# 20230511_Wirtschaftsinformatik_MPW2

May 11, 2023

[1]:
```
# Logische Statements
```

[2]:
```
# a == b "a ist gleich b"
# a = b "a ist gleich b"
# a != b "a ist nicht gleich b"
# a < b "a ist kleiner b"
# a <= b "a ist kleiner gleich b"
```

[3]:
```
# IF bedeutet in englisch "WENN"

a = 33
b = 200

if b > a:
    print('b ist größer als a')
```

```
b ist größer als a
```

[5]:
```
# "ELIF" in englisch bedeutet es "wenn der IF nicht zutrifft, dann probiere die␣
 ↪neue Kondition..."
# ELSE IF

a = 33
b = 33

if b > a:
    print('b ist größer als a')
elif a ==b:
    print('a und b sind gleich')
```

```
a und b sind gleich
```

[6]:
```
# 'ELSE' probiert eine Kondition wenn die vorherigen Bedingungen nicht erfüllt␣
 ↪wurden.

a = 200
b = 33
```

```
if b > a:
    print('b ist größer als a')
elif a ==b:
    print('a und b sind gleich')
else:
    print('b ist nicht größer als a')
```

```
b ist nicht größer als a
```

[7]:
```
# "AND" bedeutet in englisch "und"

a = 200
b = 33
c = 500

if a>b and c>a:
    print('beide Konditionen sind Wahr')
```

```
beide Konditionen sind Wahr
```

[8]:
```
# "OR" beudeutet in englisch "oder"

a = 200
b = 33
c = 500

if a>b or a>c:
    print('mindestens eine Kondition ist Wahr')
```

```
mindestens eine Kondition ist Wahr
```

[9]:
```
# 'NOT' beudeutet in englisch die Negation einer Kondition

a = 33
b = 200

if not a>b:
    print('a ist NICHT größer als b')
```

```
a ist NICHT größer als b
```

[14]:
```
# NESTED CONDITIONAL STATEMENT

x = 15

if x>10:
    print('Größer 10, ')
```

```
    if x > 20:
        print('und auch größer 20!')

    else:
        print('aber nicht größer 20.')

else:
    print('Ist nicht größer 10.')
```

```
Größer 10,
aber nicht größer 20.
```

[15]:
```
# LOOPS.

# Loops werden benutzt damit Python eine Tätigkeit mehrmals wiederholt.
# "WHILE" (während) und "FOR" (so lange)
```

[18]:
```
# WHILE

i = 1

while i < 12:
    print(i)
    i = i + 1
```

```
1
2
3
4
5
6
7
8
9
10
11
```

[20]:
```
# mit dem 'BREAK' Befehl können wir den Loop abbrechen, auch wenn es Wahr ist

i = 1

while i < 6:
    print(i)
    if i == 3:
        break
    i = i + 1
```

```
1
```

```
2
3
```

[21]:
```
# Übung. bitte schreiben Sie alle Zahlen zw. 1 und 5
# und dann geben Sie eine Fehlermeldung "i ist nicht mehr kleiner als 6"

i = 1

while i < 6:
    print(i)
    i = i + 1
else:

    print('i ist nicht mehr kleiner als 6')
```

```
1
2
3
4
5
i ist nicht mehr kleiner als 6
```

[22]:
```
# ein FOR loop wird benutzt um über eine Sequenz zu Iterieren

obst = ['banane', 'apfel', 'birne']

for x in obst:
    print(x)
```

```
banane
apfel
birne
```

[24]:
```
for x in 'apfel':
    print(x)
```

```
a
p
f
e
l
```

[25]:
```
# Um eine Reihe von Codes eine bestimmte Anzahl von Malen zu durchlaufen,
# können wir die Funktion "range()" verwenden.
# Die Funktion "range() gibt eine Zahlenfolge zurück, die standardmäßig bei 0␣
 ↪beginnt,
# und um 1 erhöht wird und bei einer bestimmten Zahl endet."
```

[26]:
```
for x in range(6):
    print(x)
```

```
0
1
2
3
4
5
```

[27]:
```
# der Startwert kann geändert werden...

for x in range(2,6): # startwert ist 2 aber nicht bis einschließlich 6

    print(x)
```

```
2
3
4
5
```

[28]:
```
# den Inkrementwert kann geändert werden

for x in range(2, 30, 3): # anfangswert ist 2, ende bei 30 (nicht␣
 ↪einschließlich), Inkrementwert ist 3.
    print(x)
```

```
2
5
8
11
14
17
20
23
26
29
```

[29]:
```
# FUNKTIONEN
```

[30]:
```
# Eine Funktion ist ein Codeblock, der nur ausgeführt wird, wenn er aufgerufen␣
 ↪wird.
# Sie können Daten, sogenannte Parameter, an eine Funktion übergeben.
# Eine Funktion kann als Ergebnis Daten zurückgeben.
# Funktionen werden in Python mit dem Schlüsselwort "def" definiert.
```

[37]:
```
def my_function_1():
    print('Hello aus einer Funktion!')
```

[38]:
```
my_function_1() # function wird gerufen
```

```
Hello aus einer Funktion!
```

[33]:
```
# Informationen können als Parameter oder Argumente an Funktionen übergeben␣
 ↪werden.
# Argumente werden nach Funnktionsnamen innerhalb der Klammern angegeben.
# Es können beliebige Argumente hinzugefügt werden, immer aber durch Komma␣
 ↪getrennt.
```

[39]:
```
def my_function_2(fname):
    print(fname + ' Referenzname')

my_function_2('Emil')
```

```
Emil Referenzname
```

[40]:
```
# Wir können beliebige Datentypen von Argumenten an eine Funktion Senden␣
 ↪(Zeichenfolgen, Zahl, Listen, usw.), und
# sie werden innerhalb der Funktion als derselbe Datentyp behandelt.
# Hier senden wir in dem Beispiel eine Liste als Argument:

def my_function_3(food):
    for x in food:
        print(x)

fruits = ['apple', 'banana', 'cherry']

my_function_3(fruits)
```

```
apple
banana
cherry
```

[41]:
```
# Um eine Funktion einen Wert zurückgeben zu lassen, verwenden wir die "return"␣
 ↪Anweisung

def my_function_4(x):
    return 5*x

print(my_function_4(9))
```

```
45
```

[42]:
```
# Übung. Bitte schreiben Sie eine Funktion, welche den Maximum von 3 Zahlen␣
↪zurückgibt

def max_of_two(x,y):
    if x>y:
        return x
    return y

def max_of_three(x,y,z):
    return(max_of_two(x, max_of_two(y,z)))

print(max_of_three(3,6,-3))
```

```
6
```

[43]:
```
# Übung. Bitte schreiben Sie eine Funktion, welche die Summe aller Zahlen in␣
↪einer Liste liefert.

def summe(numbers):
    total = 0
    for x in numbers:
        total = total + x
    return total

liste = (8,2,3,0,7)

print(summe(liste))
```

```
20
```

[44]:
```
# Übung. bitte schreiben Sie eine Funktion, welche die Geradezahlen zwischen 4␣
↪und 30 ausgibt.

def gerade_zahlen(a,b):
    return print(list(range(a,b,2)))

gerade_zahlen(4,30)
```

```
[4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28]
```

[ ]:
```

```