Short Summary

Data Types

- none → no data type
- bool → True, False
- int → 0, 1, -1, 2, … (supports +, -, *, /, %, ...)
- float → 0.1, -5.23, … (supports +, -, *, /, …)
- string → "Hallo!" (supports + to concatenate strings)

- Accessing parts of a string: a = "abcdefghij"
  a[2:10:3] → "cfi", a[-1:-9:-1] → "jihgfedc“
- Accessing the length of a string: len(a) → 10

- Type Conversion: int(), str(), float(), bool(), type()

Control Structures: if, elif, else

- The keywords if, elif and else start indented blocks
- The if-block is executed only if the condition after the if is fulfilled

```
zahl = int(input("Gib eine Zahl ein: "))
if zahl%3 == 0 and zahl%2 == 0:
    print(str(zahl) + " ist durch 6 teilbar")
```

```
Gib eine Zahl ein: 12
12 ist durch 6 teilbar
```

```
Gib eine Zahl ein: 4
```

```
Gib eine Zahl ein: 5
```

Control Structures: if, elif, else

- The keywords if, elif and else start indented blocks
- The elif-block is executed only if the if-block is not executed and the condition after the elif is fulfilled

```
zahl = int(input("Gib eine Zahl ein: "))
if zahl%3 == 0 and zahl%2 == 0:
    print(str(zahl) + " ist durch 6 teilbar")
elif zahl%3 == 0 or zahl%2 == 0:
    print(str(zahl) + " ist durch 3 oder durch 2 teilbar")
```

→
```
Gib eine Zahl ein: 12
12 ist durch 6 teilbar
```

→
```
Gib eine Zahl ein: 4
4 ist durch 3 oder durch 2 teilbar
```

→
```
Gib eine Zahl ein: 5
```

Control Structures: if, elif, else

- The keywords if, elif and else start indented blocks
- The else-block is only executed if none of the if-blocks and elif-blocks were executed

```
zahl = int(input("Gib eine Zahl ein: "))
if zahl%3 == 0 and zahl%2 == 0:
    print(str(zahl) + " ist durch 6 teilbar")
elif zahl%3 == 0 or zahl%2 == 0:
    print(str(zahl) + " ist durch 3 oder durch 2 teilbar")
else:
    print(str(zahl)+" ist weder durch 2 noch durch 3 teilbar")
```

```
Gib eine Zahl ein: 12
12 ist durch 6 teilbar

Gib eine Zahl ein: 4
4 ist durch 3 oder durch 2 teilbar

Gib eine Zahl ein: 5
5 ist weder durch 2 noch durch 3 teilbar
```

Control Structures: while

- The keyword while starts a new indented block, which is executed for as long as the condition after the while is fulfilled

```
zahl = int(input("Gib eine Zahl zwischen 0 und 20 ein: "))
while zahl < 0 or zahl > 20:
    print("Das war keine Zahl zwischen 0 und 20")
    zahl = int(input("Gib eine Zahl zwischen 0 und 20 ein: "))
```

→

```
Gib eine Zahl zwischen 0 und 20 ein: -5
Das war keine Zahl zwischen 0 und 20

Gib eine Zahl zwischen 0 und 20 ein: 28
Das war keine Zahl zwischen 0 und 20

Gib eine Zahl zwischen 0 und 20 ein: 84
Das war keine Zahl zwischen 0 und 20

Gib eine Zahl zwischen 0 und 20 ein: 14
```

Control Structures: for - in

- The keywords for *iterator* in *iterable* start a new indented block
- The block is executed for every instance the iterator can assume in the iterable

```
zaehler = 0
for i in range(15,4,-3):
    zaehler = zaehler+1
    print(str(zaehler)+". Durchgang: "+str(i))
```

```
1. Durchgang: 15
2. Durchgang: 12
3. Durchgang: 9
4. Durchgang: 6
```

Lists

- A list is an ordered assortment of values
- A list is generated by assigning some list elements

myList = [. . .], an empty list by myList = list()

```
Liste = [2,4,8,9,4,3]
leereListe = list()
```

- The keyword **in** is used to check, whether an element is in a list

```
print(5 in Liste)
print(4 in Liste)
```

→

```
False
True
```

Lists: Access to Lists

- Elements and parts of the list are accessed using […]
- Lists are iterable

```
abc = ["a", "b", "c", "d", "e", "f", "g", "h"]
print(abc[0:5:2])
for buchstabe in abc:
    print(buchstabe)
```

→

```
['a', 'c', 'e']
a
b
c
d
e
f
g
h
```

Lists: Methods on Lists

- Sort lists with sort() method

```
Liste.sort()
print(Liste)
```

```
[2, 3, 4, 4, 8, 9]
```

- Concatenate lists with "+" or extend()

```
abc1 = ["a", "b", "c"]
abc2 = ["d", "e", "f"]
abc = abc1 + abc2
print(abc)
abc1.extend(["d", "e"])
print(abc1)
```

```
['a', 'b', 'c', 'd', 'e', 'f']
['a', 'b', 'c', 'd', 'e']
```

- Reverse the order of the list items with reverse()

```
abc.reverse()
print(abc)
abc.reverse()
```

```
['f', 'e', 'd', 'c', 'b', 'a']
```

Lists: Methods on Lists

- Insert new items with insert() or append()

```
abc.insert(1, "ä")
abc.append("i")
print(abc)
```

→ `['a', 'ä', 'b', 'c', 'd', 'e', 'f', 'i']`

- Delete items with remove() or pop()

```
abc.pop(7)
abc.remove("ä")
print(abc)
```

→ `['a', 'b', 'c', 'd', 'e', 'f']`

Sets

- A set is disordered and does not have duplicates
- A set is generated by assigning set elements, i.e.

mySet = {. . . }, an empty set by mySet = set()

```
Menge = {"a", 5, "b", 7}
leereMenge = set()
```

- The keyword **in** is used to check, whether an element is in a set

```
print(5 in Menge)
print(4 in Menge)
```

→

```
True
False
```

- Sets are iterable

Sets: Set operations

- Unions of sets can be formed using the union() method or with the "|" sign
- The intersection of two quantities can be found using the intersection() method or with the "&" sign
- The difference of two sets can be formed with the difference() method or with "–"

```
Menge2 = {2,4,6,8}
Menge3 = {3,6,9,12}
print(Menge2 | Menge3)
print(Menge2 - Menge3)
print(Menge2 & Menge3)
```

```
{2, 3, 4, 6, 8, 9, 12}
{8, 2, 4}
{6}
```

Dictionaries

- A dictionary is a set of keys and associated values
- A dictionary is generated by giving the values to the respective keys, either with { } or with dict()

```
alph = {"a": "Ananas", "b": "Banane",
        "c":"Clementine", "d":"Datteln"}
alph["e"] = "Erdbeere"
```

- The keyword **in** is used to check, whether a key is in a dictionary
- Dictionaries are iterable

Dictionaries: Methods

alph.keys()	returns the list of keys
alph.values()	returns the list of values
alph.get(key)	returns the value of this key
alph.pop(key)	deletes the entry for this key
len(alph)	returns the number of entries
key **in** alph	returns true if the key exists
alph.clear()	deletes all entries

Classes

Classes are defined with the class keyword and need an __init__ method to create objects of this type.

```
class Person:
  def __init__(self, fname, lname):
    self.__firstname = fname
    self.__lastname = lname

  def getFirstname(self):
    return self.__firstname

  def getLastname(self):
    return self.__lastname

  def __str__(self):
    return self.__firstname + " " + self.__lastname
```

Inheritance

The customer class inherits from the person class. Therefore, the member variables and methods are inherited. The constructor of the base has to be called by **super**().__init__(…).

```
# Customer of a class
class Customer(Person):
  def __init__(self, id, lastname, firstname):
    super().__init__(lastname, firstname)
    self.__id = id

  def getId(self):
    return self.__id

  def getAccounts(self):
    return self.__accountList
```

Exception handling

To prevent the program from crashing, critical code can be wrapped into a `try ... except` construct

```
while True:
    try:
        v = input("Please enter an integer: ")
        x = int(v)
    except ValueError:
        print("Invalid input, please try again")
    else:
        return x
```

The input is repeated until the user enters an integer.

Output

```
Please enter an integer: a
Invalid input, please try again

Please enter an integer: b
Invalid input, please try again
```

File IO

We can open, read and write text from and to files.

Examples for reading and writing:

```
def speichereZahlen(path, anzahl):
    with open(path, "w") as outputFile:
        for i in range(1,anzahl+1):
            res = random(i)
            outputFile.write(str(res) + "\n")


def zahlenLesen(path):
    with open(path, "r") as inputFile:
        for line in inputFile:
            print(line)
```