# FS24 CAS PML - Python

## 5. Kontrollfluss: Schleifen

# Schleifen

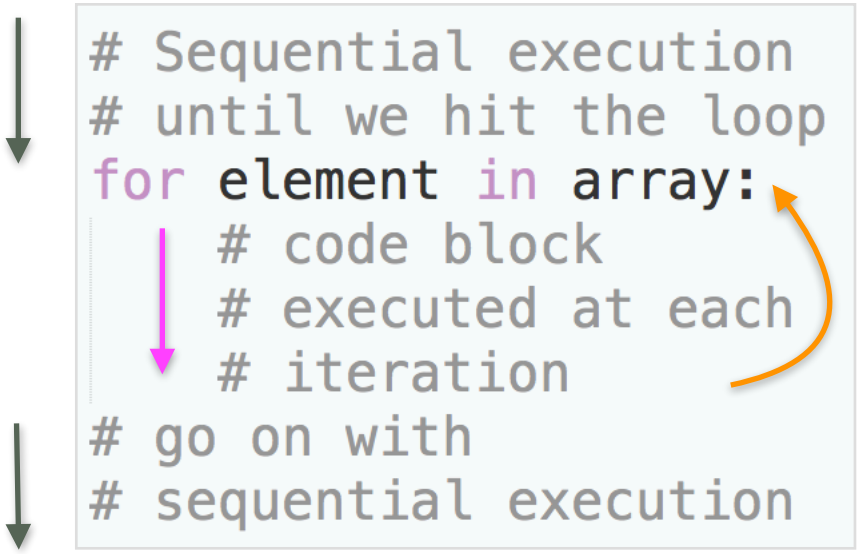
- Man will oft die selben Anweisungen mehrmals ausführen:
  - für jedes Element in einem array
  - bis eine gewisse Kondition erfüllt ist
- Schleifen (loops) erlauben das
- In python gibt es 2 Schleifen Typen:
  - *for*-loop
  - *while*-loop
- Die Anweisungen die wiederholt werden formen einen Code-Block

# Die *for*-loop

- Die *for*-loop iteriert über jedes element einer Sequenz
- Die Syntax ist:

```
array = [1,2,3]
for element in array:
    #do something with element
```

- Der Ausführungsfluss ist



# Die *for*-loop

- der Name *element* kann irgend ein gültiger Variablenname sein

```
message = "Hello!"
for letter in message:
    print(letter)
```

- Für jede Iteration wird das nächste Objekt von *array* zu *element* zugewiesen

```
array = [1, 2, 3]
for element in array:
    print(element)
print("after loop")
```

```
element = array[0]
print(element)
element = array[1]
print(element)
element = array[2]
print(element)
print("after loop")
```

# Die *for*-loop

- Man kann über irgend eine Sequenz iterieren
- Elemente in der Sequenz können verschiedene Typen haben

```
In [30]: mixed_list = [1, "hello", abs]
    ...: for el in mixed_list:
    ...:     el_type = type(el)
    ...:     print(el_type)
    ...:
<class 'int'>
<class 'str'>
<class 'builtin_function_or_method'>
```

# *range*

- *range()* definiert einen iterator über Ganzzahlen
- *range(n)* von *0* zu *n-1*
- *range(i,n)* von *i* zu *n-1*
- *range(i,n,step)* von *i* zu *n-1*, mit einem Schritt von *step*

```
In [31]: list(range(10))
Out[31]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [32]: list(range(1, 4))
Out[32]: [1, 2, 3]

In [33]: list(range(3, 20, 4))
Out[33]: [3, 7, 11, 15, 19]
```

# Die *for*-loop mit *range*

- mit range ist es einfach über die Indizien einer Sequenz zu iterieren

```
In [2]: my_list = ["first", "second"]
   ...: for i in range(len(my_list)):
   ...:     print(i, my_list[i])
   ...:
0 first
1 second
```

# Die *for*-loop mit *range*

- mit range ist es einfach über die Indizien einer Sequenz zu iterieren

```
In [4]:
   ...: list1 = ["a", "b"]
   ...: list2 = [2, 4]
   ...: for i in range(len(list1)):
   ...:     print(i, list1[i], list2[i])
   ...:
0 a 2
1 b 4
```

# Die *for*-loop mit *enumerate*

- *enumerate()* iteriert über die Indizien und Elemente einer Sequenz
- Für jede Iteration gibt es einen *tuple* zurück der den Index und das Element enthält

```
In [1]: my_list = ["first", "second"]
   ...: for pair in enumerate(my_list):
   ...:     print(type(pair), pair)
   ...:
<class 'tuple'> (0, 'first')
<class 'tuple'> (1, 'second')
```

# Die *for*-loop mit *zip*

- *zip()* iteriert gleichzeitig über mehrere Sequenzen
- Für jede Iteration gibt es einen *tuple* mit einem Element jeder Sequenz
- Die erste Iteration ergibt das erste Element jeder Sequenz, dann das zweite, usw.

```
In [2]: list1 = [1, 3]
   ...: list2 = [2, 4, 5]
   ...: for element_pair in zip(list1, list2):
   ...:     print(type(element_pair), element_pair)
   ...:
<class 'tuple'> (1, 2)
<class 'tuple'> (3, 4)
```

# Die *for*-loop mit mehreren Variablen

- Wenn jede Iteration eine Sequenz zurückgibt, kann man diese Sequenz mehreren Variablen zuweisen

```
In [10]: ll = [[11, 12, 13], [21, 22, 23]]
    ...: for i, j, k in ll:
    ...:     print(i,k)
    ...:
11 13
21 23
```

# Die *for*-loop mit mehreren Variablen

- Wenn jede Iteration eine Sequenz zurückgibt, kann man diese Sequenz mehreren Variablen zuweisen

```
In [12]: list1 = [1, 3]
    ...: for i, el in enumerate(list1):
    ...:     print(i, el)
    ...:
0 1
1 3
```

# Die *for*-loop mit mehreren Variablen

- Wenn jede Iteration eine Sequenz zurückgibt, kann man diese Sequenz mehreren Variablen zuweisen

```
In [13]: list1 = [1, 3]
    ...: list2 = [2, 4]
    ...: for el1, el2 in zip(list1, list2):
    ...:     print(el1, el2)
    ...:
1 2
3 4
```

# Die *while*-loop

- Die *while*-loop iteriert bis eine Kondition falsch ist.
- Die Syntax ist:

```
while condition:
    #do something
```

- Beispiel:

```
In [16]: i = 1
    ...: while i < 3:
    ...:     print(i)
    ...:     i += 1
    ...:
1
2
```

# Die *break* Anweisung

- *break* erlaubt eine Schleife zu stoppen
- Man verlässt die Schleife sobald die *break* Anweisung ausgeführt wird

```
In [18]: i = 1
    ...: while True:
    ...:     i += 1
    ...:     if i == 3:
    ...:         break
    ...:     print(i)
    ...:
2
```

# Die *continue* Anweisung

- *continue* erlaubt in einer Schleife zur nächsten Iteration zu gehen.

```
In [20]: # continue
     ...: for i in range(3):
     ...:     print(i)
     ...:     if i != 1:
     ...:         continue
     ...:     print("after condition")
     ...:
0
1
after condition
2
```

# Nested loops

- Eine Schleife kann in einer anderen Schleife enthalten sein (nested loop)

```
In [22]: for i in range(3):
    ...:     for j in ["one", "two"]:
    ...:         print(i, j)
    ...:
0 one
0 two
1 one
1 two
2 one
2 two
```

# Reminder

- for el in array:
- for i in range(n):
- zip(seq1, seq2)
- enumerate(seq)
- while condition: