

Python

Programmierpraxis (Kurzversion)

21. September 2023

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 1

```
resultat = ((93.46-86.39)/(97.42+48.64)-8.93)**2
```

```
resultat = round(resultat, 2)
```

```
print(resultat) # => 78.88
```

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 2

```
import math
```

```
resultat = math.sqrt(60.37+6.41*80.46/(69.47+40.04))
```

```
resultat = round(resultat, 2)
```

```
print(resultat) # => 8.07
```

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 $e = 2.7181828\dots$

Aufgabe 3

```
import math

resultat = math.exp(-0.5*((18.65-19.74)/17.15))

resultat = round(resultat, 2)

print(resultat) # => 1.0
```

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 4

```
import math

resultat = -3/4*math.log(1-4/3*0.64)

resultat = round(resultat, 2)

print(resultat) # => 1.44
```

Aufgabe 5

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

Aufgabe 5

```
b = bin(6409)
```

```
print(b) # => 0b1100100001001
```

Aufgabe 6

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

Aufgabe 6

```
d = int('1101110111', 2)
```

```
print(d) # => 887
```

Aufgabe 7

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

Aufgabe 7

```
h = hex(2020)
```

```
print(h) # => 0x7E4
```

Aufgabe 8

Rechne die Hexadezimalzahl $D02C0FA7_{16}$ in der Shell oder mit einem Programm (`math-08.py`) ins Dezimalsystem um.

Aufgabe 8

```
d = int('D02C0FA7', 16)
```

```
print(d) # => 3492548519
```

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 9 (leicht)

```
A = [4, 7, 8, 1, 9, 3]
```

```
B = []
```

```
n = len(A)
```

```
for i in range(0, n):
```

```
    B.append(A[n-i-1])
```

```
print(A)
```

```
print(B)
```

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` angefügt werden, die zu Beginn leer ist. Nach der letzten Eingabe soll die Liste ausgegeben werden.

Aufgabe 10 (leicht)

```
L = []  
while True:  
    eingabe = input('Geben sie eine Zahl ein: ')  
    if eingabe == '':  
        break  
    else:  
        zahl = float(eingabe)  
        L.append(zahl)  
  
print(L)
```

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 11 (leicht)

```
def gendern(wort, typ='s'):
    '''Ergänzt 'wort' um ein Gener-Suffix. 's'=singular,
        'p'=plural'''
    if typ == 's':
        return wort + '*in'
    elif typ == 'p':
        return wort + '*innen'
    else:
        print(f'Gib "s" (singular) oder "p" (plural)
              statt {typ} ein')
        return wort + '?'

if __name__ == '__main__': # Testcode

    print(gendern('Radfahrer'))
    print(gendern('Schüler', 's'))
    print(gendern('Pilot', 'p'))
    print(gendern('Akademiker', 'x'))
```

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 12 (leicht)

```
infile = 'werte_in_yard.txt'
outfile = 'werte_in_metern.txt'

def yard_to_meter(x):
    return x*0.9144

dd_in = open(infile, mode='r')
dd_out = open(outfile, mode='w')

for line in dd_in:
    try:
        meter = yard_to_meter(float(line))
        dd_out.write('{0:.3f}\n'.format(meter))
    except ValueError:
        print('Skipping line ...')

dd_in.close()
dd_out.close()
```

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.

Aufgabe 13 (leicht)

```
def summe_datei(dateiname):
    dd = open(dateiname)
    summe = 0
    for zeile in dd:
        summe += float(zeile)
    dd.close()
    return summe

def summe_datei_plus(dateiname):
    dd = open(dateiname)
    summe = 0
    for zeile in dd:
        try:
            summe += float(zeile)
        except ValueError:
            print('Überspringe Zeile ...')

    dd.close()
    return summe
```

Aufgabe 14 (leicht)

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

Aufgabe 14 (leicht)

```
L = [4, 7, 1, 2, 8, 2]
```

```
summe = 0
```

```
for x in L:
```

```
    summe += x
```

```
if len(L) == 0:
```

```
    print('Der Mittelwert der leeren Liste ist nicht  
          definiert.')
```

```
else:
```

```
    mw = summe/len(L)
```

```
    print(mw)
```

Aufgabe 15 (leicht)

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

Aufgabe 15 (leicht)

```
L = [3,7,2,1,8,4,6]
```

```
m = L[0]
```

```
for i in range(1, len(L)):
```

```
    if L[i] > m:
```

```
        m = L[i]
```

```
print(m, L)
```

Aufgabe 16 (mittel)

Erstelle ein Modul `ziffernhaeufigkeit.py`, das eine Zahl als Argument entgegennimmt 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,

Aufgabe 16 (mittel)

```
def ziffernhaeufigkeit(number):  
    freq = [0] * 10  
  
    for c in str(number):  
        if c in '0123456789':  
            freq[int(c)] += 1  
  
    n = sum(freq)  
    for i in range(0, 10):  
        freq[i] /= n  
  
    return freq  
  
print(ziffernhaeufigkeit(12121212))
```

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:

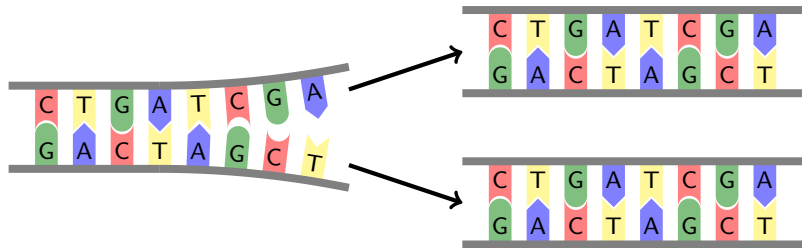
- ▶ Alle durch 400 teilbaren Jahreszahlen sind Schaltjahre.
- ▶ Alle durch 100 (aber nicht durch 400) teilbaren Jahreszahlen sind keine Schaltjahre.
- ▶ Alle durch 4 (aber nicht durch 100) teilbaren Jahreszahlen sind Schaltjahre.
- ▶ Alle übrigen Jahre sind keine Schaltjahre.

Aufgabe 17 (mittel)

```
def schaltjahr(jahr):  
    '''Gibt True zurück, wenn 'jahr' ein Schaltjahr ist  
        und False sonst.'''  
    if jahr % 400 == 0:  
        return True  
    elif jahr % 100 == 0:  
        return False  
    elif jahr % 4 == 0:  
        return True  
    else:  
        return False
```

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']
```

⇒

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

Aufgabe 18 (mittel)

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

```
for x in N:  
    if x == 'A':  
        K.append('T')  
    elif x == 'C':  
        K.append('G')  
    elif x == 'G':  
        K.append('C')  
    elif x == 'T':  
        K.append('A')  
    else:  
        K.append('?')  
        print(f'{x} ist kein gültiges Nukleotid.')
```



```
print('N:', N)  
print('K:', K)
```

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 arithmetische 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 19 (mittel)

```
def median(liste):  
    '''Gibt den Median von 'liste' zurück.'''  
    L = sorted(liste) # erzeuge eine neue Liste  
    n = len(L)  
    if n % 2 == 1:  
        return L[n//2]  
    else:  
        return (L[n//2-1] + L[n//2])/2  
  
if __name__ == '__main__': # Testcode  
  
    L = [8, 7, 4, 9] # => [4, 7, 8, 9] => 7.5  
    print(median(L))  
  
    L = [5, 1, 5, 6, 3] # => [1, 3, 5, 5, 6] => 5  
    print(median(L))
```


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 Gleitkommazahl 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.

Aufgabe 20 (mittel/schwierig)

```
def kennzahlen(dateiname):
    data = []
    dd = open(dateiname, mode='r')
    for zeile in dd:
        try:
            data.append(float(zeile))
        except ValueError:
            print('Überspringe Zeile ...')

    x_min = min(data)
    x_max = max(data)
    R = x_max - x_min
    n = len(data)
    m = sum(data)/n
    v = sum([(x - m)**2 for x in data])/(n-1)

    print('Anzahl Werte: {}'.format(n))
    print('Minimum: {}'.format(x_min))
    print('Maximum: {}'.format(x_max))
    print('Spannweite: {}'.format(R))
```

Aufgabe 21 (mittel/schwierig)

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

Algorithm 1: Das Sieb des Eratosthenes

input : Eine natürliche Zahl n

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

```
1 sieb[0] ← 0
2 sieb[1] ← 0
3  $i \leftarrow 2$ 
4 while  $i < \lfloor \sqrt{n} \rfloor + 1$  do
5     if sieb[i] ≠ 0 then
6         for  $j$  in range(i * i, n, i) do
7             sieb[j] ← 0
8      $i \leftarrow i + 1$ 
9 return sieb
```

Aufgabe 21 (mittel/schwierig)

```
def eratosthenes(n):
    sieb = [1 for i in range(0, n)]
    sieb[0], sieb[1] = 0, 0
    i = 2
    while i < int(n**0.5)+1:
        if i != 0:
            for j in range(i*i, n, i):
                sieb[j] = 0
            i += 1
    return sieb
```

```
fd = open('primzahlen.txt', mode='w')
sieb = eratosthenes(36)
for i in range(0, len(sieb)):
    if sieb[i] == 1:
        fd.write(f'{i}\n')
fd.close()
```

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 minimaler Münzenzahl.

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

Aufgabe 22 (schwierig)

```
def wechselgeld(preis, zahlung):
    '''Gibt die Anzahl der Münzen eines Wechselgelds
        zurück.'''
    muenzen = [500, 200, 100, 50, 20, 10, 5]
    anzahl = []
    n = len(muenzen)
    differenz = zahlung - preis
    if differenz < 0:
        return "Noch {0} Rappen
                nachzahlen!\n".format(-differenz)

    for i in range(0, n):
        anzahl.append(differenz // muenzen[i])
        differenz %= muenzen[i]

    txt = 'Zahlung: {0} Rappen\n'.format(zahlung)
    txt += 'Preis:    {0} Rappen\n'.format(preis)
    txt += 'Retour:   {0} Rappen\n'.format(zahlung-preis)
    for i in range(0,n):
        if anzahl[i] > 0:
```