ORGANISATORISCHES

- Du: Sebastian
- Whatsapp: 0172 690 84 57
- 17.03. + 24.03., 31.03., 14.04., 21.04., 28.04.
- 6 Stunden **Workshop** am12.05. (19.05. fällt dafür aus)
- 6 Stunden **Hackathon** am 26.05. (09.06. fällt dafür aus)
- 6 Stunden Vertiefung und Klausurvorbereitung am 16.06. (23.06. fällt dafür aus)

Workshop und Hackathon finden in Düsseldorf Benrath statt: Benrather Schloßallee 99

Teil 1

Funktionen & Variablen

Fortsetzung II

Eine Minute Übung

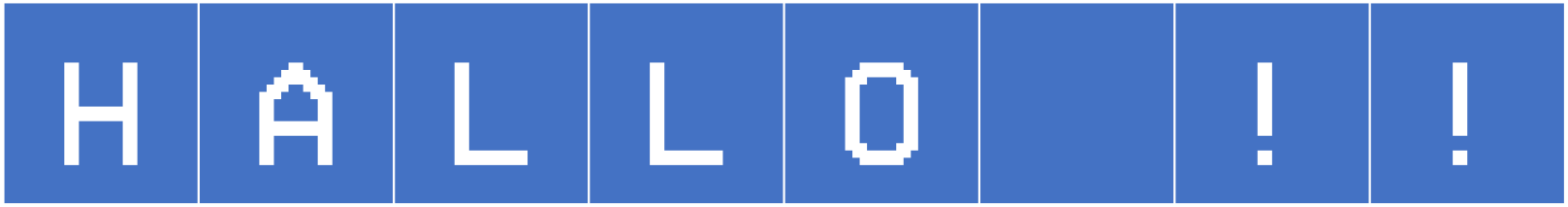
Ergänze die oberste und unterste Reihe mit der Zahlenfolge.

Position (Index)								
Sequenz: String „Hallo !!“	H	A	L	L	O		!	!
Länge								

Noch eine Minute Übung

Was ist jeweils die Ausgabe? (???)

Sequenz:
String
„Hallo !!“




```
>>> a = "Hallo !!"
>>> len(a)
>>> ???
>>> a[1]
>>> '???'
>>> a[:3]
>>> '???’
>>> a[1:4]
>>> '???’
>>> a[::2]
>>> '???'
```

Negatives Indexing

Positiver Index	0	1	2	3	4	5	6	7
Sequenz: String „Hallo !!“	H	A	L	L	O		!	!
Negativer Index	-8	-7	-6	-5	-4	-3	-2	-1

```
>>> a[-5]
L
```

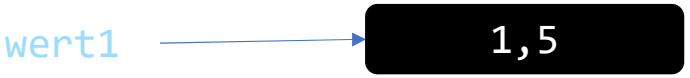
Lesbarkeit vs. Effizienz des Codes

Verketten & Einsetzen

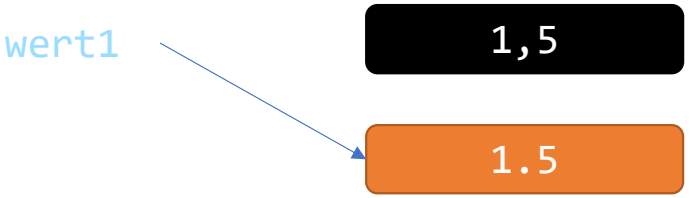
Wie lesbar ist der Code noch?

```
wert1 = input("Kommazahl 1? ")
# Entferne Leerzeichen am Anfang und Ende
wert1 = wert1.strip()
# Ersetze Kommas durch Punkte
wert1 = wert1.replace(",", ".")
# Wandele den String in eine Kommazahl um
wert1 = float(wert1)

# Oder in einer Zeile
wert2 = float(input("Kommazahl 2? ").strip().replace(",", "."))

if wert1 == wert2:
    print("Die Zahlen sind gleich.")
else:
    print("Die Zahlen sind ungleich. Es wurden unterschiedliche Eingaben gemacht.")

ergebnis = wert1 * 2
ergebnis = round(ergebnis, 2)
print(ergebnis)

# Oder in einer Zeile
print(round(wert2 * 2, 2))

# Oder alles in einer Zeile
print(round(float(input("Kommazahl 3? ").strip().replace(",", ".")) * 2, 2))
```

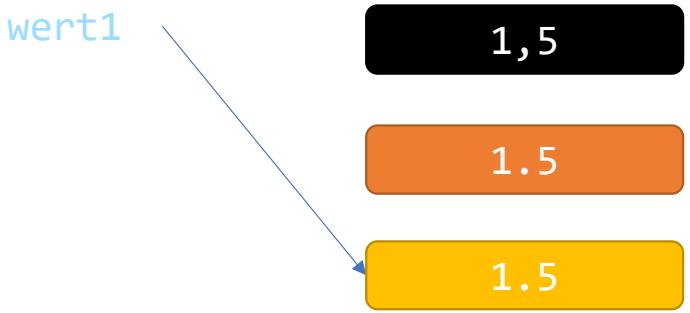
```
wert1 = "1,5"
```



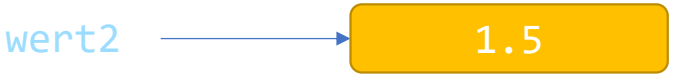

```
wert1 = wert1.replace(",", ".")
```




```
wert1 = float(wert1)
```



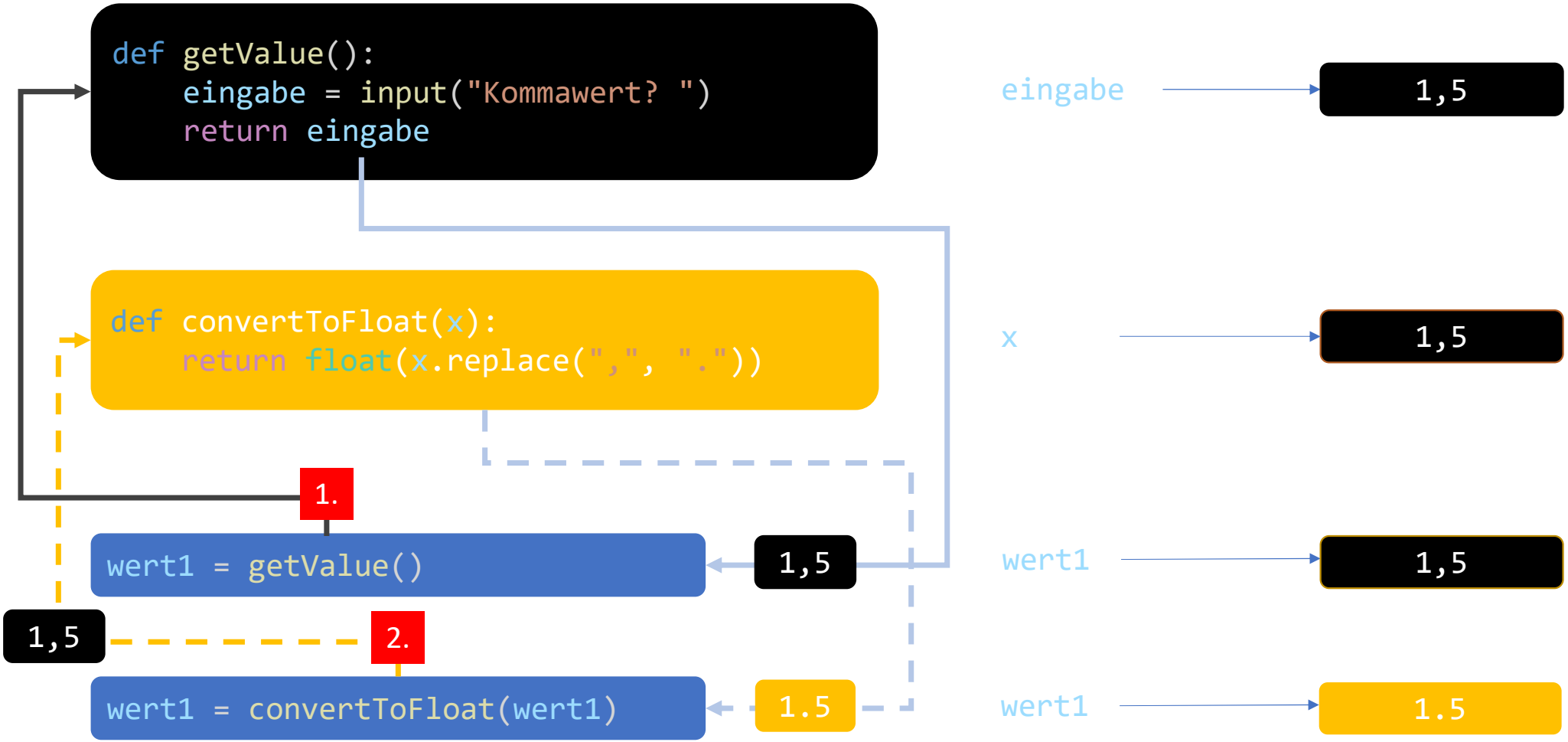
```
wert2 = float("1,5".replace(",", "."))
```

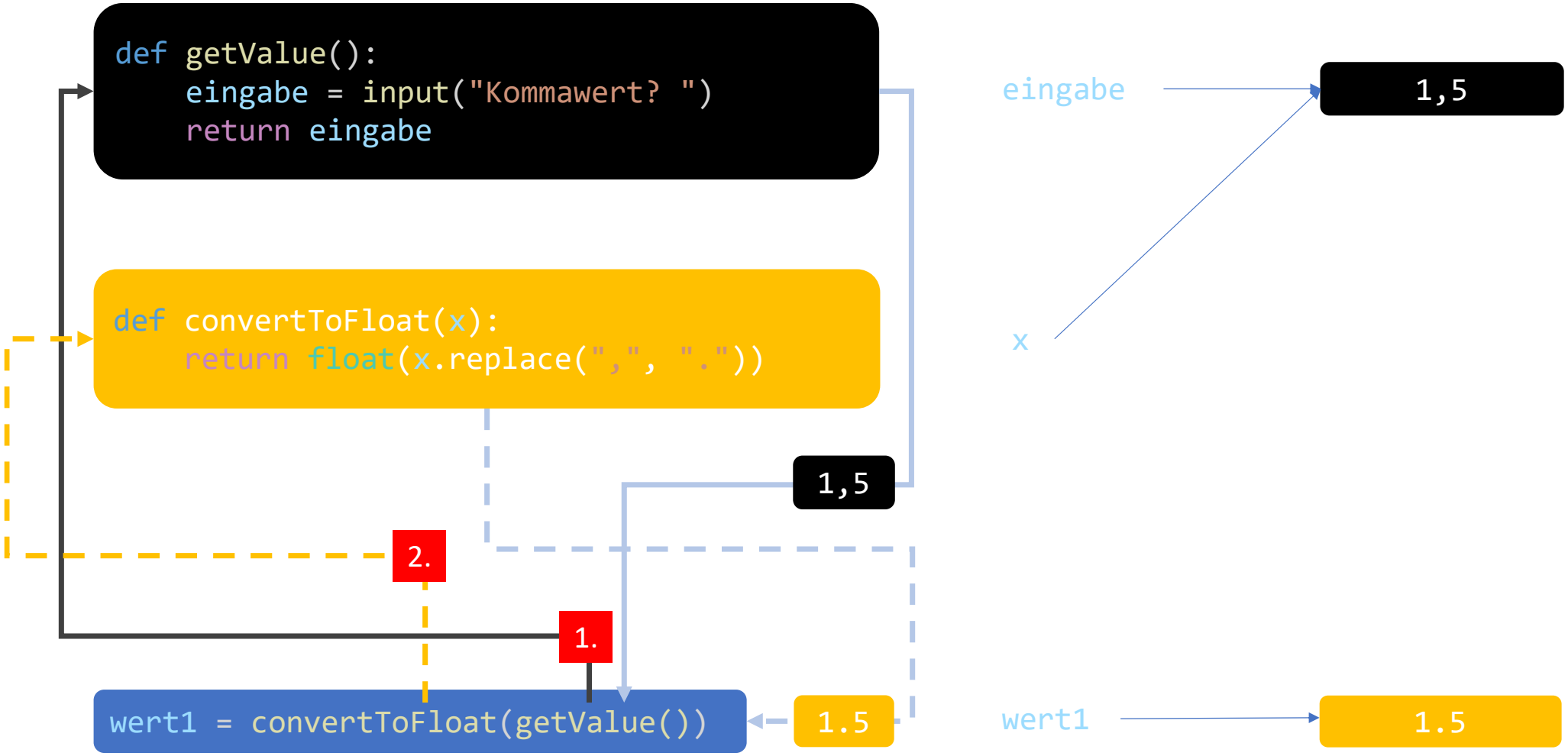


```python
wert1 = "1,5"
print(id(wert1))
wert1 = wert1.replace(",", ".")
print(id(wert1))
wert1 = float(wert1)
print(id(wert1))
```

```
1495626169968
1495627869936
1495625285008
```

Hinweis:
Die IDs sind zu jeder Laufzeit verschieden. Probiert ihr es aus, werden es also andere sein.


def getValue():
eingabe = input("Kommawert? ")
return eingabe
def convertToFloat(x):
return float(x.replace(",", "."))
1.
wert1 = getValue()
1,5
1,5
2.
wert1 = convertToFloat(wert1)
1.5
eingabe
1,5
x
1,5
wert1
1,5
wert1
1.5


def getValue():
eingabe = input("Kommawert? ")
return eingabe
def convertToFloat(x):
return float(x.replace(",", "."))
1,5
2.
1.
wert1 = convertToFloat(getValue())
1.5
eingabe
1,5
x
wert1
1.5

Wie lesbar ist der Code noch?

```
def getValue():
    eingabe = input("Kommawert? ")
    print(id(eingabe))
    return eingabe


def convertToFloat(x):
    print(id(x))
    return float(x.replace(",", "."))


# Variante 1
print("Variante 1:")
wert1 = getValue()
print(id(wert1))
wert1 = convertToFloat(wert1)
print(id(wert1))
print(wert1)

# Variante 2
print("Variante 2:")
print(convertToFloat(getValue()))
```

```
Variante 1:
Kommawert? 1,5
2187424557232
2187424557232
2187424557232
2187421859216
1.5
Variante 2:
Kommawert? 1,5
2187424555760
2187424555760
1.5
```

Immutable (unveränderliche) Sequenzen

Fortsetzung: Tuples

Tuples

- Immutable Sequenz von Unicode
- „,“ als Seperator notwendig per definitionem
- Zugriff auf Positionen per [ ]

```
Python 3.11.0 (main, Oct 24 2022, 18:26:48) [MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> t = () # leeres Tuple
>>> type(t)
<class 'tuple'>
>>> single_value_tuple = (42, ) # das Komma wird hier benötigt
>>> type(single_value_tuple)
<class 'tuple'>
>>> three_value_tuple = (1, 2, 3) # Klammern sind in dem Fall optional
>>> type(three_value_tuple)
<class 'tuple'>
>>> a, b = 0, 1 # Tuple mit Mehrfachzuweisung
>>> a, b = b, a # “swap” von Werten in einer Zeile und ohne temporäre Variable!
>>> a, b
(1, 0)
>>> c = (1, 2, 3)
>>> c[0] # Zugriff auf Position 0, das erste Element
1
>>>
```

Mutable (veränderbare) Sequenzen*

Lists, Dictionaries

* Übersprungen werden Sets, Bytearrays und Spezialtypen

Lists

- [ ] oder list()
- Komma-separiert
- Fortgeschritten: *List Comprehensions*, Beispiel: [x+5 for x in [2,3,4]]
- list((1, 2, 3, 4)) # Liste aus Tuple
- list(„Hallo“) # Liste aus String
- Methoden: append(), count(), extend(), index(), insert(), pop(), remove(), reverse(), sort(), clear()
- Gemischte Inhalte, Beispiel: [1,2,‘h‘]
- Operatoren: min(), max(), sum(), prod(), len()
- *Operator Overloading*, Beispiel: Liste * 2

```
Python 3.11.0 (main, Oct 24 2022, 18:26:48) [MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> [] # erstellt eine leere Liste
[]
>>> list() # erstellt auch eine leere Liste
[]
>>> [1 , 2, 3] # Wie bei einem Tuple werden die Werte mit Komma-separiert
[1, 2, 3]
>>> list('hello') # Liste aus einem String
['h', 'e', 'l', 'l', 'o']
>>> list((1, 2, 3)) # Liste aus einem Tuple
[1, 2, 3]
>>> a = [1, 2, 3]
>>> a.append(9) # die Liste am Ende ergänzen
>>> a
[1, 2, 3, 9]
>>> a.extend([9, 1, 3]) # die Liste um eine Liste ergänzen
>>> a
[1, 2, 3, 9, 9, 1, 3]
>>> a.count(9) # zählen wie oft ein Element vorkommt
2
>>> a.index(2) # herausfinden an welcher Position die 2 steht (0-Index!)
1
>>> a.insert(2, 7) # ein Element (hier die 7) an einer Position (hier die 2) einfügen
>>> a
[1, 2, 7, 3, 9, 9, 1, 3]
>>>
```

```
>>> a
[1, 2, 7, 3, 9, 9, 1, 3]
>>> a.pop() # zeige und entferne das letzte Element
3
>>> a
[1, 2, 7, 3, 9, 9, 1]
>>> a.pop(1) # zeige und entferne das Element auf Position 1
2
>>> a.remove(9) # entferne ein Element
>>> a
[1, 7, 3, 9, 1] # es wurde nur eine 9 entfernt!
>>> a.reverse() # ändere die Reihenfolge
>>> a
[1, 9, 3, 7, 1]
>>> a.sort() # sortiere die Liste
>>> a
[1, 1, 3, 7, 9]
>>> a.clear() # entferne alle Elemente aus der Liste
>>> a
[]
>>>
```

```
>>> b = [1, 2, 3]
>>> b.append("a") # Liste können verschiedene heterogene Elemente enthalten
>>> b
[1, 2, 3, 'a']
>>> c = [102, 344, 57]
>>> min(c) # kleinster Wert
57
>>> max(c) # größter Wert
344
>>> sum(c) # Summe der Werte
503
>>> from math import prod # aus der Library math importieren wir die Funktion prod
>>> prod(c) # Produkt der Listenelemente
2000016
>>> len(c) # Länge der Liste
3
>>> b * 2 # Beispiel für Operator Overloading: mehrfach ausgeben
[1, 2, 3, 'a', 1, 2, 3, 'a']
>>> b + c # Beispiel für Operator Overloading: anhängen
[1, 2, 3, 'a', 102, 344, 57]
>>>
```

Eine Minute Übung

Was kommt hier raus?

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> b + a * 2
```

Dictionary

- { } oder dict()
- key:value Paare komma-separiert
- Operatoren: len(), del, |, |=
- Methode: clear(), keys(), values(), items(), popitem(), pop(), update(), get()
- Zugriff auf Value [ ]
- Operator „in“

```
Python 3.11.0 (main, Oct 24 2022, 18:26:48) [MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> d = {} # Erstellen eines Dictionaries
>>> type(d)
<class 'dict'>
>>> d["a"] = 1 # Erstellen eines Eintrages: key “a”, value 1
>>> d
{'a': 1}
>>> "a" in d # Prüfen, ob ein Schlüssel (key) vorhanden ist
True
>>> d["b"] = 2
>>> d
{'a': 1, 'b': 2}
>>> len(d) # Anzahl der Einträge
2
>>> d["a"] # Zugriff auf einen Eintrag
1
>>> "c" in d
False
>>> "b" in d
True
>>> 2 in d # Sucht nicht in den Werten (Values)
False
```

```
>>> d.keys() # Gibt alle keys aus
dict_keys(['a', 'b'])
>>> d.values() # Gibt alle values aus
dict_values([1, 2])
>>> 2 in d.values() # Prüfen, ob ein Value vorhanden ist
True
>>> d.items() # Gibt alle Einträge
dict_items([('a', 1), ('b', 2)])
>>> ("a", 1) in d.items()
True
>>> d["c"] = 3
>>> d["d"] = 4
>>> d
{'a': 1, 'b': 2, 'c': 3, 'd': 4}
>>> d.popitem() # Entfernt einen zufälligen Eintrag
('d', 4)
>>> d
>>> d.pop("a") # Entfernt den Eintrag mit dem Schlüssel “a”
1
>>> d.update(b=9) # Aktualisiert einen Eintrag (und fügt ihn hinzu, wenn er fehlt)
```

```
>>> d.update(f=5)
>>> d
{'b': 9, 'c': 3, 'f': 5}
>>> d["z"] # Greift man auf einen nicht vorhandenen Schlüssel zu, dann gibt es einen Fehler
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'z'
>>> d.get("a") # Nicht aber, wenn man get() benutzt
>>> d.get("c")
3
>>> d.get("a", 99) # Default-Wert, wenn nicht vorhanden
99
>>> z = {"a":77,"b":"überschrieben","auch neu":999}
>>> d
{'b': 9, 'c': 3, 'f': 5}
>>> z
{'a': 77, 'b': 'überschrieben', 'auch neu': 999}
>>> d | z # Ausgabe dict d “Merge und Overwrite” mit dict z
{'b': 'überschrieben', 'c': 3, 'f': 5, 'a': 77, 'auch neu': 999}
>>> d
{'b': 9, 'c': 3, 'f': 5}
>>> z
{'a': 77, 'b': 'überschrieben', 'auch neu': 999}
>>> d |= z # dict d “Merge und Overwrite” mit dict z
>>> d
{'b': 'überschrieben', 'c': 3, 'f': 5, 'a': 77, 'auch neu': 999}
>>>
```

Eine Minute Übung

Um welchen Datentyp handelt es sich jeweils?

```
a = (1, 2, 3)
b = [1, 2, 3]
c = "1, 2, 3"
d = 1.5
e = {}
f = 1
```

```
Python 3.11.0 (main, Oct 24 2022, 18:26:48) [MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> text = "Hallo" # String oder str
>>> ganze_zahlen = 1 # Integer oder int
>>> fliesskomma = 1.5 # Float oder float
>>> tuple = () # Tuple oder tuple
>>> liste = [] # List oder list
>>> lexikon = {'A': 'Alpha', 'B': 'Beta'} # Dictionary oder dict
>>> type(text)
<class 'str'>
>>> type(ganze_zahlen)
<class 'int'>
>>> type(fliesskomma)
<class 'float'>
>>> type(tuple)
<class 'tuple'>
>>> type(liste)
<class 'list'>
>>> type(lexikon)
<class 'dict'>
>>>
```

Teil 2

Konditionen & Iterationen

Fortsetzung: Flow of Control

Beispiele (if)

Verkettete Vergleiche, Optimierungen

```python
klausurpunkte = int(input("Wie viele Punkte wurden erreicht? "))

if klausurpunkte >= 90 and klausurpunkte <= 100:
    print("1")
elif klausurpunkte >= 80 and klausurpunkte < 90:
    print("2")
elif klausurpunkte >= 70 and klausurpunkte < 80:
    print("3")
elif klausurpunkte >= 60 and klausurpunkte < 70:
    print("4")
else:
    print("mehr üben")
```

```python
klausurpunkte = int(input("Wie viele Punkte wurden erreicht? "))

if 100 >= klausurpunkte >= 90:
    print("1")
elif 90 > klausurpunkte >= 80:
    print("2")
elif 80 > klausurpunkte >= 70:
    print("3")
elif 70 > klausurpunkte >= 60:
    print("4")
else:
    print("mehr üben")
```

```python
klausurpunkte = int(input("Wie viele Punkte wurden erreicht? "))

if klausurpunkte >= 90:
    print("1")
elif klausurpunkte >= 80:
    print("2")
elif klausurpunkte >= 70:
    print("3")
elif klausurpunkte >= 60:
    print("4")
else:
    print("mehr üben")
```

Beispiele (if)

strings vergleichen

```python
# Variante 1:
user_input = input("Do you agree? (Y/N) ")

if user_input == "Y" or user_input == "y":
    print("You agreed.")
else:
    print("You did not agree.")
```

```python
# Variante 2:
def is_agreed(answer):
    if answer.startswith("y") or answer.startswith("Y"):
        return True
    else:
        return False


user_input = input("Do you agree? (Y/N) ")
if is_agreed(user_input):
    print("You agreed.")
else:
    print("You did not agree.")
```

```python
# Variante 3:
def is_agreed(answer):
    return answer.startswith("y") or answer.startswith("Y")


user_input = input("Do you agree? (Y/N) ")
if is_agreed(user_input):
    print("You agreed.")
else:
    print("You did not agree.")


# Variante 4:
user_input = input("Do you agree? (Y/N) ")
if user_input.startswith("y") or user_input.startswith("Y"):
    print("You agreed.")
else:
    print("You did not agree.")
```

Ternärer Operator

```
def main():
    x = int(input("Nenne eine ganze Zahl: "))
    if is_even(x):
        print("Gerade")
    else:
        print("Ungerade")

def is_even(n):
    return True if n % 2 == 0 else False

main()
```

match

```
name = input("What is your name? ")

match name:
    case "Alice":
        print("Hi, Alice.")
    case "Bob":
        print("Hi, Bob.")
    case "Sebastian" | "Seb":
        print("Hi, Sebastian.")
    case _:
        print(f"Hi, {name}.")
```

Loops (Schleifen)

for & while

for

- Iteriert über eine Sequenz
- continue (geht sofort zur nächsten Iteration)
- break (unterbricht die Schleife)

```python
# Beispiel 1
a = [1, 2, 3, 4]

for _ in a:
    print(_)


# Beispiel 2
# Probiere auch mal range(10, 20) und range(10, 20, 2); erkennst du das System wieder?
for n in range(10):
    print(n)


# Beispiel 3
text = "Shorten my text for Twitter, please!"

for buchstabe in text:
    if buchstabe.lower():
        if buchstabe in "aeiou":
            continue
    print(buchstabe, end="")
```

```python
# Beispiel 4
students = {"Tom":80, "Maria":95, "Lisa":90, "Karla":75, "Paul":60, "Peter":85}

for student in students:
    if students[student] >= 90:
        print(student, "hat eine 1")
    elif students[student] >= 80:
        print(student, "hat eine 2")
    elif students[student] >= 70:
        print(student, "hat eine 3")
    elif students[student] >= 60:
        print(student, "hat eine 4")
    else:
        print(student, "hat eine 5")
```

```python
# Beispiel 5
students = [
    {"name": "Peter", "points": 90, "yob": 2001},
    {"name": "John", "points": 92, "yob": 2004},
    {"name": "Anna", "points": 94, "yob": 2003},
    {"name": "Thomas", "points": 75, "yob": None},
    {"name": "Bob", "points": 81, "yob": 2002},
    {"name": "Donald", "points": 65, "yob": 2001},
    {"name": "Maria", "points": 96, "yob": 2003},
]

for student in students:
    if student["yob"] is None:
        continue
    print(student["name"], student["points"], student["yob"], sep=", ")
```

while

- Iteriert bis eine bestimmte Kondition erreicht wird
- continue (geht sofort zur nächsten Iteration)
- break (unterbricht die Schleife)
- Infinite loops

```python
# Beispiel 1
while True:
    user_input = input("Bitte ein positive, ganze Zahl eingeben: ")
    if user_input.isdigit():
        break


# Beispiel 2
a = 0
while a < 3:
    user_input = input("Bitte dreimal was mit mindestens drei Zeichen eingeben: ")
    if len(user_input) > 3:
        a += 1 # kann man statt a = a + 1 schreiben
        print(a)
```

```python
# Fortgeschritten:
# Wandle eine Zahl in Binärcode um

n = int(input("Gib eine Zahl ein: "))
rest = [] # Liste für die Reste
while n > 0: # solange n größer als 0 ist
    rest.append(n % 2) # % ist der Modulo-Operator; das Ergebnis wird der Liste angehangen
    n = n // 2 # // ist die Ganzzahldivision

print(rest[::-1]) # [::-1] reversiert die Liste
```

Falls du nicht mehr weiß, wie eine Binärzahl aufgebaut ist, dann schau dir diesen Link an:
https://www.matheretter.de/wiki/binarzahlen

Übungsaufgabe 10

Leetspeak

Als Benutzer:in möchte das mein Text in Leetspeak umgewandelt wird, damit er nur von Insidern gut lesbar ist.

Akzeptanzkriterien:

Es wird eine vereinfachte Leetspeak verwendet: a ist 4, b ist 8, e ist 3, l ist 1, o ist 0, s ist 5 und t ist 7.

Verwende ein Dictionary (dict).

Aus “Meine Güte ist das ein langer Text.“ wird “M3in3 Gü73 i57 d45 3in 14ng3r 73x7. “

Lerne mehr über die Leetspeak: https://de.wikipedia.org/wiki/Leetspeak

Übungsaufgabe 11

Parkautomat mit Handicap

Als Benutzer:in möchte ich die pauschale Parkgebühr in Höhe von 2,50 EUR in Münzen zahlen können und erhalte bei einer Überzahlung auch mein Wechselgeld.

Akzeptanzkriterien:

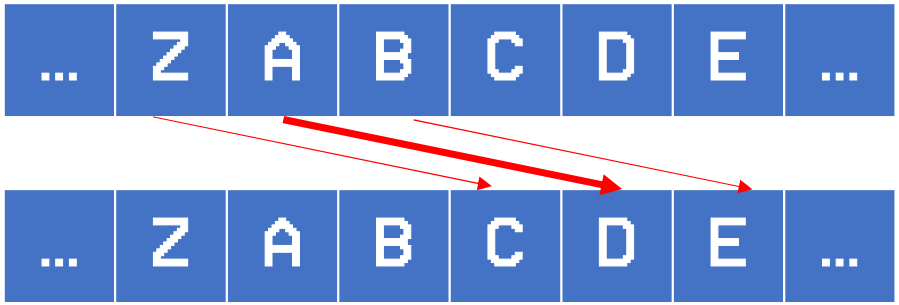
- Wir akzeptieren folgende Münzen: 10, 20 und 50 Cent. Andere Münzen werden mechanisch ausgesondert.
- Es ist daher davon auszugehen, dass der Benutzer eine beliebige Anzahl dieser Münzen in beliebiger Reihenfolge einwirft.
- Der Benutzer kann keine Münze mehr einwerfen, wenn 2,50 EUR erreicht oder mit dem letzten Einwurf überzahlt wurden.
- Das Wechselgeld wird angezeigt.
- Wurde exakt eingeworfen, dann wird „Danke“ angezeigt.
- Benutzereingabe (nacheinander): 50,50,50,50,20,20,20 führt zur Ausgabe: 10
- Benutzereingabe (nacheinander): 50,50,50,50,50 führt zur Ausgabe: Danke

Tipp: Nutze eine while-Schleife.

Übungsaufgabe 12

Geheimsprache

Eine der ersten Verschlüsselungen, die sogenannte Caesar-Verschlüsselung, beruhte auf dem einfachen System alle Buchstaben eines Textes durch einen anderen Buchstaben zu ersetzen. Der Schlüssel war eine einfache Zahl. Diese zeigte um wie viele Positionen der Buchstabe versetzt wurde. Wäre der Schlüssel eine 3, dann würde statt einem A ein D eingesetzt, usw.:



Als Benutzer möchte ich einen (kleingeschriebenen) Text eingeben können und danach noch den Schlüssel. Ich erhalte dann den verschlüsselten Text.

Akzeptanzkriterien:

- Bei der Eingabe „angriffammorgen“ und der Eingabe 7 als Schlüssel erhält man die Ausgabe: hunypmmhttvynlu
- Bei der Eingabe „rechteflankeamabend“ und der Eingabe 13 als Schlüssel erhält man: erpugrsynaxrnznoraq
- Bei der Eingabe „heute nicht angreifen“ und der Eingabe 9 als Schlüssel erhält man: qndcn wrlqc jwpanronw

Hier gibt es weitere Informationen zu dieser Verschlüsselung: https://de.wikipedia.org/wiki/Caesar-Verschl%C3%BCsselung

Tipp: Schau dir die Methode index() eines list Objektes an: https://docs.python.org/3/tutorial/datastructures.html

Übungsaufgabe 13

Entschlüsselung

Basierend auf Übungsaufgabe 12

Als Benutzer:in möchte ich einen Ceasar-verschlüssten Text und den Schlüssel eingeben, um dann den entschlüsselten Text zu erhalten.

Akzeptanzkriterien:

Bei der Eingabe „Aki tuc Oijud dqxj Guvqxh“ mit dem Schlüssel 16 wird ausgegeben: „Aus dem Osten naht Gefahr“

Übungsaufgabe 14

Brute Force Angriff

Basierend auf Übungsaufgabe 13

Als Benutzer:in möchte ich einen verschlüsselten Text angeben können – ohne den Schlüssel zu kennen und anzugeben -, der mit Ceasar verschlüsselt wurde und erhalte als Ausgabe alle möglichen Varianten mit Angabe des Schlüssels.

Akzeptanzkriterien:

Bei der Eingabe „Agzkbyy tnl Wxlmxg bg spxb Ttzxg“ erfolgt als Ausgabe (aufgrund der Länge nur Auszug):

(...)

Arkvmjj eyw Wiwxir mr daim Tekir 15

Aqjulii dxv Whvwhq lq czhl Tdjhq 16

Apitkhh cwu Wguvgp kp bygk Tcigp 17

Aohsjgg bvt Wftufo jo axfj Tbhfo 18

Angriff aus Westen in zwei Tagen 19

Amfqhee ztr Wdrsdm hm yvdh Tzfdm 20

Alepgdd ysq Wcqrcl gl xucg Tyecl 21

Akdofcc xrp Wbpqbk fk wtbf Txdbk 22

(...)