

```
def pow (a, b, n):
    res = 1
    while b > 0:
        res = (res * a) % n
        b -= 1
    return res
```

Mit der Funktion `convDecToBin` (siehe Aufgabe 33) kann man `pow2` so schreiben:

```
def pow2 (a, b, n):
    res = 1
    counter = 0
    for i in convDecToBin(b):
        res = (res * res) % n
        counter += 1
        if i == 1:
            res = (res * a) % n
            counter += 1
    print("{} Multiplikationen".format(counter))
    return res
```

Wenn man es noch etwas eleganter haben will, erspart man es sich, die Binärdarstellung wie in `convDecToBin` erst zu berechnen und dann umzudrehen. Stattdessen arbeitet man gleich in der umgekehrten Reihenfolge. Im Falle von 6^{41} würde das so aussehen:

6^{41}		6^9			6^1	6^0	`res`
6^{32}	6^{16}	6^8	6^4	6^2	6^1		`temp`
1	0	1	0	0	1		

`res` (das spätere Resultat) hat zunächst den Wert $6^0 = 1$. Die Tabelle wird von rechts nach links gelesen. Der Wert `temp` wird in jedem Schritt quadriert. Immer wenn wir eine Eins als Binärziffer sehen, wird `res` mit `temp` multipliziert. In PYTHON sieht das so aus:

```
def pow2 (a, b, n):
    temp = a
    res = 1
    while b > 0:
        if b % 2 == 1:
            res = (res * temp) % n
        b //= 2
        temp = (temp * temp) % n
    return res
```

Die Technik, die wir hier gelernt haben, nennt man übrigens *binäre Exponentiation*. Sie ist schon seit 200 v. Chr. (Indien) bekannt und wird heutzutage im Prinzip in jeder Programmiersprache eingesetzt, die hohe Potenzen exakt berechnen kann.

Lösung P5: Mit `extGCD` (siehe Aufgabe 115) ist es ganz einfach:

```
def inverse (a, n):
    d, [x1, b1, x2, b2] = extGCD(a, n)
    if d != 1:
        print("{} und {} nicht teilerfremd!".format(a, n))
    return (x1 if b1 == a else x2) % n

def div (a, b, n):
    return (a * inverse(b, n)) % n
```

Lösung P6: Man kann das so lösen:

```
L = []
n = 2
while n <= 100:
    a = 1
    found = False
    while a < n:
        i = 0
        p = 1
        while i < n:
            p = (p * a) % n
            if p == 0:
                found = True
                break
            i += 1
        if found:
            break
        a += 1
    if found:
```

```
        L.append(n)
    n += 1
L
```

Die ausgegebenen Zahlen sind genau die, in deren Primfaktorenzerlegung mindestens ein Quadrat vorkommt, z.B. $27 = 3^2 \cdot 3$ oder $50 = 5^2 \cdot 2$. (Die „anderen“ Zahlen nennt man übrigens *quadratfrei*.)

Lösung P7: Meine Lösungen sehen so aus:

```
from math import gcd

def phi1 (n):
    c = 0
    for i in range(1, n + 1):
        if gcd(i, n) == 1:
            c += 1
    return c

def phi2 (n):
    factors = primeFactors(n)
    if len(factors) == 0:
        return 1
    last = 0
    result = 1
    for factor in factors:
        if factor == last:
            result *= factor
        else:
            result *= factor - 1
            last = factor
    return result
```

Wenn Sie alles richtig gemacht haben, berechnen `phi1` und `phi2` immer denselben Wert. (Nicht nur für die Eingaben 2 bis 2000, sondern für *alle* natürlichen Zahlen.)

Sie berechnen hier die sogenannte *Eulersche φ-Funktion*. Bei Interesse können Sie sich ja mal überlegen, warum in beiden Fällen dasselbe herauskommen muss. (Hinweise: Welchen Wert muss `phi1(p)` haben, wenn p eine Primzahl ist? Welcher Zusammenhang besteht zwischen `phi1(m) * phi1(n)` und `phi1(m * n)`, wenn m und n teilerfremd sind, und warum?)

Lösung P8: Eine Summe paarweise verschiedener[104] Stammbrüche nennt man auch einen *ägyptischen Bruch*, weil die alten Ägypter offenbar nur Hieroglyphen für Stammbrüche hatten und sie daher andere Brüche als solche Summen darstellen mussten.

[104] Wenn die Summanden nicht verschieden sein müssten, wäre die Aufgabe trivial; zum Beispiel $3/7 = 1/7 + 1/7 + 1/7$.

Eine Funktion, die beide Aufgaben löst, könnte so aussehen:[105]

```
from fractions import Fraction

def greedy (x, forceOdd = False):
    L = []
    print(str(x) + " = ", end = "")
    u = 0
    while x > 0:
        inv = Fraction(1) / x
        u = max(u+1, int(inv))
        if u < inv:
            u += 1
        if forceOdd and u % 2 == 0:
            u += 1
        f = Fraction(1, u)
        L.append(f)
        x -= f
    print(" + ".join(map(str, L)))
```

Wenn man das Argument `forceOdd` nicht benutzt, wird diese Aufgabe für Brüche, die kleiner als eins sind, immer eine Lösung finden. Das kann man folgendermaßen begründen:[106]

Wenn wir zu einem Bruch p/q den größten Stammbruch $1/m$ mit $1/m \leq p/q$ suchen, dann gilt für den nächstgrößeren Stammbruch $1/(m-1)$ offenbar $1/(m-1) > p/q$. Wenn $1/m$ im späteren Verlauf des Algorithmus noch einmal vorkommen würde, dann müsste $1/m \leq p/q - 1/m$, also $1/m + 1/m \leq p/q < 1/(m-1)$ gelten. Das bedeutet aber $2/m < 1/(m-1)$ bzw. (wenn man beide Seiten mit den Nennern multipliziert) $m < 2$. Dann verbleibt aber nur die Möglichkeit $m = 1$, was wegen $1/m \leq p/q < 1$ nicht sein kann. Also sind alle Stammbrüche zunächst einmal wirklich verschieden.

Außerdem folgt aus $1/(m-1) > p/q$ durch simple Umformung $pm - q < p$. Daher hat der auf p/q folgende Bruch $p/q - 1/m = (pm-q)/(qm)$ einen kleineren Zähler als p/q. Da das nicht immer so weitergehen kann, muss der Algorithmus irgendwann terminieren. Es wird also immer eine Lösung gefunden. Ebenso lässt sich leicht begründen, warum man *keine* Lösung der Art

$$\frac{p}{q} = \frac{1}{m_1} + \frac{1}{m_2} + \frac{1}{m_3} + \ldots \frac{1}{m_k} \tag{A.14}$$

[105] In der Informatik nennt man Algorithmen, die in jedem Schritt einfach immer die Lösung wählen, die zu dem Zeitpunkt die beste zu sein scheint, *greedy* oder *gierig*.

[106] Dieser Algorithmus aus dem 13. Jahrhundert stammt übrigens von Fibonacci. Siehe Seite 137.

finden kann, wenn q gerade ist und p und alle m_i ungerade sind. Gleichung (A.14) ließe sich dann nämlich so umformen (wobei wir der Übersichtlichkeit halber $k = 3$ gesetzt haben):

$$\frac{p}{q} = \frac{m_2m_3 + m_1m_3 + m_1m_2}{m_1m_2m_3}$$

Auf der rechten Seite stünde nun aber im Nenner eine Zahl, die ungerade wäre, und damit wäre die Gleichung falsch.

Dass man für ungerade q jedoch immer eine Lösung finden kann, bei der alle Stammbrüche ungerade Nenner haben, kann man mathematisch beweisen. Hingegen ist es überraschenderweise eine offene Frage, also ein ungelöstes mathematisches Problem, ob der oben gezeigte „gierige“ Algorithmus immer so eine Lösung findet!

Zum Schluss noch ein verbesserter Algorithmus für die ursprüngliche Fragestellung. Der obige Algorithmus hat nämlich das Problem, dass die Nenner sehr groß werden können. Probieren Sie es z.B. mal mit $5/121$ aus:

$$\frac{5}{121} = \frac{1}{25} + \frac{1}{757} + \frac{1}{763\,309} + \frac{1}{873\,960\,180\,913} + \frac{1}{1\,527\,612\,795\,642\,093\,418\,846\,225}$$

Der folgende Algorithmus findet hingegen garantiert immer eine Lösung, bei der der größte Nenner kleiner als das Quadrat des ursprünglichen Nenners ist. Zunächst schreiben wir eine Funktion, die die sogenannte *Farey-Folge* für die Zahl n berechnet. Damit meint man die sortierte Liste aller gekürzten Brüche zwischen 0 und 1, deren Nenner nicht größer als n sind.

```
def Farey (n):
    a, b, c, d = 0, 1,  1 , n
    L = [Fraction(a, b)]
    while c <= n:
        k = int((n + b) / d)
        a, b, c, d = c, d, k * c - a, k * d - b
        L.append(Fraction(a, b))
    return L
```

Mit deren Hilfe kann man nun den Algorithmus schreiben, den der Amerikaner Michael N. Bleicher 1972 vorgeschlagen hat:

```
def Bleicher (x):
    L = []
    print(str(x) + " = ", end = "")
    while (x > 0):
        y1 = 0
        q = x.denominator
        for y2 in Farey(q)[1:]:
            if y2 >= x:
                break
            y1 = y2
```

```
        u = Fraction(1, q * y1.denominator)
        L.append(u)
        x -= u
    print(" + ".join(reversed(list(map(str, L)))))
```

Dieser liefert eine deutlich bessere Lösung für $5/121$:[107]

$$\frac{5}{121} = \frac{1}{25} + \frac{1}{1\,225} + \frac{1}{3\,577} + \frac{1}{7\,081} + \frac{1}{11\,737}$$

Lösung P9: Siehe Lösung zu Aufgabe P8.

Lösung P10: Eine einfache Lösung sieht so aus:

```
def computePi (n):
    s = 1
    num = 1
    den = 1
    for k in range(1, n):
        num *= k
        den *= 2 * k + 1
        s += Fraction(num, den)
    return 2 * s
```

Man kann die Funktion allerdings signifikant schneller machen, wenn man die zu berechnende Summe durch Ausklammern zu einem Produkt macht. Hier z.B. die entsprechende Umformung für die ersten fünf Summanden:

$$\begin{aligned}
&\frac{1}{1} + \frac{1}{1\cdot 3} + \frac{1\cdot 2}{1\cdot 3\cdot 5} + \frac{1\cdot 2\cdot 3}{1\cdot 3\cdot 5\cdot 7} + \frac{1\cdot 2\cdot 3\cdot 4}{1\cdot 3\cdot 5\cdot 7\cdot 9} \\
&= 1 + \frac{1}{3}\cdot\left(1 + \frac{2}{5} + \frac{2\cdot 3}{5\cdot 7} + \frac{2\cdot 3\cdot 4}{5\cdot 7\cdot 9}\right) \\
&= 1 + \frac{1}{3}\cdot\left(1 + \frac{2}{5}\cdot\left(1 + \frac{3}{7} + \frac{3\cdot 4}{7\cdot 9}\right)\right) \\
&= 1 + \frac{1}{3}\cdot\left(1 + \frac{2}{5}\cdot\left(1 + \frac{3}{7}\cdot\left(1 + \frac{4}{9}\right)\right)\right)
\end{aligned}$$

Wenn man nun „von rechts nach links“ arbeitet, Zähler und Nenner getrennt führt und erst am Ende in einen Bruch umwandelt (wodurch man das Kürzen zwischendurch spart), erhält man den folgenden Code:

```
def computePi (n):
    n -= 1
    d = 2 * n + 1
    num = n
    den = d
    while True:
```

[107] Der Algorithmus garantiert, dass der größte Nenner kleiner als $121^2 = 14\,641$ ist.

```
        num += den                          # add 1
        if n <= 1:
            return Fraction(2 * num, den)
        n -= 1
        d -= 2
        num *= n
        den *= d
```

In der folgenden Tabelle wird das Ergebnis jeweils auf k Stellen nach dem Komma gerundet angezeigt. Ziffern, die nicht korrekt sind, sind rot markiert.

k	n	`computePi(`n`)`
10	21	3.141592299
	27	3.141592649
	32	3.141592653
	33	3.141592654
20	54	3.14159265358979321 21
	56	3.14159265358979323 20
	64	3.14159265358979323 84
	65	3.14159265358979323 85
30	87	3.14159265358979323846264338085
	91	3.14159265358979323846264338313
	96	3.14159265358979323846264338327
	97	3.14159265358979323846264338328

Die Approximationsformel, die wir benutzt haben, geht ursprünglich auf Newton zurück, benutzt aber eine Transformation von Euler, um die Konvergenzgeschwindigkeit zu erhöhen. Der aktuelle Rekord für die Anzahl der berechneten Nachkommastellen liegt bei mehreren Billionen. Für solche Zwecke werden allerdings (komplizierte) Formeln benutzt, die wesentlich schneller konvergieren. Mehr dazu in Kapitel 52.

Lösung P11: Man könnte es z.B. so machen:

```
from fractions import Fraction

def frac (k, n):
    return k if k == n else k + Fraction(k, frac(k+1, n))

def computeE (n):
    return 2 if n == 0 else 2 + Fraction(1, frac(1, n))
```

Dabei ist `frac` eine rekursive Hilfsfunktion, die den Bruch

$$\cfrac{1}{1+\cfrac{1}{2+\cfrac{2}{3+\cfrac{3}{\ddots+\cfrac{\ddots}{(n-1)+\cfrac{n-1}{n}}}}}}$$

berechnet, wenn sie mit den Argumenten 1 und n aufgerufen wird. (Das ist der blaue Teil in Gleichung (P.1).)

In der folgenden Tabelle wird das mit n Brüchen berechnete Ergebnis jeweils auf k Stellen nach dem Komma gerundet angezeigt. Ziffern, die nicht korrekt sind, sind rot markiert.

k	n	`computeE(`n`)`
10	8	2.718281658
	9	2.718281843
	11	2.718281829
	12	2.718281828
20	16	2.71828182845904517 75
	17	2.71828182845904523 83
	18	2.71828182845904523 52
	19	2.71828182845904523 54
30	23	2.71828182845904523536028748902
	24	2.71828182845904523536028747070
	25	2.71828182845904523536028747138
	26	2.71828182845904523536028747135

Lösung P12: Man muss relativ viele Kombinationen betrachten. Daher sollten Sie nicht erwarten, dass man die Aufgabe mit einem „eleganten Zweizeiler“ lösen kann. Hier mein Versuch, eine halbwegs kompakte Lösung zu programmieren:

```
def SetToInterval (I):
    if type(I) is set:
        if len(I) == 0:
            return [0, 0, False, False]
        if len(I) == 1:
            (x,) = I
            return [x, x, True, True]
    return I

def IntervalToSet (I):
    if type(I) is set:
```

```
        return I
    if I[0] > I[1]:
        return set()
    if I[0] == I[1]:
        if I[2] and I[3]:
            return {I[0]}
        return set()
    return I

def Intersection (A, B):
    A, B = map(SetToInterval, [A, B])
    Left = A[0]
    if A[0] == B[0]:
        LeftIn = (A[2] and B[2])
    elif A[0] > B[0]:
        LeftIn = A[2]
    else:
        Left = B[0]
        LeftIn = B[2]
    Right = A[1]
    if A[1] == B[1]:
        RightIn = (A[3] and B[3])
    elif A[1] < B[1]:
        RightIn = A[3]
    else:
        Right = B[1]
        RightIn = B[3]
    return IntervalToSet([Left, Right, LeftIn, RightIn])

def Union (A, B):
    if IntervalToSet(A) == set():
        return IntervalToSet(B)
    if IntervalToSet(B) == set():
        return IntervalToSet(A)
    A, B = map(SetToInterval, [A, B])
    if A[0] > B[0]:
        A, B = B, A
    if A[1] < B[0] or (A[1] == B[0]
                       and not (A[3] or B[2])):
        return False
    LeftIn = (A[2] or B[2]) if A[0] == B[0] else A[2]
    Right = A[1]
    if A[1] == B[1]:
        RightIn = (A[3] or B[3])
```

```
    elif A[1] > B[1]:
        RightIn = A[3]
    else:
        Right = B[1]
        RightIn = B[3]
    return IntervalToSet([A[0], Right, LeftIn, RightIn])
```

Lösung P13: Meine Lösung sieht so aus:

```
def plot (f, a, b, n = 100):
    step = (b - a) / (n - 1)
    X = [a + k * step for k in range(n)]
    Y = [f(x) for x in X]
    plt.plot(X, Y)
```

Beachten Sie, dass es sich um einen klassischen *Off-by-one-Error* handelt, wenn Sie durch n statt durch $n-1$ dividieren.

Lösung P14: Wenn man das Konzept von anonymen Funktionen verstanden hat, ist es ganz einfach:

```
def diff (f, h):
    return lambda x: (f(x + h) - f(x - h)) / (2 * h)
```

Lösung P15: Hier handelt es sich offenbar um das auf Seite 475 besprochene *Rucksackproblem*. Meine Lösung sieht so aus:[108]

```
def powerSetHelper (A):
    if A == set():
        yield set()
    else:
        a = A.pop()
        for X in powerSetHelper(A):
            yield X
            yield X | {a}

def powerSet (A):
    return powerSetHelper(A.copy())

def pack (objects, maxWeight):
    bestValue = 0
    for someObjects in powerSet(objects):
```

[108] Beachten Sie, dass `powerSetHelper` als Generator realisiert ist. Das ist für große Mengen deutlich effizienter als die Alternative, tatsächlich alle Teilmengen erst zu erzeugen und dann irgendwo abzuspeichern.

```
        weight = 0
        value = 0
        for w, v in someObjects:
            weight += w
            value += v
            if weight > maxWeight:
                break
        if weight <= maxWeight:
            if value > bestValue:
                bestValue = value
    return bestValue
```

Wenn Sie mit `measure` messen, werden Sie feststellen, dass sich mit jedem weiteren Objekt die Laufzeit ungefähr verdoppelt. (Das liegt natürlich daran, dass sich die Anzahl der Teilmengen in jedem Schritt verdoppelt.) Wir haben es hier also mit exponentiellem Laufzeitverhalten zu tun.

Es gibt effizientere Lösungen für diese Aufgabe. Allerdings ist bisher kein Algorithmus bekannt, der das Problem immer löst und kein exponentielles Laufzeitverhalten hat. Sollten Sie einen finden, dann haben Sie eines der *Millennium-Probleme* gelöst und können sich das Preisgeld in Höhe von einer Million Dollar abholen!

Lösung P16: Wenn Sie die folgende Funktion in der Form `draw(c)` aufrufen, wobei `c` ein `Canvas`-Objekt ist, dann wird die Figur gezeichnet. Mit `draw(c, True)` erhalten Sie alternativ eine Animation.

```
from matrices import Matrix
from vectors import Vector
from canvas import Canvas
from time import sleep
from math import cos, sin, pi

def rotatePoints (angle, points):
    c = cos(angle)
    s = sin(angle)
    M = Matrix([[c, -s], [s, c]])
    return map(lambda p: M * p, points)

def rotatePoint (angle, point):
    return list(rotatePoints(angle, [point]))[0]

def translatePoints (vector, points):
    return map(lambda p: p + vector, points)

def drawPolygon (canvas, points):
```

```
    for p1, p2 in zip(points, points[1:] + points[:1]):
        canvas.drawSegment(p1, p2)

def draw (c, animate = False):
    squarePoints = [Vector(x,y) for x, y
                    in [(2,2), (2,-2), (-2,-2), (-2,2)]]
    innerAngle = 0
    outerAngle = 0
    shift = Vector(6,0)
    maxAngle = 4 * pi if animate else 2 * pi
    c.clear()
    while outerAngle <= maxAngle:
        if animate and outerAngle > 0:
            sleep(0.04)
            c.clear()
        innerAngle = innerAngle - pi / 20
        outerAngle = outerAngle + pi / 40
        drawPolygon(c, list(translatePoints(
                               rotatePoint(outerAngle,
                                           shift),
                               rotatePoints(innerAngle,
                                            squarePoints))))
```

Lösung P17: Hier eine Beispiellösung:

```
from vectors import Vector
from canvas import Canvas

def drawTriangle (c, L, color):
    for i in range(3):
    c.drawSegment(L[i], L[(i + 1) % 3], color = color)

def draw (c, L):
    c.clear()
    c.drawGrid()
    c.drawCircle([0, 0], 10, color = "blue")
    S = sum(L, Vector(0, 0)) / 3
    d = max((V - S).norm() for V in L)
    L2 = [10 / d * (V - S) for V in L]
    drawTriangle(c, L2, "red")
    for i in range(3):
        c.drawSegment(L2[i],
                      (L2[(i+1)%3] + L2[(i+2)%3]) / 2,
```

```
                    color = "orange")
    drawTriangle(c, L, "green")
```

Lösung P18: Meine Lösung:

```
def countGaussians (n):
    count = 0
    x = 0
    while x*x <= n:
        y = 0
        while y*y < n:
            if x*x + y*y == n:
                count += 1
                break
            y += 1
        x += 1
    return 4 * count
```

Die Funktion arbeitet nur im oberen rechten Quadranten, weil die Punkte offenbar symmetrisch zu den Achsen verteilt sind. Die Funktion ist nicht besonders effizient, weil sie viel zu viele Punkte testet.

Um ein Muster erkennen zu können, kann man z.B. so anfangen:

```
for k in range(3, 200, 2):
    print(countGaussians(k) if isprime(k) else '-', end='')
```

Man muss das nicht unbedingt sehen, aber ich meinte das folgende Muster: Zu einer ungeraden Primzahl gehören entweder 0 oder 8 Punkte. 8 Punkte sind es genau dann, wenn die Primzahl von der Form $4k + 1$ für ein $k \in \mathbb{N}$ ist. Das besagt der *Zwei-Quadrate-Satz* von Fermat.

Lösung P19: Meine Lösung sieht so aus:

```
import matplotlib.pyplot as plt
from math import log, sin

plt.figure(figsize=(15, 4))
R = range(1, 10001)
plt.scatter([log(x) for x in R], [sin(x) for x in R], s = 1)
```

Lösung P20: Zum Beispiel so:

```
from plot import *
from math import exp, pi, cos, sin
```

```
plotSurface3D(lambda u, v: (u, exp(-abs(u)) * cos(v),
                               exp(-abs(u)) * sin(v)),
              [-2, 2], [0, 2 * pi])
```

Lösung P21: Das kann man recht kurz halten:

```
from random import uniform
from math import sin, exp, pi, e, sqrt, ceil

def randomIntegration (f, a, b, n):
    M = 1.1 * max(f(uniform(a, b)) for i in range(ceil(sqrt(n))))
    c = 0
    for i in range(n):
        if uniform(0, M) <= f(uniform(a, b)):
            c += 1
    return c / n * (b - a) * M
```

Lösung P22: Das ist schon die ganze Lösung:

```
from scipy.integrate import quad

def p (f, n):
    Q = lambda x: (1 - x*x)**n
    I = quad(Q, -1, 1)[0]
    return lambda x: quad(lambda t: f(t) / I * Q(t - x), 0, 1)[0]
```

Zum Ausprobieren bin ich so vorgegangen:

```
from plot import *
from math import exp, sin, pi

f = lambda x: sin(2*x*pi)
plotFunc2D([lambda x: f(x) + 0.2, lambda x: f(x) - 0.2,
            p(f, 50)], [0, 1])

g = lambda x: exp(x) - 1 - x * (exp(1) - 1)
plotFunc2D([lambda x: g(x) + 0.02, lambda x: g(x) - 0.02,
            p(g, 100)], [0, 1])
```

An dieser Stelle kann der Approximationssatz natürlich nicht bewiesen werden. Ich will aber zumindest kurz begründen, warum die von `p` zurückgegebene Funktion ein Polynom ist. Dafür wählen wir als Beispiel den festen Wert $n = 2$. Zunächst können wir den Term $q(t-x)$ ausrechnen und als Polynom in der Variablen x auffassen:

$$\begin{aligned} q(t-x) &= (1-2t^2+t^4) + (4t-4t^3)x + (-2+6t^2)x^2 - 4tx^3 + x^4 \\ &= a_0(t) + a_1(t)x + a_2(t)x^2 + a_3(t)x^3 + a_4(t)x^4 \end{aligned}$$

Terme wie $a_3(t) = -4t$ sind unter diesem Blickwinkel einfach Koeffizienten des Polynoms. Nun schreiben wir das Integral entsprechend um, indem wir die obigen Summanden von $q(t-x)$ auseinanderziehen:

$$\begin{aligned}\int_0^1 I^{-1}f(t)q(t-x)\,\mathrm{d}t &= \int_0^1 I^{-1}f(t)a_0(t)\,\mathrm{d}t + \cdots + \int_0^1 I^{-1}f(t)a_4(t)x^4\,\mathrm{d}t \\ &= \int_0^1 b_0(t)\,\mathrm{d}t + \int_0^1 b_1(t)x\,\mathrm{d}t + \cdots + \int_0^1 b_4(t)x^4\,\mathrm{d}t \\ &= \left(\int_0^1 b_0(t)\,\mathrm{d}t\right) + \left(\int_0^1 b_1(t)\,\mathrm{d}t\right)x + \cdots + \left(\int_0^1 b_4(t)\,\mathrm{d}t\right)x^4\end{aligned}$$

In der zweiten Zeile haben wir $b_i(t)$ als Abkürzung für $I^{-1}f(t)a_i(t)$ geschrieben. Der „Trick“ ist nun, dass wir die Potenzen von x aus den Integralen herausziehen können, da sie nicht von der Integrationsvariable t abhängen, sondern aus Sicht des Integrals lediglich konstante Faktoren sind. Damit haben wir ein Polynom (vierten Grades) in x, dessen Koeffizienten bestimmte Integrale sind, die wir nur einmal ausrechnen (oder zumindest gut genug approximieren) müssen.

Die obige Python-Funktion `p` eignet sich übrigens zum Visualisieren der Approximation, liefert aber kein Polynom, das man als effizient berechenbaren Ersatz für f verwenden könnte. Dafür müsste man die obigen Koeffizienten tatsächlich ausrechnen und das Polynom dann z.B. mit dem Horner-Schema implementieren.

Falls Sie mit dieser Methode wirklich mal Funktionen approximieren wollen, hier noch zwei weitere Hinweise:

- Was machen Sie, wenn Ihre Funktion f auf $[0,1]$ definiert ist, aber $f(0)$ und $f(1)$ irgendwelche beliebigen Werte sind? Definieren Sie eine Funktion g wie folgt:

  $$g(x) = f(x) - f(0) - x\cdot(f(1) - f(0))$$

  Für g gilt $g(1) = g(0) = 0$. (Rechnen Sie nach!) Approximieren Sie nun g durch ein Polynom r. g wurde so definiert, dass $g(x) - f(x)$ ein Polynom ist, also ist $r(x) - (g(x) - f(x))$ ein Polynom, das f approximiert.
- Was machen Sie, wenn Ihre Funktion f nicht auf $[0,1]$, sondern auf einem anderen Intervall $[a,b]$ definiert ist? Definieren Sie h auf $[0,1]$ so:

  $$h(x) = f(a + x(b-a))$$

  Approximieren Sie h durch ein Polynom r. Das durch $r((x-a)/(b-a))$ definierte Polynom approximiert dann f auf $[a,b]$.

Lösung P23: Zum Berechnen der Koeffizienten:

```
from sympy import *
x = symbols("x")

g = lambda x: 1 / (1 + x*x)
[diff(g(x), x, k).subs(x, 1)/factorial(k) for k in range(12)]
```

Die Funktion könnte dann so aussehen:

```
def f (x, n):
    x -= 1              # (x-1)
    pow = 1             # (x-1)**0
    S = 0               # Summe (Funktionswert)
    c = 0.5             # Koeffizient
    for k in range(n):
        m = k % 4
        if m == 3:
            c *= -0.5  # Vorzeichenwechsel und Division durch 2
            pow *= x   # (x-1)**k * (x-1)
            continue   # nichts addieren, nächste Schleife
        S += c * pow   # aufsummieren
        pow *= x       # (x-1)**k * (x-1)
        if m == 0:
            c *= -1    # Vorzeichenwechsel
        elif m == 1:
            c *= -0.5  # Vorzeichenwechsel und Division durch 2
    return S
```

Ab etwa $n = 14$ kann ich auf meinem Bildschirm keinen relevanten Unterschied mehr erkennen.

```
from plot import *
plotFunc2D([lambda x: f(x, 14), g], [0, 2], samples=500)
```

Und es sieht so aus, als würde der Konvergenzradius bei etwa 1.5 liegen.

```
plotFunc2D([lambda x: f(x, 20), g], [-1, 2], samples=500)
```

Der tatsächliche Konvergenzradius ist $\sqrt{2}$. Das kann man sich dadurch klarmachen, dass die Funktion bei i und $-$i Pole hat. Der Abstand von 1 zu den beiden Polen ist jeweils $\sqrt{2}$. (Mehr dazu am Ende von Kapitel 36.)

Lösung P24: Mein Lösungsvorschlag:

```
from canvas import Canvas
from vectors import Vector
from random import uniform
from math import sqrt, pi, acos, atan2

r = 9.5
a = 3*r/sqrt(3)

def circle (c):
    c.clear()
```

```
    c.fillCircle((0,0), r, color="#ccccff")
    L = [Vector.polar(r, phi)
         for phi in [0.3, 0.3+2*pi/3, 0.3+4*pi/3]]
    c.fillPolygon(L, color="#ccffcc")

def helper (c, count, x, y, midpoints, col):
    long = (x - y).norm() >= a
    if long:
        count += 1
    if not c:
        return count
    if midpoints:
        c.drawPoint((x + y)/2, radius = 1)
    else:
        if not col:
            col = "red" if long else "black"
        c.drawSegment(x, y, width = 1, color = col)
    return count

def bertrand1 (n, canvas = None, midpoints = False, col = None):
    if canvas:
        circle(canvas)
    count = 0
    for i in range(n):
        x = Vector.polar(r, uniform(0, 2*pi))
        y = Vector.polar(r, uniform(0, 2*pi))
        count = helper(canvas, count, x, y, midpoints, col)
    if n > 0:
        return count / n

def bertrand2 (n, canvas = None, midpoints = False, col = None):
    if canvas:
        circle(canvas)
    count = 0
    for i in range(n):
        phi = uniform(0, 2*pi)
        R = uniform(0, r)
        psi = acos(R / r)
        x = Vector.polar(r, phi+psi)
        y = Vector.polar(r, phi-psi)
        count = helper(canvas, count, x, y, midpoints, col)
    if n > 0:
        return count / n
```

```
def bertrand3 (n, canvas = None, midpoints = False, col = None):
    if canvas:
        circle(canvas)
    count = 0
    for i in range(n):
        phi = uniform(0, 2*pi)
        R = sqrt(uniform(0, r*r))
        psi = acos(R / r)
        x = Vector.polar(r, phi+psi)
        y = Vector.polar(r, phi-psi)
        count = helper(canvas, count, x, y, midpoints, col)
    if n > 0:
        return count / n
```

Man kann die drei Funktionen auf mehrere Arten aufrufen. (Dabei soll `c` ein `Canvas`-Objekt sein.)

```
# Generiere 5000 Sehnen und gib den Anteil der "langen" Sehnen an
bertrand1(5000)
# Generiere 50 Sehnen und zeichne sie
bertrand1(50, c)
# Wie eben, aber zeichne alle Sehnen mit derselben Farbe
bertrand1(50, c, col="black")
# Generiere 500 Sehnen und zeichne ihre Mittelpunkte
bertrand1(500, c, midpoints=True)
```

Lösung P36: Aufgabe 1137 deutet an, wie Ihr Programm aussehen sollte. Zudem liefert sie auch eine Begründung dafür, warum sich e ergibt.

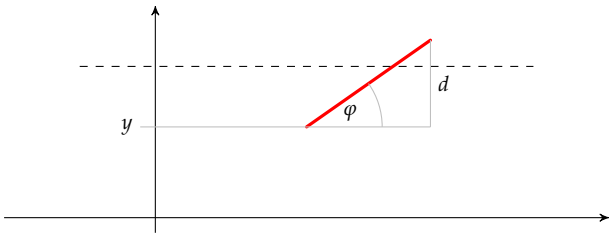
Lösung P37: Ohne Rücksicht auf Effizienz kann man das z.B. so realisieren:

```
def ones (name, k):
    with open(name, "rb") as infile:
        block = infile.read(k)
        while block:
            num = int.from_bytes(block, "little")
            yield bin(num).count("1")
            block = infile.read(k)

from matplotlib.pyplot import bar

def fileDist (name, k):
    C = [0] * (k * 8 + 1)
    for c in ones(name, k):
```

```
        C[c] += 1
    bar(list(range(k * 8 + 1)), C)
```

Lösung P38: Ich habe versucht, meine Lösung möglichst minimal zu halten. Die Trennlinien der Dielen sind bei mir zur x-Achse parallele Geraden der Form $y = k$ für ganze Zahlen k. Der Abstand zweier Dielen ist also 1. Für die Fragestellung der Aufgabe ist die horizontale Lage der Nadel offenbar irrelevant. Es wird also nur zufällig eine y-Koordinate ausgewählt. Diese soll ein Ende der Nadel beschreiben. Hier ist der tatsächliche Ort ebenfalls unerheblich; daher wird ein Wert zwischen 0 und 1 gewählt.

Dann wird der Winkel φ der Nadel zur x-Achse zufällig ermittelt. Man überlegt sich leicht, dass man sich auf den Bereich zwischen 0 und $\pi/2$ beschränken kann, weil die anderen drei Viertel des Kreises zu identischen Ergebnissen führen. Im Koordinatensystem zeigt die Nadel dann quasi nach „rechts oben“ und die Trennlinie $y = 1$ wird durchschnitten, wenn die Summe aus y und $d = \sin(\varphi)$ größer als 1 ist.



Das ergibt das folgende Programm:

```
from random import uniform
from math import pi, sin

def buffon ():
    y = uniform(0, 1)
    d = sin(uniform(0, pi/2))
    return y + d > 1

def testBuffon (n):
    return sum(buffon() for i in range(n)) / n
```

Ein Aufruf wie `testBuffon(1000000)` sollte ein Ergebnis in der Nähe von $2/\pi$ liefern. Es handelt sich hierbei um das sogenannte *Buffonsche Nadelproblem*. Mehr dazu im Video, das neben der Aufgabe verlinkt ist.