Python

Lösungen

Sommersemester 2022 - 5CS21-1

Leipzig

Autor: Dr.-Ing. Mike Müller
E-Mail: mmueller@python-academy.de
Twitter: pyacademy
Version: 7.0

Trainer: Dr.-Ing. Mike Müller
E-Mail: mmueller@python-academy.de

Inhaltsverzeichnis

1 Einleitung

2 Syntax

3 Anweisungen und Ausdrücke

3.1 Übungen

3.1.1 Übung 1

Schreiben Sie einen Ausdruck der die Zahlen `8` und `4` addiert und das Ergebnis durch `3` teilt.

```
8 + 4 / 3
```

```
9.333333333333334
```

Weisen Sie dem Ergebnis einen Namen zu.

```
res = 8 + 4 / 3
```

3.1.2 Übung 2

Nutzen Sie die Anweisung zum Import eines Moduls und importieren Sie `os`.

```
import os
```

3.1.3 Übung 3

Lösen sie die Aufgaben 1 und 2 indem Sie den Quelltext in ein Modul speichern und diese auf der Kommandozeile ausführen.

Inhalt der Datei `aufgabe1.py`:

```
res = 8 + 4 / 3
print(res)
```

Auf der Kommandozeile im Verzeichnis eingeben:

```
python aufgabe1.py
```

Ergebnis:

```
9.333333333333334
```

Inhalt der Datei `aufgabe2.py`:

```
import os
print(sys)
```

Auf der Kommandozeile im Verzeichnis eingeben:

```
python aufgabe2.py
```

Ausgabe:

```
<module 'sys' (built-in)>
```

4 Entscheidungen

4.1 Übungen

4.1.1 Übung 1

Schreiben Sie ein Programm, das vom Nutzer die Eingabe einer Zahl verlangt. Vergleichen Sie diese Zahl mit einem Grenzwert und geben Sie aus, ob die Zahl darüber oder darunter liegt bzw. dem Grenzwert entspricht.

Hinweis

Nutzen Sie diesen Code für die Eingabe und die Umwandlung in eine Zahl:

```
eingabe = input('Bitte eine Zahl eingeben')
zahl = float(eingabe)
```

```
zahl = float(input('Bitte eine Zahl eingeben: '))
grenzwert = 100

if zahl < grenzwert:
    print('Die Zahl', zahl, 'ist kleiner als der Grenzwert',
          grenzwert)
elif zahl > grenzwert:
    print('Die Zahl', zahl, 'ist größer als der Grenzwert',
          grenzwert)
else:
    print('Die Zahl', zahl, 'ist gleich dem Grenzwert.')
```

```
Bitte eine Zahl eingeben:  123
```

```
Die Zahl 123.0 ist größer als der Grenzwert 100
```

4.1.2 Übung 2

Schreiben Sie ein zweites Programm, das prüft, ob der eingegebene Wert innerhalb eines vorgegebenen Bereichs liegt. Nutzen Sie dazu (a) boolsche Operatoren und (b) einen einzeiligen Mehrfachvergleich.

(a)

```
untere_grenze = 10
obere_grenze = 100
wert = 50

in_grenzen = untere_grenze <= wert <= obere_grenze

print(in_grenzen)
```

```
True
```

(b)

```
in_grenzen = (untere_grenze <= wert) and (wert <= obere_grenze)

print(in_grenzen)
```

```
True
```

```
print('a if a > b else b')
```

```
a if a > b else b
```

5 Schleifen

5.1 Schleifen mit `for`

5.2 Schleifen mit `while`

5.3 Schleifen vorzeitig beenden

5.4 Schleifendurchläufe überspringen

5.5 Übungen

5.5.1 Übung 1

Schreiben Sie eine Schleife, die die Quadrate der Zahlen 10 bis 20 ausgibt mit Hilfe von `for`. Hinweise: Das Quadrat lässt sich durch Multiplikation, also `x * x` oder Potenzierung mit 2, also `x ** 2` erzeugen.

```
for x in range(10, 21):
    print(x ** 2)
```

```
100
121
144
169
196
225
256
289
324
361
400
```

5.5.2 Übung 2

Schreiben Sie eine Schleife mit `while`, die das Gleiche tut.

```
x = 10
while x <  21:
    print(x ** 2)
    x += 1
```

```
100
121
144
169
196
225
256
289
324
361
400
```

5.5.3 Übung 3

Führen Sie Übungen 1. und 2. am interaktiven Prompt aus. Speichern Sie den Quelltext in einem Modul und führen Sie Ihr Programm auf der Kommandozeile aus.

```
python mein_programm.py
```

5.5.4 Übung 4

Modifizieren Sie Ihre Schleife aus Übung 1. so, dass diese nur jede zweite Zahl ausgibt (beginnend mit der ersten Zahl). Es gibt dafür mehrere Lösungen. Nutzen Sie hier eine `if`-Abfrage und `continue` im Schleifenkörper.

Hinweis: Der Modulo-Operator `%` liefert den Rest einer ganzzahligen Division. Mit der Zahl Zwei als Divisor erhalten wir für alle geraden Zahlen eine Null und für alle ungeraden Zahlen eine Eins:

```
>>> 10 % 2
0
>>> 9 % 2
1
```

```
for x in range(10, 21):
    if x % 2 == 0:
        print(x ** 2)
```

```
100
144
196
256
324
400
```

5.5.5 Übung 5

Lösen Sie Übung 4. ohne `if`-Abfrage im Schleifenkörper, sondern (a) durch Änderung des Schleifenkopfes für die `for`- und durch (b) eine andere Schrittweite der Zählvariable für die `while`-Schleife.

(a)

```
for x in range(10, 21, 2):
    print(x ** 2)
```

```
100
144
196
```

(continues on next page)

(continued from previous page)

```
256
324
400
```

(b)

```
x = 10
while x <  21:
    print(x ** 2)
    x += 2
```

```
100
144
196
256
324
400
```

6 Datentypen

6.1 Statische und dynamische Typisierung

6.2 Starke und schwache Typisierung

6.3 Einfache Datentypen

6.4 Kollektionen

6.5 Übungen

6.5.1 Übung 1

Wandeln Sie `10` in eine Gleitkommazahl und eine komplexe Zahl um.

```
complex(10)
```

```
(10+0j)
```

6.5.2 Übung 2

Wandeln Sie `123.95` in eine Ganzzahl um. Erläutern Sie das Ergebnis.

```
int(123.95)
```

```
123
```

Die Umwandlung in eine Ganzzahl schneidet alle Dezimalstellen ab.

6.5.3 Übung 3

Addieren Sie zu der Zahl eine Billion (eine Eins mit 12 Nullen, in wissenschaftlicher Notation `1e12`) ein Millionstel (in wissenschaftlicher Notation `1e-6`). Erläutern Sie Ihr Ergebnis. Verringern Sie den Größenunterschied der beiden Summanden schrittweise, bis Sie eine Änderung im Ergebnis sehen.

```
1e12 + 1e-6
```

```
1000000000000.0
```

Die Addition hat keinen Effekt, da der Größenunterschied mehr als die von Gleitkommazahlen unterstützen 15 signifikanten Stellen beträgt. Das Verkleinern des zweiten Summanden um eine Größenordnung ändert nichts:

```
1e12 + 1e-5
```

```
1000000000000.0
```

Noch eine Größenordnung weniger ergibt nun endlich einen Effekt:

```
1e12 + 1e-4
```

```
1000000000000.0001
```

6.5.4 Übung 4

Legen sie eine Liste mit 5 Elementen an. Greifen Sie auf einzelne Elemente zu.

```
L = [1, 2, 3, 4, 5]
```

Zugriff auf das erste Element:

```
L[0]
```

```
1
```

und das zweite:

```
L[1]
```

```
2
```

6.5.5 Übung 5

Legen Sie ein Dictionary mit drei Schlüssel-Wert-Paaren an. Greifen Sie auf einzelne Werte mit Hilfe des Schlüssels zu.

```
d = {'x': 10, 'y': 20, 'z': 30}
```

Zugriff:

```
d['x']
```

```
10
```

```
d['z']
```

```
30
```

7 Sequenzen im Detail

7.1 Auf Daten in Sequenzen zugreifen

7.2 Übungen

7.2.1 Übung 1

Erstellen sie ein Tupel mit ca. 8 bis 15 Ganzzahlen, Gleitkommazahlen und Strings.

```
t = tuple(range(2, 15)) + (4.5, 10, 45, 'abc', 'x')
```

```
t
```

```
(2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 4.5, 10, 45, 'abc', 'x')
```

Greifen auf das zweite Element dieses Tupels zu.

```
t[1]
```

```
3
```

Erzeugen Sie ein Teil-Tupel vom zweiten bis zum vorletzten Element.

```
t[1:-1]
```

```
(3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 4.5, 10, 45, 'abc')
```

7.2.2 Übung 2

Erstellen Sie ein Tupel, das dreimal so lang ist wie das aus Übung 1 und die gleichen Elemente dreimal enthält. Verwenden Sie dazu mindestens zwei verschiedene Methoden.

Methode 1

```
print(t + t + t)
```

```
(2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 4.5, 10, 45, 'abc', 'x', 2, 3, 4, 5,␣
↪6, 7, 8, 9, 10, 11, 12, 13, 14, 4.5, 10, 45, 'abc', 'x', 2, 3, 4, 5, 6, 7, 8, 9,␣
↪10, 11, 12, 13, 14, 4.5, 10, 45, 'abc', 'x')
```

Methode 2

```
print(t * 3)
```

```
(2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 4.5, 10, 45, 'abc', 'x', 2, 3, 4, 5,␣
↪6, 7, 8, 9, 10, 11, 12, 13, 14, 4.5, 10, 45, 'abc', 'x', 2, 3, 4, 5, 6, 7, 8, 9,␣
↪10, 11, 12, 13, 14, 4.5, 10, 45, 'abc', 'x')
```

7.2.3 Übung 3

Erzeugen Sie ein neues Tupel mit allen Elementen des Tupels von Übung 2 aber in umgekehrter Reihenfolge mit Hilfe von Slicing.

```
t[::-1]
```

```
('x', 'abc', 45, 10, 4.5, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2)
```

7.2.4 Übung 4

Erstellen Sie ein neues Tupel, das nur jedes zweite Element des Tupels aus Übung 2 enthält.

```
t[::2]
```

```
(2, 4, 6, 8, 10, 12, 14, 10, 'abc')
```

7.2.5 Übung 5

Überprüfen Sie, ob ein bestimmtes Element im Tupel enthalten ist.

```
10 in t
```

```
True
```

```
'abc' in t
```

```
True
```

```
'yxz' in t
```

```
False
```

8 Listen im Detail

8.1 Verändern oder neu?

8.2 Sortieren im Detail

8.3 Die eingebauten Funktionen `zip` und `enumerate`

8.4 List-Comprehensions

8.5 Fortgeschrittene List-Comprehensions

8.6 Übungen

8.6.1 Übung 1

Erstellen sie eine kurze Liste mit mindestens 12 Ganzzahlen.

```
L = list(range(10, 25))
L
```

```
[10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24]
```

Ändern Sie den Wert des ersten Elements.

```
L[0] = 45
L
```

```
[45, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24]
```

8.6.2 Übung 2

Löschen Sie die Elemente von Index 2 bis 6.

```
del L[2:7]
```

Bis 7unter der Annahmen das ” 2 bis 6” die “6” einschließt.

Fügen Sie dann 8 neue Zahlen an Stelle der Elemente von Index 3 bis 5 der neuen Liste in.

```
L[3:6] = range(8)
L
```

```
[45, 11, 17, 0, 1, 2, 3, 4, 5, 6, 7, 21, 22, 23, 24]
```

8.6.3 Übung 3

Hängen Sie 5 neue Zahlen an diese Liste an. Tun Sie dies:

(a) indem Sie jede Zahl einzeln anhängen und

```
L
```

```
[45, 11, 17, 0, 1, 2, 3, 4, 5, 6, 7, 21, 22, 23, 24]
```

```
for x in range(100, 501, 100):
    L.append(x)
```

```
L
```

```
[45, 11, 17, 0, 1, 2, 3, 4, 5, 6, 7, 21, 22, 23, 24, 100, 200, 300, 400, 500]
```

(b) indem Sie alle 5 Zahlen in einer Liste zusammenfassen und mit einer Operation die einzelnen Zahlen anhängen.

```
L.extend(range(100, 501, 100))
```

```
print(L)
```

```
[45, 11, 17, 0, 1, 2, 3, 4, 5, 6, 7, 21, 22, 23, 24, 100, 200, 300, 400, 500, 100,␣
↪200, 300, 400, 500]
```

8.6.4 Übung 4

Entfernen Sie das jeweils letzte Element der Liste. Nutzen Sie dazu drei verschiedene Methoden.

Methode 1

```
del L[-1]
```

Methode 2

```
L.pop()
```

```
400
```

Methode 2a

```
L.pop(-1)
```

```
300
```

Methode 3

```
L[-1:] = []
```

Methode 4

```
L[:] = L[:-1]
```

Falsche Ansätze

```
L.remove(L[-1])
```

Versagt wenn das Element an der letzten Stelle nochmals in der Liste vorkommt.

```
L = L[:-1]
```

Erzeugt eine (flache) Teilkopie der Liste `L` ohne das letzte Element. Die Originalliste wird aber nicht modifiziert. Der Name `L` zeigt jetzt auf die neue Liste. Überprüfung mit `id()`:

```
id(L)
```

```
4674739648
```

```
del L[-1]
```

```
id(L)
```

```
4674739648
```

Die ID hat sich nicht geändert. Dagegen:

```
L = L[:-1]
```

```
id(L)
```

```
4675814720
```

Die ID hat sich geändert.

8.6.5 Übung 5

Kehren Sie die Reihenfolge der Liste selbst um (in place).

```
L
```

```
[45, 11, 17, 0, 1, 2, 3, 4, 5, 6, 7, 21, 22, 23, 24, 100]
```

```
L.reverse()
```

```
L
```

```
[100, 24, 23, 22, 21, 7, 6, 5, 4, 3, 2, 1, 0, 17, 11, 45]
```

8.6.6 Übung 6

Erzeugen Sie eine Liste mit Tupeln mit jeweils zwei Elementen aus zwei vorher bestehenden Listen.

```
L1 = list(range(10, 41, 10))
L2 = list(range(100, 401, 100))
list(zip(L1, L2))
```

```
[(10, 100), (20, 200), (30, 300), (40, 400)]
```

8.6.7 Übung 7

Erzeugen Sie eine neue Liste aus einer bestehenden Liste von Zahlen. Jedes Element der neuen Liste soll das Zehnfache des jeweiligen Elements der alten Liste betragen. Nutzen Sie sowohl die Methode `append`, um die neue Liste aus einer leeren Liste in einer Schleife aufzubauen als auch eine List-Comprehension.

```
L = list(range(2, 11))
L
```

```
[2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Mit `append`:

```
result = []
for x in L:
    result.append(x * 10)
result
```

```
[20, 30, 40, 50, 60, 70, 80, 90, 100]
```

Mit List-Comprehension

```
[x * 10 for x in L]
```

```
[20, 30, 40, 50, 60, 70, 80, 90, 100]
```

8.6.8 Übung 8

Fortgeschritten: Sortieren sie Ihre Liste. Definieren Sie dazu eine Funktion, die mit dem Schlüsselwortargument `key` genutzt werden kann und die Sortierung so modifiziert, dass alle geraden Zahlen vor den ungeraden Zahlen stehen. Hinweis: der Modulo-Operator `%` ergibt den Rest beim ganzzahligen Teilen und kann mit der Zahl `2` bei der Unterscheidung gerader und ungerader Zahlen helfen:

```
10 % 2
```

```
0
```

```
11 % 2
```

```
1
```

```
L = [4, 1, 2, 4, 1, 2, 3, 5, 8, 2]
```

```
sorted(L)
```

```
[1, 1, 2, 2, 2, 3, 4, 4, 5, 8]
```

```
def even_first(value):
    return value % 2

sorted(L, key=even_first)
```

```
[4, 2, 4, 2, 8, 2, 1, 1, 3, 5]
```

Zahlen innerhalb der geraden und geraden Gruppen sortiert:

```
def even_first2(value):
    return value % 2, value

sorted(L, key=even_first2)
```

```
[2, 2, 2, 4, 4, 8, 1, 1, 3, 5]
```

8.6.9 Übung 9

Diskutieren sie die unterschiedlichen Anwendungsfelder für Listen und Tupel.

- Tupel lassen sich als Schlüssel in Dictionarys nutzen, da sie unveränderlich sind. Natürlich dürfen Tupel dann keine Listen oder andere veränderliche Objekte enthalten.
- Tupel sind etwas schneller als Listen. Wenn die Struktur nie geändert werden muss, kann ein Tupel deshalb vorteilhaft sein.
- Listen kommen typischer Weise für viele gleichartige Elemente zum Einsatz. Wenn sich die Elemente unterscheiden und sich mit Namen belegen lassen, sind Tupel meist die bessere Wahl. Zum Beispiel lassen sich die Einträge in einer Tabelle mit den Spalten Name, Vorname und Geburtsdatum besser mit einer Liste abbilden. Also eine Liste für alle Namen, eine Liste für alle Vornamen und eine Liste für alle Geburtsdaten. Für die Abbildung einer Zeile mit den drei Einträgen Name, Vorname und Geburtsdatum einer Person eignet sich ein Tupel besser.

9 Dictionarys im Detail

9.1 Alternative Erzeugung eines Dictionarys

9.2 Übungen

9.2.1 Übung 1

Erzeugen Sie ein Dictionary mit Namen und Telefonnummern.

```
tel = {'Thomas': '+1 456 7893 766',
       'Susan': '04522199',
       'James': None}
```

```
tel['Susan']
```

```
'04522199'
```

```
tel['Thomas']
```

```
'+1 456 7893 766'
```

```
tel['James']
```

Testen Sie, ob bestimmte Namen eingetragen sind.

```
'Susan' in tel
```

```
True
```

```
'Kathrin' in tel
```

```
False
```

Listen Sie alle Telefonnummern auf.

```
tel.values()
```

```
dict_values(['+1 456 7893 766', '04522199', None])
```

```
for number in tel.values():
    if number is not None:
        print(number)
```

```
+1 456 7893 766
04522199
```

Tun Sie das Gleiche für die Namen.

```
tel.keys()
```

```
dict_keys(['Thomas', 'Susan', 'James'])
```

```
for name in tel:
    print(name)
```

```
Thomas
Susan
James
```

9.2.2 Übung 2

Greifen Sie auf nichtexistierende Namen in Ihrem Dictionary aus Übung 1 zu, ohne einen Fehler auszulösen. Geben Sie dabei (a) `None` und (b) die Zahl Null zurück.

(a)

```
tel.get('Magrit')
```

(b)

```
tel.get('Magrit', 0)
```

```
0
```

Modifizieren Sie die Lösung und fügen Sie diesen Vorgabewert nun auch in das Dictionary ein. Hinweis: Diese Aufgaben lassen sich jeweils mit einem Methodenaufruf des Dictionarys lösen.

```
tel
```

```
{'Thomas': '+1 456 7893 766', 'Susan': '04522199', 'James': None}
```

```
tel.setdefault('Magrit', 0)
```

```
0
```

```
tel
```

```
{'Thomas': '+1 456 7893 766', 'Susan': '04522199', 'James': None, 'Magrit': 0}
```

9.2.3 Übung 3

Aktualisieren Sie Ihre Telefonnummern mit einem zweiten Dictionary, das für eine Person im bestehenden "Telefonbuch" eine neue Telefonnummer und einen ganz neuen Eintrag mit Namen und Nummer enthält. Beispiel:

```
neue_nummern = {
    'bestehender Name': '353646',
    'neuer Name': '64467'
}
```

Nutzen Sie dazu zwei Methoden:

- (a) eine Schleife und ändern Sie einzelne Elemente und
- (b) eine dafür nützliche Methode des Dictionarys.

Hinweis: Um den Effekt der Änderungen zu zeigen sollten Sie Ihr Telefonnummern-Dictionary vorher kopieren und die Änderungen an der jeweiligen Kopie vornehmen. Es gibt eine Methode zum Kopieren eines Dictionarys.

Kopie erstellen um die Wirkung beider Methoden (a und b) zu zeigen:

```
tel_copy = tel.copy()
```

Neues Dictionary mit Telefonnummern anlegen:

```
neu_tel = {'James': '353424', 'Vera': '0123 46466 12'}
```

(a)

```
tel
```

```
{'Thomas': '+1 456 7893 766', 'Susan': '04522199', 'James': None, 'Magrit': 0}
```

```
for name, number in neu_tel.items():
    tel[name] = number
tel
```

```
{'Thomas': '+1 456 7893 766',
 'Susan': '04522199',
 'James': '353424',
 'Magrit': 0,
 'Vera': '0123 46466 12'}
```

(b)

```
tel_copy
```

```
{'Thomas': '+1 456 7893 766', 'Susan': '04522199', 'James': None, 'Magrit': 0}
```

```
tel_copy.update(neu_tel)
tel_copy
```

```
{'Thomas': '+1 456 7893 766',
 'Susan': '04522199',
 'James': '353424',
 'Magrit': 0,
 'Vera': '0123 46466 12'}
```

9.2.4 Übung 4

Entfernen Sie Einträge aus Ihrem Telefon-Dictionary für von Ihnen vorgegebene Namen (a) ohne sich die entfernte Telefonnummer anzeigen zu lassen und (b) mit Anzeige der entfernten Telefonnummer.

(a)

```
tel
```

```
{'Thomas': '+1 456 7893 766',
 'Susan': '04522199',
 'James': '353424',
```

(continues on next page)

(continued from previous page)

```
 'Magrit': 0,
 'Vera': '0123 46466 12'}
```

```
del tel['Magrit']
```

```
tel
```

```
{'Thomas': '+1 456 7893 766',
 'Susan': '04522199',
 'James': '353424',
 'Vera': '0123 46466 12'}
```

(b)

```
tel.pop('Thomas')
```

```
'+1 456 7893 766'
```

```
tel
```

```
{'Susan': '04522199', 'James': '353424', 'Vera': '0123 46466 12'}
```

Entfernen Sie eine Telefonnummer, ohne den Namen oder die Telefonnummer vorzugeben. Lassen Sie sich dabei den entfernten Namen und die dazu gehörige Telefonnummer anzeigen.

```
tel
```

```
{'Susan': '04522199', 'James': '353424', 'Vera': '0123 46466 12'}
```

```
tel.popitem()
```

```
('Vera', '0123 46466 12')
```

```
tel
```

```
{'Susan': '04522199', 'James': '353424'}
```

Schönere Anzeige:

```
name, number = tel.popitem()
print(f'Name: {name}, Nummer: {number}')
```

```
Name: James, Nummer: 353424
```

9.2.5 Übung 5

Vergleichen sie die Geschwindigkeit der Suche in einem Dictionary und in einer Liste (siehe Tipp unten zur Zeitmessung). Bauen Sie dazu eine lange Liste und ein großes Dictionary (mit je 100, 1000, 10 000 oder mehr Elementen). Platzieren sie das zu suchende Element in die Mitte der Liste. Im Dictionary wird das zu suchende Element als Schlüssel abgelegt. Prüfen Sie, ob das Element in der Liste bzw. dem Dictionary enthalten ist. Nutzen Sie dafür die für die jeweilige Datenstruktur geeignete Methode. Nehmen sie an, dass Sie die Listenposition des zu prüfenden Elements nicht kennen.

Zeitmessung

In Jupyter Notebooks oder in IPython können Sie die magische Funktion `%timeit` nutzen. Zum Beispiel lässt sich so die Zeit für die Berechnung `1 + 1` messen:

```
%timeit 1 + 1
10.6 ns ± 0.071 ns per loop
(mean ± std. dev. of 7 runs, 100000000 loops each)
```

Als Ergebnis erhalten Sie den Mittelwert und die Standardabweichung für sieben Läufe mit einer großen Zahl von Schleifen (hier: 100.000.000). Die Anzahl der Schleifen bestimmt `%timeit` dynamisch so, dass ein Durchlauf ca. eine Sekunde dauert. Sie warten also ca. 7 Sekunden, bis das Ergebnis erscheint. Hilfe dazu erhalten Sie mit `%timeit?`.
In einer Python-Quelltext-Datei können Sie die Funktion `time.perf_counter_ns()` (wenn nicht verfügbar `time.perf_counter()` für Werte in Sekunden) nutzen, um einen aktuellen Zeitstempel in Nanosekunden zu ermitteln. Durch Differenzbildung zwischen zwei Zeitstempeln lässt sich die dazwischen vergangene Zeit ermitteln. Hier sind zum Beispiel zwischen dem Berechnen von `start` und `end` ca. 6,5 Sekunden vergangen:

```
import time
start = time.perf_counter_ns()
end = time.perf_counter_ns()
print((end - start) / 1e9)
6.5461903677702056
```

Bei Nutzung von `time.perf_counter()` entfällt die Teilung durch `1e9`, da die Einheit des Zeitstempels Sekunden und nicht Nanosekunden, also 1e9 Sekunden, ist.

Zuerst erzeugen wir unsere Datenstrukturen. Wir definieren wir groß dies sein sollen:

```
n = 10
```

Legen das gesuchte Objekt fest:

```
target = 'target'
```

Legen unsere Liste an:

```
L = list(range(n))
```

Bei der Liste die Position des Ziels wichtig. Deshalb legen wir es in Mitte:

```
L[len(L) // 2] = target
```

Hierbei nutzen wir `//`, um bei der Division eine Ganzzahl zu erhalten. Denn die mathematische Division ergibt Gleitkommazahlen:

```
1 / 2
```

```
0.5
```

Wogegen die sogenannte “floor divison” `//`

```
1 / 2
```

```
0.5
```

Damit erhalten wir dies Liste:

```
L
```

```
[0, 1, 2, 3, 4, 'target', 6, 7, 8, 9]
```

Diese nutzen wir, um ein entsprechendes Dictionary zu erzeugen:

```
d = dict.fromkeys(L)
d
```

```
{0: None,
 1: None,
 2: None,
 3: None,
 4: None,
 'target': None,
 6: None,
 7: None,
 8: None,
 9: None}
```

Jetzt können wir die Laufzeit für unsere Liste:

```
%timeit target in L
```

```
119 ns ± 2.25 ns per loop (mean ± std. dev. of 7 runs, 10000000 loops each)
```

und unser Dictionary testen:

```
%timeit target in d
```

```
45.7 ns ± 1.07 ns per loop (mean ± std. dev. of 7 runs, 10000000 loops each)
```

Ein paar Zeilen mit einer Schleife über verschiedene Größen und ein paar formatierte Textausgaben zeigen schön wie sich die Laufzeiten mit der sich verändernden Größe verändern:

```
n_name = 'Anzahl'
ratio_name = 'Ratio'
print(f'{n_name:>8s} {ratio_name:>10s}')
print('-' * 19)
for n in [10, 100, 1000, 10_000, 100_000, 1_000_000]:
    target = 'target'
    L = list(range(n))
    L[len(L) // 2] = target
    d = dict.fromkeys(L)
    time_list = %timeit -o -q target in L
    time_dict = %timeit -o -q target in d
    print(f'{n: 8d} {time_list.average / time_dict.average: 10.2f}')
```

```
  Anzahl      Ratio
-------------------
      10       2.57
     100      17.00
    1000     173.95
   10000    1566.22
  100000   15262.05
 1000000  144726.41
```

Die Option `-p` für `%timeit` unterdrückt die Textausgabe. Die Option `-o` gibt ein `TimeitResult` zurück. Wir können dann dessen Attribut `average` nutzen, um die durchschnittliche Laufzeit zu erhalten. Mit den beiden Werten für `average` können wir das Verhältnis der Laufzeiten ermitteln. Die Suche in der Liste mit einer Million Elementen benötigt also mehr als das 150.000-fache der Zeit der Suche in eine Dictionary der gleichen Größe. Die Suchzeit nach Schlüsseln in einem Dictionary ist unabhängig von dessen Größe. Deshalb ist die Suche auch n sehr großen Dictionarys sehr schnell.

10 Mengen im Detail

10.1 Übungen

10.1.1 Übung 1

Definieren Sie zwei Mengen mit Namen von Personen, sodass zwei Namen sowohl in der einen als auch der anderen Menge vorkommen. Zum Beispiel könnten Sie eine Menge von Personen, die mit den ÖPNV (Bus, Bahn etc.) und eine Gruppe, die mit Fahrrad zu Arbeit fahren erstellen. Dabei sollten einige Personen nur mit ÖPNV, einige nur mit dem Fahrrad und einige manchmal mit dem und manchmal mit dem anderen Verkehrsmittel fahren.

```
bus = {'James', 'Lisa', 'Thomas', 'Kathrin'}
rad = {'Alexandra', 'Thomas', 'James', 'Susan'}
```

Prüfen Sie, ob bestimmte Namen in den Mengen vorkommen.

```
'Lisa' in bus
```

```
True
```

```
'Susan' in rad
```

```
True
```

```
'Susan' in bus
```

```
False
```

Bilden Sie

- Schnittmenge

```
bus & rad
```

```
{'James', 'Thomas'}
```

- Differenz

```
bus - rad
```

```
{'Kathrin', 'Lisa'}
```

```
rad - bus
```

```
{'Alexandra', 'Susan'}
```

- Vereinigung

```
rad | bus
```

```
{'Alexandra', 'James', 'Kathrin', 'Lisa', 'Susan', 'Thomas'}
```

der beiden Mengen.

10.1.2 Übung 2

Prüfen Sie ob, eine zweite Menge mit Namen eine Untermenge Ihrer Menge aus Übung 1 ist.

```
neu = {'Lisa', 'Thomas'}
```

```
neu <= rad
```

```
False
```

```
neu <= bus
```

```
True
```

Alternativ:

```
neu.issubset(bus)
```

```
True
```

10.1.3 Übung 3

Definieren Sie eine Menge, die ihrerseits wiederum aus Mengen von Namen besteht. Definieren Sie eine zweite Menge, die eine Schnittmenge mit dieser Menge hat.

1. Versuch

```
mm = {bus, rad}
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-36-ab56e2f7f776> in <module>
----> 1 mm = {bus, rad}

TypeError: unhashable type: 'set'
```

Geht nicht, da die Elemente einer Menge hashable sein müssen. Mengen selbst sind veränderlich und dann nicht hashable. Es gibt aber eine unveränderliche Menge: `frozenset``:

```
mm = {frozenset(bus), frozenset(rad)}
mm
```

```
{frozenset({'James', 'Kathrin', 'Lisa', 'Thomas'}),
 frozenset({'Alexandra', 'James', 'Susan', 'Thomas'})}
```

10.1.4 Übung 4

Erweitern Sie Ihr Programm aus Übung 5 zu Dictionarys und verwenden Sie eine Menge, um zu prüfen, ob das gesuchte Element darin enthalten ist. Vergleichen Sie die Laufzeit mit der für die Liste und das Dictionary (aus Übung 5 zu Dictionarys).

Wir nehmen unsere Lösung aus dem Abschnitt Dictionarys und fügen `s = set(L)`. Weiterhin messen wir die Nachschau-Zeit in unserer Menge und geben das Zeit-Verhältnis von Menge und Dictionary aus:

```
n_name = 'Anzahl'
ratio_name = 'Ratio'
print(f'{n_name:>8s} {ratio_name:>10s}')
print('-' * 19)
for n in [10, 100, 1000, 10_000, 100_000, 1_000_000]:
    target = 'target'
    L = list(range(n))
    L[len(L) // 2] = target
    d = dict.fromkeys(L)
    s = set(L)
    time_set = %timeit -o -q target in s
    time_dict = %timeit -o -q target in d
    print(f'{n: 8d} {time_set.average / time_dict.average: 10.2f}')
```

```
  Anzahl      Ratio
-------------------
      10       0.93
     100       0.94
    1000       1.05
   10000       1.12
  100000       0.95
 1000000       0.97
```

Die Zeiten für Menge und Dictionary sind defacto gleich.

11 Funktionen

11.1 Übungen

11.1.1 Übung 1

Schreiben Sie eine Funktion, die die Zahlen von 0 bis 9 mit `print()` ausgibt.

```
def show_0_9():
    for x in range(10):
        print(x)

show_0_9()
```

```
0
1
2
3
4
5
6
7
8
9
```

11.1.2 Übung 2

Schreiben Sie eine Funktion, die das Doppelte einer Zahl zurückgibt.

```
def double(value):
    """Return `value` multiplied by 2."""
    return value * 2

double(10)
```

```
20
```

11.1.3 Übung 3

Schreiben Sie eine Funktion, die zwei Strings zusammenfügt (konkateniert) und übergeben Sie dieser Funktion Schlüsselwort-Argumente in unterschiedlichen Reihenfolgen.

```
def concat(str1, str2):
    """Concatenate two strings."""
    return str1 + str2
```

```
concat(str1='Hallo ', str2='Welt!')
```

```
'Hallo Welt!'
```

```
concat(str2='Welt!', str1='Hallo ')
```

```
'Hallo Welt!'
```

11.1.4 Übung 4

Schreiben Sie eine Funktion, die das arithmetische Mittel (Summe geteilt durch die Anzahl) von drei ihr als Argumente übergebenen Zahlen berechnet.

```
def mean(a, b, c):
    """Mean of three numbers."""
    return (a + b + c) / 3
```

```
mean(4, 8, 1)
```

```
4.333333333333333
```

11.1.5 Übung 5

Nutzen Sie einen voreingestellten Parameter für die Funktion aus Übung 4, so dass diese auch mit zwei Argumenten aufrufbar ist. Achtung: Die Summe der Argumente muss je nach Fall durch zwei oder drei geteilt werden. Ein sinnvoller Vorgabewert könnte `None` sein, da dieser sich eindeutig mit `var_name is None` von Zahlen unterscheiden lässt.

```
def mean(a, b, c=None):
    """Mean of two or three numbers."""
    if c is None:
        return (a + b) /2
    return (a + b + c) / 3
```

```
mean(4, 8, 1)
```

```
4.333333333333333
```

```
mean(4, 8)
```

```
6.0
```

Bonus: Lösung für eine beliebige Anzahl von Argumenten

```
def mean(*args):
    return sum(args) / len(args)
```

```
mean(4, 8, 1)
```

```
4.333333333333333
```

```
mean(4, 8, 1, 5, 3, 2, 1)
```

```
3.4285714285714284
```

11.1.6 Übung 6

Fortgeschritten: Schreiben Sie eine Funktion, die eine beliebige andere Funktion aufrufen und deren Laufzeit messen kann. Die Mess-Funktion soll die aufzurufende Funktion als erstes Argument und deren Parameter als weitere Argumente haben. (Hinweis: `*` und `**` erlauben eine unbekannte Anzahl von positionellen und Schlüsselwort-Argumenten zu verarbeiten.) Für die Zeitmessung können Sie `timeit.default_timer` nutzen.

```
import timeit

def measure_time(func, *args, **kwargs):
    """Measure the runtime of `func`.
    """
    start = timeit.default_timer()
    res = func(*args, **kwargs)
    duration = timeit.default_timer() - start
    print('run time:', duration)
    return res
```

```
measure_time(mean, 2, 3, 4)
```

```
run time: 2.2499999996483666e-06
```

```
3.0
```

```
import time

measure_time(time.sleep, 2)
```

```
run time: 2.001190791
```

12 Iteratoren und Generatoren

12.1 Iteratoren

12.2 Generatoren

12.3 Übungen

12.3.1 Übung 1

Definieren Sie einen Generator mit Hilfe eines Generatorausdrucks, der das Dreifache der Zahlen von 0 bis 20 ausgibt.

```
g = (x * 3 for x in range(21))
```

```
print(list(g))
```

```
[0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48, 51, 54, 57, 60]
```

12.3.2 Übung 2

Modifizieren Ihren Generator aus Übung 1 so, dass dieser nur gerade Zahlen ausgibt. Hinweis: Der Modulo-Operator `%` liefert den Rest einer ganzzahligen Division. Mit der Zahl Zwei als Divisor erhalten wir für alle geraden Zahlen eine Null und für alle ungeraden Zahlen eine Eins:

```
>>> 10 % 2
0
>>> 9 % 2
1
```

```
g = (x * 3 for x in range(21) if x % 2 == 0)
list(g)
```

```
[0, 6, 12, 18, 24, 30, 36, 42, 48, 54, 60]
```

12.3.3 Übung 3

Schreiben Sie eine Generatorfunktion, die beim ersten Aufruf von `next()` die Zahl `10` und bei allen anderen Aufrufen von `next()` die Zahlen `100`, `200` ... `900` ausgibt.

```
def gen_func():
    yield 10
    for x in range(100, 901, 100):
        yield x

list(gen_func())
```

```
[10, 100, 200, 300, 400, 500, 600, 700, 800, 900]
```

12.3.4 Übung 4

Vergleichen Sie den Speicherverbrauch zweier Python-Programme, die für die Repräsentation einer Reihe von Zahlen

1. eine Liste und
2. einen Generatorausdruck verwenden.

Die Liste bzw. der Generator soll nacheinander 10.000, 100.000 und 1.000.000 Zahlen repräsentieren.

Es gibt zwei Möglichkeiten den Speicherverbrauch zu messen.

(1) Nutzen Sie das mitgelieferte Script `measure_mem.py`. Importieren dieses in Ihrem Programm:

```
import measure_mem

#  Ihr Programm-Code

measure_mem.show_memory()
```

Sie müssen dafür `psutil` installieren. Wenn Sie einen Import-Fehler erhalten, der sich über das Fehlen `psutil` beschwert, installieren Sie das Modul mit `conda` oder `pip`.

Wenn Sie `conda` nutzen, geben Sie dies auf der Kommandozeile ein:

```
conda install psutil
```

andernfalls nutzen Sie `pip`:

```
pip install psutil
```

(2) Wenn Sie `psutils` nicht installieren können, nutzen Sie bitte das für ihr Betriebssystem verfügbare Programm zur Überwachung von Prozessen wie den Taskmanager unter Windows, die Aktivitätsanzeige unter Mac OSX oder `top` unter Linux, um den Speicherverbrauch für beide Programme zu messen.

Hinweise: Mit `input()` können Sie die Ausführung des Programms anhalten. Mit Hilfe der Funktion `getpid()` aus dem Modul `os` können Sie die eigene Prozess-ID bestimmen. Suchen Sie diese ID in dem Prozess-Beobachtungs-Werkzeug Ihres Betriebssystems. Wenn es keine Spalte dafür anzeigt, müssen Sie die Spalte PID in den Anzeigeoptionen des Programms auswählen.

Lösung

Dieses Programm erzeugt eine Liste der Größe `n`. Der Wert für `n` kommt als Kommandozeilen-Argument:

```
import sys

import measure_mem

n = int(float(sys.argv[1]))

L = [x + 10 for x in range(n)]

measure_mem.show_memory()
```

Die Version mit dem Generator-Ausdruck sieht fast genauso aus. Nur `[]` ist durch `()` ersetzt. Wir nennen das Objekt jetzt `g` statt `L`:

```
import sys

import measure_mem

n = int(float(sys.argv[1]))

g = (x + 10 for x in range(n))

measure_mem.show_memory()
```

Jetzt lassen wir unser Programm mit eine Listen- bzw. Generator-Größe von `10` laufen:

```
$ python mem_list.py 10
Memory usage: 9.41015625 MB
$ python mem_gen.py 10
Memory usage: 9.42578125 MB
```

Beide Programme benötigen die gleiche Menge an Speicher. Versuchen wir eine Größe von 10.000 Elementen:

```
$ python mem_list.py 10_000
Memory usage: 9.79296875 MB
$ python mem_gen.py 10_000
Memory usage: 9.4609375 MB
```

Jetzt benötigt die Listen-Version etwas mehr Speicher. Versuchen die Größe vom einer Million Elementen:

```
$ python mem_list.py 1e6
Memory usage: 55.1015625 MB
$ python mem_gen.py 1e6
Memory usage: 9.4765625 MB
```

Das ist schon ein großer Unterschied. Die Listen-Version benötigt fast da sechsfache des Speichers. Mit einer Größe von 10 Millionen wir der Unterschied noch deutlicher:

```
$ python mem_list.py 1e7
Memory usage: 348.8515625 MB
$ python mem_gen.py 1e7
Memory usage: 9.5 MB
```

Jetzt ist der Unterschied schon mehr als das 36-fache. Der Speicherverbrauch des Generators ist unabhängig von der Größe von `n`.

13 Klassen

13.1 Grundlagen

13.2 Übungen

13.2.1 Übung 1

Schreiben Sie eine Klasse `Person`, die die Attribute `name` und `standort` hat.

Lösung

```
class Person:
    """Einfache Person."""

    def __init__(self, name, standort):
        self.name = name
        self.standort = standort
```

13.2.2 Übung 2

Fügen Sie eine Methode `gehezu` hinzu, die einen neuen `standort` setzt.

Lösung

```
class Person:
    """Einfache Person."""

    def __init__(self, name, standort):
        self.name = name
        self.standort = standort

    def gehezu(self, neuer_standort):
        """Standort ändern"""
        self.standort = neuer_standort
```

13.2.3 Übung 3

Machen Sie mehrere Instanzen dieser Person und lassen Sie diese Personen zu verschiedenen Standorten gehen.

Lösung

```
person1 = Person('Erika', 'Büro')
person1.standort
```

```
'Büro'
```

```
person1.gehezu('Restaurant')
person1.standort
```

```
'Restaurant'
```

```
person2 = Person('Max', 'zu Hause')
person2.standort
```

```
'zu Hause'
```

```
person1.gehezu('Arbeit')
person1.standort
```

```
'Arbeit'
```

13.3 Vererbung

13.3.1 Übungen

Übung 1

Schreiben Sie eine Klasse `EingeschraenktePerson`, die von `Person` erbt.

Lösung

```
class Person:
    """Einfache Person."""

    def __init__(self, name, standort):
        self.name = name
        self.standort = standort

    def gehezu(self, neuer_standort):
        """Standort ändern"""
        self.standort = neuer_standort

class EingeschraenktePerson(Person):
    """Person mit anderm Typ"""
```

Übung 2

Überschreiben Sie in der Klasse `EingeschraenktePerson` die Methode `gehezu` und verbieten Sie der Person zu bestimmten Ort zu gehen. So können Sie z.B. auf dem Bildschirm `Der Zugang zu Ort xxx ist verboten.` ausgeben und behalten Sie den alten Ort bei, wenn der Zielort in der Liste Ihrer verbotenen Orte ist.

Lösung

```
class EingeschraenktePerson(Person):
    """Person, die betsimmet Standorte nicht besuchen darf"""

    verbotene_standdorte = ['Restaurant', 'Bar', 'Kneipe']

    def gehezu(self, neuer_standort):
        """Standort ändern"""
        if neuer_standort in self.verbotene_standdorte:
            print(f'Der Zugang zu Ort {neuer_standort} ist verboten.')
            print(f'Bleibe hier: {self.standort}')
```

```
eperson1 = EingeschraenktePerson('Erika', 'zu Hause')
```

```
eperson1.standort
```

```
'zu Hause'
```

```
eperson1.gehezu('Bar')
```

```
Der Zugang zu Ort Bar ist verboten.
Bleibe hier: zu Hause
```

```
eperson1.standort
```

```
'zu Hause'
```

```
eperson1.gehezu('Arbeit')
eperson1.standort
```

```
'zu Hause'
```

13.4 Operatorüberladung

13.4.1 Spezielle Methoden

13.4.2 Übungen

Übung 1

Überladen Sie den Operator >> (Hinweis: Nutzen Sie die spezielle Methode `__rshift__`) mit der gleichen Funktionalität wie `gehezu`.

Das Ergebnis sollte so aussehen:

Lösung

```
class Person:
    """Einfache Person."""

    def __init__(self, name, standort):
        self.name = name
        self.standort = standort

    def gehezu(self, neuer_standort):
        """Standort ändern"""
        self.standort = neuer_standort

    def __rshift__(self, neuer_standort):
        self.gehezu(neuer_standort)
```

```
person = Person('Fritz', 'zu Hause')
person.standort
```

```
'zu Hause'
```

```
person >> 'hinter den sieben Bergen'
person.standort
```

```
'hinter den sieben Bergen'
```

14 Ausnahmen und Fehlerbehandlung

14.1 Übungen

14.1.1 Übung 1

Lösen Sie einen `AttributeError` aus, indem Sie ein nicht definierte Attribute eines Objektes zugreifen. (Hinweis: `myobject.attribute`)

Lösung

```
sum.xyz
```

```
AttributeError: 'builtin_function_or_method' object has no attribute 'xyz'
```

14.1.2 Übung 2

Fangen Sie diesen Fehler ab und geben Sie eine Nachricht aus, dass das Attribut nicht definiert ist.

Lösung

```
try:
    sum.xyz
except AttributeError:
    print("Attribute 'xyz' ist nicht definiert")
```

```
Attribute 'xyz' ist nicht definiert
```

14.1.3 Übung 3

Geben Sie Sie auf dem Bildschirm `Fertig!` aus, unabhängig davon, ob der `AttributeError` auftritt oder nicht.

Lösung

```
try:
    sum.xyz
except AttributeError:
    print("Attribute 'xyz' ist nicht definiert")
finally:
    print('Fertig!')
```

```
Attribute 'xyz' ist nicht definiert
Fertig!
```

14.1.4 Übung 4

Legen Sie eine Liste mit fünf Elementen an. Greifen Sie auf das achte Element zu. Fangen Sie diesen Fehler ab und geben Sie eine entsprechende Nachricht auf dem Bildschirm aus.

Lösung

```
L = list(range(5))
L[7]
```

```
IndexError: list index out of range
```

```
index = 7
try:
    L[index]
except IndexError:
    print(f'Index {index} nicht in Liste der Länge {len(L)}')
```

```
Index 7 nicht in Liste der Länge 5
```

14.1.5 Übung 5

Schreiben Sie eine Funktion, die als Parameter ein Dictionary, einen Schlüssel und einen Vorgabewert (default) hat. Implementieren Sie mit Hilfe von `try` und `except` mit dieser Funktion das Verhalten der Methode `get` eines Dictionarys.

So soll zum Beispiel dieses Verhalten:

```
my_dict = {'a': 100, 'b': 200}
my_dict.get('a')
```

```
100
```

```
print(my_dict.get('x'))
```

```
None
```

```
my_dict.get('x', 999)
```

```
999
```

durch die Funktion `my_get` abgebildet werden:

```
my_get(my_dict, 'a')
```

```
100
```

```
print(my_get(my_dict, 'x'))
```

```
None
```

```
my_get(my_dict, 'x', 999)
```

```
999
```

Lösung

```
def my_get(dic, key, default=None):
    """Works like `dic.get()`.
    """
    try:
        return dic[key]
    except KeyError:
        return default
```

14.1.6 Übung 6

Definieren Sie eine eigene Ausnahme `PositiveOnly`, die Sie nutzen wollen, um negative Zahlen zu verbieten. Testen Sie, ob eine Zahl negativ ist, und werfen Sie Ihre Ausnahme `PositiveOnly`, wenn dies zutrifft. Gegeben Sie dabei auf dem Bildschirm eine entsprechende Nachricht inklusive des Wertes der negativen Zahl aus.

Lösung

```
class PositiveOnly(Exception):
    pass
```

```
value = -5
if value < 0:
    raise PositiveOnly(f'got negative value {value}')
```

```
PositiveOnly: got negative value -5
```

15 Ein- und Ausgabe

15.1 Interaktive Eingabe

15.2 Kommandozeilenargumente

15.3 Dateien schreiben

15.4 Die `with`-Anweisung

15.5 Dateien lesen

15.6 Methoden zum Lesen und Schreiben von Dateien

15.7 Datenstrukturen einfach speichern

15.8 Übungen

15.8.1 Übung 1

Schreiben Sie ein Programm, dass Sie zur Eingabe Ihres Namens auffordert und diesen am Bildschirm wieder ausgibt.

Lösung

```
name = input('Bitte geben Sie Ihren Namen ein: ')
print(f'Hallo: {name}')
```

Ausgabe:

```
Bitte geben Sie Ihren Namen ein:  Max Mustermann
Hallo: Max Mustermann
```

15.8.2 Übung 2

Modifizieren Sie das Programm so, dass die Abfrage so lange wiederholt wird, bis mit dem Wort `end` die Ausführung des Programms beendet wird.

Lösung

```
while True:
    name = input('Bitte geben Sie Ihren Namen ein: ')
    if name.strip().lower() == 'end':
        print('Tschüß')
        break
    print(f'Hallo: {name}')
```

15.8.3 Übung 3

Erzeugen Sie mit Python eine neue Textdatei und schreiben Sie die Zahlen von 1 bis 10, eine Zahl pro Zeile, hinein.

Lösung

```
with open('numbers.txt', 'w') as fobj:
    for number in range(1, 11):
        fobj.write(f'{number}\n')
```

Inhalt von `numbers.txt`:

```
1
2
3
4
5
6
7
8
9
10
```

15.8.4 Übung 4

Lesen Sie die gerade erzeugte Datei

(a) in einen String und

Lösung

```
with open('numbers.txt') as fobj:
    numbers = fobj.read()

print(numbers)
```

```
1
2
3
4
```

(continues on next page)

(continued from previous page)

```
5
6
7
8
9
10
```

(b) in eine Liste.

Lösung

```
with open('numbers.txt') as fobj:
    numbers = fobj.readlines()

print(numbers)
```

```
['1\n', '2\n', '3\n', '4\n', '5\n', '6\n', '7\n', '8\n', '9\n', '10\n']
```

15.8.5 Übung 5

Öffnen Sie die Datei so, dass sie nach Verlassen des Kontextes automatisch geschlossen wird.

Lösung

```
with open('numbers.txt') as fobj:
    print('eingerückt:', end=' ')
    print('geschlossen' if fobj.closed else 'geöffnet')
print('ausgerückt:', end=' ')
print('geschlossen' if fobj.closed else 'geöffnet')
```

```
eingerückt: geöffnet
ausgerückt: geschlossen
```

15.8.6 Übung 6

Verwenden Sie das Modul `pickle` um eine Liste `[1, 2, 3]` in eine Datei zu speichern und lesen Sie diese wieder ein.

Lösung

```
import pickle

L = [1, 2, 3]

with open('liste.pcl', 'wb') as fobj:
    pickle.dump(L, fobj)

with open('liste.pcl', 'rb') as fobj:
    L2 = pickle.load(fobj)

L2
```

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

15.8.7 Übung 7

Fortgeschrittene Aufgabe: Nutzen Sie das Modul `argparse` aus der Standard-Bibliothek, um Kommandozeilenargumente um optionale Namen für eine Eingabedatei und eine Ausgabedatei zu verarbeiten. Das Programm sollte diese Ausgabe erzeugen:

```
python argparse_example.py --input myinput.txt --output myoutput.txt
Using input file: myinput.txt
Using output file: myoutput.txt
```

Lösung

```
"""
Parse commandline options with argparse.
"""

import argparse

parser = argparse.ArgumentParser(description='process data')
parser.add_argument(
    '-i',
    '--input',
    help='input file name',
    default='in.txt')
parser.add_argument(
    '-o',
    '--output',
    help='output file name',
    default='out.txt')

args = parser.parse_args()

print(f'Using input file: {args.input}')
print(f'Using output file: {args.output}')
```

Aufruf ohne Angabe von Argumenten:

```
python argparse_example.py
```

```
Using input file: in.txt
Using output file: out.txt
```

Ausgabe mit Kurzangabe für die Eingabe-Datei:

```
python argparse_example.py -i myinput.txt
```

```
Using input file: myinput.txt
Using output file: out.txt
```

Ausgabe mit Angabe beider Datei-Namen:

```
argparse_example.py --input myinput.txt --output myoutput.txt
```

```
!python argparse_example.py --input myinput.txt --output myoutput.txt
```

```
Using input file: myinput.txt
Using output file: myoutput.txt
```

16 Die eigene Bibliothek - Beispiel: Rechnen mit Listen

16.1 Listen-Mathematik

16.2 Verzeichnis-Struktur

16.3 Import

16.4 Übungen

16.4.1 Übung 1

Erstellen Sie im Datei-Explorer oder der Kommandozeile ein Verzeichnis und Dateien, die ein Python-Paket mit einem Modul mit einer Funktion enthält. Die Funktion soll `Hello world!` ausgeben. Aus dem Verzeichnis in dem sich Ihr Paket befindet, soll dies funktionieren:

```
from mypackage.mymodule import hello

hello()
```

```
Hello world!
```

Lösung

Das ist die Datei-Struktur:

```
mypackage/
├── __init__.py
└── mymodule.py
```

Der Inhalt von `mypackage/mymodule.py` sieht so aus:

16.4.2 Übung 2

Erweitern Sie ihr Paket um ein Unter-Paket. Nach der Erweiterung soll dies möglich sein:

```
from mypackage.utils.helpers import show_help
```

```
show_help()
```

```
I would help you, if I could.
```

Lösung

Das ist die Datei-Struktur:

```
mypackage2/
├── __init__.py
├── mymodule.py
└── utils
    ├── __init__.py
    └── helpers.py
```

Der Inhalt von `mypackage/utils/helper.py` sieht so aus:

```
"""Helpers.
"""


def show_help():
    """A helper function.
    """
    print('I would help you, if I could.')
```

17 Module und Pakete

17.1 Definition

17.2 Pakete finden

17.3 Übung

17.3.1 Übung 1

Kontrollieren Sie, ob das Paket `listmath` bereits im Suchpfad von Python ist. (Hinweis: Nutzen Sie `sys.path`.)

Lösung

Nach dem auflisten von `sys.path`

```
import sys

sys.path

['pfad/1/,
 'pfad/2/,
 ...
 'pfad/zu/meinen/modul]
```

Manuell nachschauen, ob der Pfad in der das Verzeichnis `listmath` mit dem Quelltext liegt, dabei ist.

17.3.2 Übung 2

Importieren Sie ein Modul aus dem Paket `listmath` und zeigen Sie dessen Docstring an.

Lösung

```
import listmath.math.arithmetics as arith

print(arith.__doc__)
```

```
List arithmetics.
```

17.3.3 Übung 3

Nutzen Sie die Kommandos `dir` und `help`, um mehr über dieses Modul zu erfahren.

```
dir(arith)
```

```
['__all__',
 '__builtins__',
 '__cached__',
 '__doc__',
 '__file__',
 '__loader__',
 '__name__',
 '__package__',
 '__spec__',
 'add',
 'div',
 'mul',
 'sub']
```

```
print(help(arith))
```

Ausgabe:

```
Help on module listmath.math.arithmetics in listmath.math:

NAME
    listmath.math.arithmetics - List arithmetics.

FUNCTIONS
    add(seq1, seq2)
        Add two sequences element-wise.

        Returns a list.

    div(seq1, seq2)
        Divide two sequences element-wise.

        Returns a list.

    mul(seq1, seq2)
        Multiply two sequences element-wise.

        Returns a list.

    sub(seq1, seq2)
        Subtract two sequences element-wise.

        Returns a list.

DATA
    __all__ = ['add', 'sub', 'mul', 'div']

FILE
    /path/to/listmath/math/arithmetics.py
```

18 Objekte im Detail

19 Namen für Objekte

19.1 Übungen

19.1.1 Übung 1

Wenden Sie PyLint auf die Module von `listmath` an.

Lösung

Auf der Kommandozeile im Verzeichnis in dem `listmath` liegt eingeben:

```
pylint listmath
```

19.1.2 Übung 2

Bauen Sie temporär absichtlich Fehler in eines der Module ein und prüfen Sie wiederholt mit PyLint.

Lösung

Mögliche Fehler:

- Variablen anlegen aber nie nutzen
- Code auf der Zeilen nach `return` mit der gleichen Einrückungsebene schreiben
- Funktion ohne Docstring

19.1.3 Übung 3

Nutzen Sie PyLint für eigene Programme, zum Beispiel für die Lösungen anderer Übungen.

20 Namensräume und Gültigkeitsbereiche

20.1 Namensräume sauber halten

20.2 Die LGB-Regel

20.3 Die LEGB-Regel

20.4 Global und nonlocal

20.5 Übungen

20.5.1 Übung 1

Bestimmen Sie wie viele Attribute das Modul `os` hat.

Lösung

```
import os

len(dir(os))
```

```
294
```

20.5.2 Übung 2

Aus welchen Namensräumen kommen `x` und `abs` wenn nur dieser Code einer Python-Quelltext-Datei steht?

```
x = 10
abs
```

Lösung

- `x` - global
- `abs` - builtin

21 Strings

21.1 String-Methoden

21.2 Formatierung

21.3 Mit `f`-Strings

21.4 Wichtige Formatierungstypen

21.5 Übungen

21.5.1 Übung 1

Schreiben sie den Text `Das soll in der Mitte stehen` *ungefähr* in die Mitte einer Zeile des Terminals oder Notebooks und füllen Sie den Rest der Zeile mit Ausrufezeichen aus (`!!!!!!`).

Lösung

```
text = 'Das soll in der Mitte stehen'
text.center(80, '!')
```

```
'!!!!!!!!!!!!!!!!!!!!!!!!!Das soll in der Mitte stehen!!!!!!!!!!!!!!!!!!!!!!!!!'
```

Im Terminal:

```
import os

text = 'Das soll in der Mitte stehen'
text.center(os.get_terminal_size().columns, '!')
```

```
'!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!Das soll in der␣
 ↪Mitte stehen!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!'
```

21.5.2 Übung 2

Überprüfen Sie, ob verschiedene Strings nur aus Zahlen oder nur aus Buchstaben bestehen.

Lösung

```
text1 = 'Text'
text2 = '123'
```

```
text1.isalpha()
```

```
True
```

```
text2.isalpha()
```

```
False
```

```
text1.isdecimal()
```

```
False
```

```
text2.isdecimal()
```

```
True
```

21.5.3 Übung 3

Schreiben Sie einen Satz und wandeln Sie ihn in eine Liste um, in der jedes Wort ein Element wird. Setzen Sie aus der Liste einen String zusammen, in dem die Worte durch Unterstriche verbunden sind.

Lösung

```
s = 'Das ist ein vollständiger Satz.'
'_'.join(s.split())
```

```
'Das_ist_ein_vollständiger_Satz.'
```

21.5.4 Übung 4

Schreiben sie ein kleines Template-System für Serienbriefe. Fügen sie Empfängernamen und die Summe, die er ihnen schuldet, automatisch ein. Die Summe sollte dreimal im Dokument erscheinen. Nutzen Sie `template.format()` mit Dictionary-Methode für die Ersetzung. Hinweise: Für Templates eigne sich ein mehrzeiliger String mit drei Anführungszeichen besonders.

Lösung

Vorlage erstellen (könnte aus Datei gelesen werden):

```
template = """
Sehr geehrte{endung} {anrede} {name},

Sie schulden mir noch {betrag}.

Bitte überweisen Sie den Betrag von {betrag:5.2f} bis zum {datum:%d.%m.%Y}.

Viele Grüße
Der Eintreiber
"""
```

Daten erzeugen (könnte aus einer Datenbank kommen):

```
import datetime

today = datetime.date.today()


data = [
    {
        'name': 'Erika Mustermann',
        'endung': '',
        'anrede': 'Frau',
        'betrag': 100,
        'datum': today + datetime.timedelta(14),
    },

    {
        'name': 'Max Mustermann',
        'endung': 'r',
        'anrede': 'Herr',
        'betrag': 150,
        'datum': today + datetime.timedelta(30),

    },
]
```

Briefe erzeugen:

```
for entry in data:
    print('#' * 80)
    print(template.format(**entry))
```

```
################################################################################

Sehr gehrte Frau Erika Mustermann,

Sie schulden mir noch 100.

Bitte überweisen Sie den Betrag von 100.00 bis zum 20.02.2022.

Viele Grüße
Der Eintreiber

################################################################################

Sehr gehrter Herr Max Mustermann,
```

(continues on next page)

(continued from previous page)

```
Sie schulden mir noch 150.

Bitte überweisen Sie den Betrag von 150.00 bis zum 08.03.2022.

Viele Grüße
Der Eintreiber
```

22 Systemfunktionen

22.1 Modul - `sys`

22.2 Modul - `os`

22.3 Modul - `os.path`

22.4 Modul - `shutil`

22.5 Mehr Informationen

22.6 Übungen

22.6.1 Übung 1

Bestimmen sie das aktuelle Arbeitsverzeichnis aus Python heraus.

Lösung

```
import os

os.getcwd()
```

22.6.2 Übung 2

Geben Sie die genutzte Python-Version aus.

Lösung

```
import sys

sys.version
```

22.6.3 Übung 3

Wechseln Sie in ein anderes Verzeichnis, das Dateien und Unterverzeichnisse enthält.

Lösung

```
import os

os.chdir('sub')
```

Listen die den Verzeichnisinhalt mit Hilfe des Moduls `os` auf.

Lösung

```
print(os.listdir(os.getcwd()))
```

```
['file1.txt']
```

22.6.4 Übung 4

Bauen Sie einen Pfad mit einer vom Betriebssystem unabhängigen Methode zusammen.

Lösung

```
os.path.join('path', 'to', 'my', 'file.txt')
```

```
'path/to/my/file.txt'
```

Lösung

Annahme: Dieses Verzeichnis und diese Dateien existieren:

```
sub
├── file1.txt
└── file2.txt
```

```
import os

os.path.isfile('sub/file1.txt')
```

```
True
```

```
os.path.isfile('sub')
```

```
False
```

```
os.path.isdir('sub/file1.txt')
```

```
False
```

```
os.path.isdir('sub')
```

```
True
```

22.6.5 Übung 6

Kopieren Sie eine Datei von einem Verzeichnis in an anderes.

Lösung

```
import shutil

shutil.copy('sub/file1.txt', 'sub2/file1.xt')
```

```
'sub2/file1.xt'
```

Lösung

Annahme: Dieser Dateibaum existiert:

```
sub
├── file1.txt
├── file2.txt
└── subsub
    ├── file_a.txt
    └── file_b.txt
```

```
# access24.py
"""
What files have been accessed during the last 24 hours?
"""

import os
import time


def get_accessed(start_dir, since=24):
    """Return the full names of all files changed within the last
    24 hours or the time in hours given with `since`.
    """
    if not os.path.exists(start_dir):
        raise ValueError('no such directory:', start_dir)
    if since <= 0:
        raise ValueError('Value for `since` must be positive.')
    time_stamp = time.time() - (since * 3600)
    found = []
    for root, _dir_names, file_names in os.walk(start_dir):
        for file_name in file_names:
            # some files are not real files
            full_name = os.path.join(root, file_name)
            try:
                atime = os.path.getatime(full_name)
            except OSError:
                print('not found', file_name)
            if atime >= time_stamp:
                found.append(full_name)
    return found
```

(continues on next page)

(continued from previous page)

```
if __name__ == '__main__':

    def test():
        """Try on `tmp`.
        """
        found = (get_accessed(os.path.join(os.path.expanduser('~'), 'tmp')))
        for file_name in found:
            print(file_name)
        print('found:', len(found), 'files', \
        'that were changed within the last 24 hours.')

test()
```