



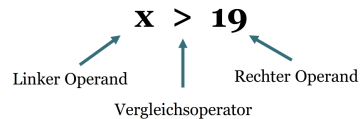
Programmieren für Einsteiger

Python cheat sheet - Tag 2

Vergleiche

Aussagenlogische Vergleiche vergleichen zwei Elemente miteinander und liefern den Wahrheitswert **True** oder **False** zurück.

Operator	Erklärung
<	kleiner
<=	kleiner gleich
>	größer
>=	größer gleich
==	gleich
!=	ungleich
in	Zugehörigkeit
not in	nicht-Zugehörigkeit



```
'mist' in 'Pessimist' -> True
'Mist' in 'Pessimist' -> False
'a' > 'B' -> True
not ((2 > 3) and (1 != 1)) -> True
```

Logik

Aussagenlogische Vergleiche lassen sich durch logische Operatoren miteinander verknüpfen. In Python gibt es folgende drei logische Operatoren:

- Konjunktion, das logische *und* (**and**)
- Disjunktion, das logische *oder* (**or**)
- Negation, das logische *nicht* (**not**)

Operand A	Operand B	A or B
True	True	True
True	False	True
False	True	True
False	False	False

Operand A	Operand B	A and B
True	True	True
True	False	False
False	True	False
False	False	False

Operand A	not A
True	False
False	True

Zufallszahlen

Mit dem Package random lassen sich Zufallszahlen erzeugen

```
import random
zufallszahl = random.randint(0,100)
```

range Funktion

Mittels der range Funktion erhält man eine Folge von Ganzzahlen (Integern)

```
Stopp
range(6) -> [0, 1, 2, 3, 4, 5]

Start Stopp
range(2, 6) -> [2, 3, 4, 5]

Start Stopp Schritt
range(1, 6, 2) -> [1, 3, 5]
```

Bedingte Anweisungen

Mit if-Anweisungen werden einzelne oder eine Folge von Anweisungen in Abhängigkeit von einer Bedingung ausgeführt.

```
a = 5
b = 19
if a > b:
    print('a ist größer als b.')
elif a < b:
    print('a ist kleiner als b.')
else:
    print('a und b sind gleich groß.')
```

if Bedingung:
(Einrückung ->) Anweisungsblock
elif Bedingung:
(Einrückung ->) Anweisungsblock
else:
(Einrückung ->) Anweisungsblock

```
x = 1
y = 2
z = int(input())

if z > x + y:
    x = z // 2
    y = 3
    z = x - y
print(z)
```

← äußerer Anweisungsblock
← innerer Anweisungsblock

Einrückungen definieren in Python zusammenhängende Anweisungsfolgen. Man spricht von Anweisungs- oder Codeblöcken.

Mit der Anweisung **continue** wird die aktuelle Wiederholung der Schleife abgebrochen und die nächste Wiederholung gestartet.
Mit der Anweisung **break** wird die Ausführung einer Schleife sofort beendet und das Programm fährt bei der ersten Anweisung nach der Schleife fort.

Bedingte Schleifen

Bedingte Schleife werden durch eine while Anweisung erzeugt und der dazugehörige Anweisungsblock so lange wiederholt, bis die Schleifenbedingung nicht mehr erfüllt ist.

while Bedingung:
(Einrückung ->) Anweisungsblock

```
a = 2
while a <= 256: #Solange a kleiner als 256 verdopple a
    a = a * 2
```

Iterationsschleifen

Iterationsschleifen werden durch eine **for in** Anweisung erzeugt und der dazugehörige Anweisungsblock so lange wiederholt, bis jedes Element der gegebenen Kollektion "abgearbeitet" ist.

for Element in Kollektion:
(Einrückung ->) Anweisungsblock

```
farben = ('grün', 'hellgrün', 'dunkelgrün')
for farbe in farben:
    print(farbe)
```



Programmieren für Einsteiger

Python cheat sheet - Tag 2

Programmfehler

Prinzipiell gibt es folgende 3 Arten von Fehlern beim Programmieren:

Syntaktische Fehler:

- Fehler in der Grammatik von Anwendungen
- werden beim Start des Programmes vom Pythoninterpreter überprüft
- z.B. SyntaxError

Laufzeitfehler:

- treten erst auf, wenn das Programm schon läuft
- Problem beim Ausführen einer Aufgabe
- z.B. ValueError

Semantikfehler (Logikfehler):

- das Programm läuft komplett durch, ohne einen Fehler zu melden
- das Ergebnis entspricht nicht den Erwartungen
- z.B. wenn man irgendwo `**` statt `*` gerechnet hat

Funktionen

Funktionen sind aufrufbare Anweisungsblöcke, welche eine spezifische Teilaufgabe eines Programms lösen sollen.

def funktionsname(parameter):

(Einrückung ->) Anweisungsblock

```
def f(x):  
    x = x + 1  
    return x
```

mathematische Funktion:
 $f(x) = x + 1$

Exceptions

Zur Laufzeit auftretende Exceptions können durch sogenannte **try ... except** Blöcke abgefangen und behandelt werden.

```
try:  
    (Einrückung ->) Anweisungsblock  
except (Ausnahmetyp):  
    (Einrückung ->) Anweisungsblock
```

```
try:  
    zahl = int(input('Bitte gib eine ganze Zahl ein: '))  
    ergebnis = 5 / zahl  
    print('Danke für die Zahl. Das Ergebnis ist', ergebnis)  
except ValueError:  
    print('Die Eingabe war nicht in Ordnung und wird auf den Wert 19 gesetzt.')  
    zahl = 19  
    ergebnis = 5 / zahl  
    print('Danke für die Zahl. Das Ergebnis ist', ergebnis)  
except ZeroDivisionError:  
    print('Division durch 0 ist nicht möglich. Ergebnis wird auf 1 gesetzt.')  
    ergebnis = 1
```