Aufgabe 1

Berechne in der Shell oder mit einem Programm (`math-01.py`) den Wert des folgenden Ausdrucks und runde ihn auf 2 Stellen nach dem Komma bzw. nach dem Dezimalpunkt.

$$\left(\frac{93.46 - 86.39}{97.42 + 48.64} - 8.93\right)^2$$

Aufgabe 2

Berechne in der Shell oder mit einem Programm (`math-02.py`) den Wert des folgenden Ausdrucks und runde ihn auf 2 Stellen nach dem Komma bzw. nach dem Dezimalpunkt.

$$\sqrt{60.37 + 6.41 \cdot \frac{80.46}{69.47 + 40.04}}$$

Aufgabe 3

Berechne in der Shell oder mit einem Programm (`math-03.py`) den Wert des folgenden Ausdrucks und runde ihn auf 2 Stellen nach dem Komma bzw. nach dem Dezimalpunkt.

$$\exp\left(-\frac{1}{2}\left(\frac{18.65 - 19.74}{17.15}\right)^2\right)$$

Hinweis: $\exp(x)$ ist eine andere Schreibweise für e^x mit $\mathrm{e} = 2.7181828\ldots$.

Aufgabe 4

Berechne in der Shell oder mit einem Programm (`math-04.py`) den Wert des folgenden Ausdrucks und runde ihn auf 2 Stellen nach dem Komma bzw. nach dem Dezimalpunkt.

$$-\frac{3}{4}\ln\left(1 - \frac{4}{3} \cdot 0.64\right)$$

Aufgabe 5

Rechne die ganze Zahl 6409 in der Shell oder mit einem Programm (`math-05.py`) vom Dezimal- ins Binärsystem um.

Aufgabe 6

Rechne die Binärzahl 1101110111_2 in der Shell oder mit einem Programm (`math-06.py`) ins Dezimalsystem um.

Aufgabe 7

Rechne die ganze Zahl 2020 in der Shell oder mit einem Programm (`math-07.py`) vom Dezimal- ins Hexadezimalsystem um.

Aufgabe 8

Rechne die Hexadezimalzahl $\texttt{D02C0FA7}_{16}$ in der Shell oder mit einem Programm (`math-08.py`) ins Dezimalsystem um.

Aufgabe 9 (leicht)

Schreibe ein Programm `reverse_list.py`, das alle Elemente einer gegebenen Liste `A` in umgekehrter Reihenfolge in der zu Beginn leeren Liste `B` speichert. Danach sind die Listen `A` und `B` auszugeben.

Aufgabe 10 (leicht)

Schreibe ein Programm `number_to_list.py`, das den Benutzer so lange auffordert einen Zahlstring einzugeben, bis die Eingabe der leere String ist.

Jede eingegebenen Zahl soll ans Ende einer Liste `L` anfgefügt werden, die zu Beginn leer ist. Nach der letzten Eingabe soll die Liste ausgegeben werden.

Aufgabe 11 (leicht)

Schreibe ein Modul `gendern.py` mit der Funktion `gendern(wort, typ)`, die als Argumente die Zeichenkette `wort` und das Zeichen `typ` entgegennimmt und `wort` mit dem Suffix (der Endung) `*in` zurückgibt, wenn für `typ` den Wert `'s'` (singular) hat oder `wort` mit dem Suffix `*innen` zurückgibt, wenn `typ` den Wert `'p'` (plural) hat. Wird ein anderer Typ als `'s'` oder `'t'` verwendet, soll eine Fehlermeldung ausgegeben werden. *Beispiele:*

- `gendern('Radfahrer', 'p')` Ausgabe: `'Radfahrer*innen'`
- `gendern('Pilot', 's')` Ausgabe: `'Pilot*in'`
- `gendern('Schüler', 'x')`

  Ausgabe: `'Gib 's' (singular) oder 'p' (plural) statt 'x' ein.`

Aufgabe 12 (leicht)

Erstelle ein Modul `umrechnung.py`, das

1. die Zahlen in der Datei `werte_in_yard.txt` zeilenweise einliest,
2. dann den Zeilentext in eine Gleitkommazahl umwandelt,
3. den Wert mit der Einheit *Yard* in *Meter* umrechnet und schliesslich
4. die Masszahl (in Metern) in der Datei `werte_in_metern.txt` speichert.

Informiere dich im Internet, mit welchem Faktor von Yard zu Metern umgerechnet wird.

Aufgabe 13 (leicht)

Erstelle ein Modul `summe_datei.py`, das die Summe aller Zahlen in der Datei `numbers.txt` berechnet und auf der Shell ausgibt. *Hinweise:*

- Jede Zeile enthält genau eine Gleitkommazahl
- Der Codeblock

  ```
  dd = open(<dateiname>, mode='r')
  for zeile in dd:
      <verarbeite zeile>
  dd.close()
  ```

  Erzeugt einen Datei-Deskriptor zum Lesen einer Datei, welche so mit einer `for`-Schleife zeilenweise durchlaufen und verarbeitet werden kann.
- Da es sich um eine Textdatei handelt, werden auch Zahlen als Text eingelesen. Deshalb müssen sie vorher mit `float(...)` in eine Gleitkommazahl umgewandelt werden, wenn man sie addieren will.
- Zur Kontrolle: Die Summe sollte 236006206.072998 betragen.

Zusatzaufgabe: Die Datei `numbers-dirty.txt` enthält leere Zeilen oder Zeilen die Kommentare enthalten. so, dass `float(zeile)` das Programm mit einem `ValueError` stoppt. Ergänze dein Programm um den folgenden `try-except`-Block, damit dein Programm auch mit dieser Art von Daten umgehen kann.

```
try:
    <verarbeite zeile>
except ValueError:
    print('Überspringe Zeile ...')
```

Im `try`-Block wird versucht, eine Anweisungsblock auszuführen. Falls dabei ein Fehler entsteht, stoppt das Programm nicht sondern führt den `except`-Block aus, falls ein `ValueError` auftritt.

Aufgabe 14 (leicht)

Schreibe ein Programm mit dem Namen `mean.py`, das den Mittelwert einer gegebenen Liste `L` von Zahlen berechnet und ausgibt.

Aufgabe 15 (leicht)

Schreibe ein Programm `find_max.py`, das *ohne* die Funktion `max()` das Maxium `m` der Werte in einer gegebenen Liste `L` findet und am Schluss dieses Maximum und die Liste auf der Shell ausgibt.

Aufgabe 16 (mittel)

Erstelle ein Modul `ziffernhaeufigkeit.py`, das eine Zahl als Argument entegennimmt und eine Liste mit den relativen Häufigkeiten der Ziffern 0, 1, 2, . . . , 9 ausgibt. *Hinweise:*

1. Wandle die Zahl mit `str(...)` in eine Zeichenkette um und weise sie der Variablen `numstring` zu.
2. Erzeuge eine Liste `freq` mit 10 Nullen. Wir verwenden hier die Abkürzung des Wortes *Frequenz* als Synonym für die Häufigkeit. Damit ist jedoch nicht der in der Physik verwendete Begriff *Frequenz* gemeint, mit dem die Anzahl Vorgänge *pro Zeiteinheit* gemeint ist.
3. Durchlaufe die Zeichenkette mit `for z in numstring:` zeichenweise und prüfe im Schleifenrumpf mit `z in '0123456789'`, ob das Zeichen `z` eine gültige Ziffer ist. Wenn ja, addiere in der Liste `freq` Eins zum Element an der Position `i`, so dass am Ende an der Position 0 die Anzahl der Nullen, an der Position 1 die Anzahl der Einsen usw. steht. Beachte, dass das Zeichen `z` mit `int(z)` wieder in einen ganzzahligen numerischen Wert umgewandelt werden muss, wenn es als Index der Liste verwendet werden soll.
4. Kontrolle:

   `zeichenhaeufigkeit(9876543210)`

   ⇒ `[0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1]`

   `zeichenhaeufigkeit(121212)`

   ⇒ `[0.0, 0.5, 0.5, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]`

Aufgabe 17 (mittel)

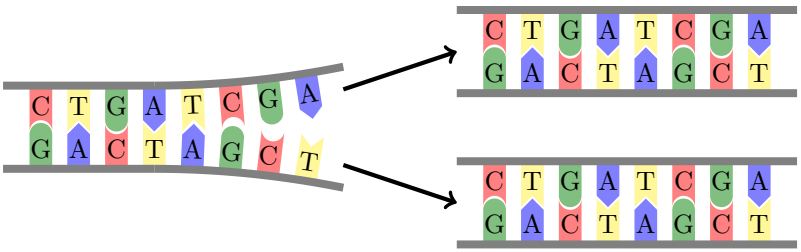
Schreibe ein Modul mit dem Namen `schaltjahr.py`, das eine Funktion `schaltjahr(jahr)` enthält, die anhand der ganzen Zahl `jahr` feststellt, ob es sich um ein Schaltjahr handelt oder nicht und entsprechend `True` oder `False` zurückgibt.

Die derzeit gültigen Regeln für Schaltjahre gelten seit der gregorianische Kalenderreform im Jahr 1582 und lauten in dieser Reihenfolge:

- Alle durch 400 teilbaren Jahreszahlen sind Schaltjahre.
- Alle durch 100 (aber nicht durch 400) teilbaren Jahreszahlen sind keine Schaltjahre.
- Alle durch 4 (aber nicht durch 100) teilbaren Jahreszahlen sind Schaltjahre.
- Alle übrigen Jahre sind keine Schaltjahre.

Aufgabe 18 (mittel)

Die *DNA-Replikation* dient dazu, die Erbinformation einer Zelle oder eines Virus zu vervielfältigen. Dabei werden die beiden komplementären DNA-Stränge, auf denen die Erbinformation durch die Abfolge der Nukleotide *Adenin (A)*, *Cytosin (C)*, *Guanin (G)* und *Thymin (T)* codiert ist, durch biochemische Prozesse aufgetrennt und durch Komplementbildung wieder vervollständigt.



Beachte, dass nur die Nukleotidpaare Cytosin/Guanin sowie Adenin/Thymin jeweils komplementär sind (zueinander passen).

Schreibe ein Programm `dna_complement.py`, das aus einer Liste `N` mit Nukleotiden (`'A'`, `'C'`, `'G'`, `'T'`) eine Liste `K` mit den jeweiligen komplementären Nukleotiden produziert. *Beispiel:*

```
N = ['A', 'C', 'C', 'A', 'T', 'G', 'T']
```

$\Rightarrow$

```
K = ['T', 'G', 'G', 'T', 'A', 'C', 'A']
```

Aufgabe 19 (mittel)

Der Median (Zentralwert) ist ein Wert, der eine der Grösse nach sortierte Liste von Werten in zwei gleich grosse Hälften zerlegt. Er kann auf folgende Weise bestimmt werden:

1. Alle Werte werden (aufsteigend) geordnet.
2. Bei einer ungerade Anzahl Werte ist der Median die mittlere Zahl.
3. Bei einer geraden Anzahl Werte ist der Median das arithmetisches Mittel der beiden mittleren Zahlen.

Erstelle ein Modul `median.py`, mit der Funktion `median(liste)`, die als Argument eine Liste mit Zahlen entgegennimmt, den Median berechnet und zurückgibt.

Aufgabe 20 (mittel/schwierig)

Schreibe ein Modul `kennzahlen.py`, das die Funktion `kennzahlen(`*`dateiname`*`)` enthält, welche folgende Aufgaben erledigt.

1. Es wird eine leere Liste mit dem Namen `daten` erzeugt.
2. Die Datei mit *`dateiname`* wird zum Lesen geöffnet und zeilenweise durchlaufen.
3. Bei jedem Schleifendurchlauf werden mit dem folgenden `try-except`-Block nur Zeilen verarbeitet, die keinen `ValueError` beim Umwandeln der Zeile (Zeichenkette) in eine Gleitkommazahl verursachen.

   ```
   try:
       <Zeile in Gleitkomazahl umwandeln
         und an die Liste anhängen>
   except ValueError:
       <mit print(...) ausgeben, dass eine Zeile
         nicht umgewandelt werden konnte>
   ```

4. Die Datei wird geschlossen.
5. Es wird die Anzahl der Werte ausgegeben.
6. Dann werden folgende Kennzahlen berechnet und ausgegeben:
   (a) Das *Minimum* `x_min` und das *Maximum* `x_max` der Werte in `daten`. Für eine Zahlenliste *`liste`* gib es dafür die Funktionen `min(`*`liste`*`)` und `max(`*`liste`*`)`.
   (b) Die *Spannweite* `R` der Daten: `R` = `x_max` − `x_min`
   (c) Der *empirische Mittelwert* `m` der Daten. Dies ist die Summe der Werte [`sum(`*`liste`*`)`] dividiert durch die Anzahl der Werte [`len(`*`liste`*`)`].
   (d) **Fakultativ:** Die *empirische Varianz* `v` der Daten mit Hilfe dieser Formel:

   $$v = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})^2$$

   Am effizientesten lässt sich diese Berechnung mit einer List-Comprehension durchführen. Der Ausdruck `[(x - m)**2 for x in data]` erzeugt eine neue Liste mit den quadrierte Abweichungen vom Mittelwert. Danach muss nur noch mit `sum(...)` die Summe dieser Werte berechnet und durch `len(daten)-1` dividiert werden.
7. Teste die Funktion mit der Datei `numbers.txt`. Sie sollte diese Ausgaben machen:

   ```
   Anzahl Werte: 100000
   Minimum: 1512.7588
   Maximum: 3152.0331
   Spannweite: 1639.2743
   empirischer Mittelwert: 2360.0620607299797
   empirische Varianz: 39885.90128701353
   ```

Aufgabe 21 (mittel/schwierig)

Schreibe ein Modul `eratosthenes.py`, dass alle Primzahlen kleiner als n mit dem unten beschriebenen *Sieb des Eratosthenes* bestimmt.

Algorithm 1: Das Sieb des Eratostenes

input : Eine natürliche Zahl n
output: Eine Liste sieb der Länge n mit sieb[i]=1, falls i Primzahl ist und 0 sonst

```
sieb[0] ← 0
sieb[1] ← 0
i ← 2
while i < ⌊√n⌋ + 1 do
    if sieb[i] ≠ 0 then
        for j in range(i * i, n, i) do
            sieb[j] ← 0
    i ← i + 1
return sieb
```

Beispiel mit $n = 30$:

Die Liste `sieb` nach dem Schritt 2 im Algorithmus

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

$i = 2$

Für $i = 2$ erhalten die Elemente mit dem Index $j = 4, 6, 8, \ldots, 28$ den Wert 0:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
0	0	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

$i = 3$

Für $i = 3$ erhalten die Elemente mit dem Index $j = 9, 12, 15, \ldots, 27$ den Wert 0:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
0	0	1	1	0	1	0	1	0	0	0	1	0	1	0	0	0	1	0	1	0	0	0	1	0	1	0	0	0	1

$i = 5$

Für $i = 5$ erhält das Element mit dem dem Index $j = 25$ den Wert 0:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
0	0	1	1	0	1	0	1	0	0	0	1	0	1	0	0	0	1	0	1	0	0	0	1	0	0	0	0	0	1

$i = 6$

Wegen $i \geq \lfloor\sqrt{30}\rfloor + 1 = 6$ endet die `while`-Schleife und die Indizes i mit `sieb[i]` $= 1$ sind Primzahlen.

Öffne danach eine Datei mit dem Namen `primzahlen.txt` zum Beschreiben. Durchlaufe die Liste `sieb` und schreibe den Index `i` in die Datei `primzahlen.txt`, wenn `sieb[i]` den Wert `1` hat.

Aufgabe 22 (schwierig)

An einem Kioskautomaten werden Snacks verkauft. Der Käufer füttert den Automaten mit Münzen und erhält dann den gewünschten Snack und eventuell noch Rückgeld, wenn genügend Geld eingeworfen wurden.

Erstelle das Modul `wechselgeld.py` mit der Funktion `wechselgeld(preis, zahlung)`, welche den `preis` des Snacks und die `zahlung` des Käufers (jeweils in Rappen) entgegennimmt und folgenden Wert zurückgibt.

- Den noch nachzuzahlenden Betrag, wenn zu wenig Geld eingeworfen wurde.
- Die Anzahl der auszugebenden Münzen bei mimimaler Münzenzahl.

Damit Rundungsprobleme die Aufgabe nicht noch schwieriger machen, werden alle Beträge in Rappen angegben und sind somit ganzzahlig. In diesem Sinne kann der Automat folgende „Münzen“ (in Rappen) verabeiten: 500, 200, 100, 50, 20, 10, 5.

Beispiel: Der Aufruf `wechselgeld(210, 500)` könnte Folgendes ausgeben:

```
Preis:    210 Rappen
Zahlung:  500 Rappen
Retour:   290 Rappen
1 x 200 Rappen (d.h. Fr. 2.--)
1 x 50 Rappen (d.h. Fr. -.50)
2 x 20 Rappen (d.h. Fr. -.20)
```