

Python BU

01_Verzweigungen

If, elif, while

Verzweigung mit if: Einstieg

Programmiersprachen werden eigentlich erst in dem Moment interessant, wo sie es erlauben, dass bestimmte Programmteile manchmal ausgeführt werden und manchmal nicht.

Das if-Kommando zum Beispiel, wertet eine Bedingung aus und führt ein Programmstück nur dann aus, wenn die Bedingung wahr ist.

```
if i > 10:  
    print "i ist größer als 10!"  
    j = 2*i+1
```

Beachten Sie den Doppelpunkt am Zeilenende hinter der Bedingung.

Pythons Philosophie, Einrückung zur Darstellung der Programmstruktur zu verwenden, ist für viele Programmierer zunächst ungewohnt und abschreckend, aber hat durchaus Vorteile, weil sie Fehler verhindern kann.

Verzweigung mit if: Boolesche Ausdrücke

Der Ausdruck hinter dem `if` muss »wahr« sein, damit die vom `if` abhängige Kommandofolge ausgeführt wird.

Als »wahr« betrachten wir in diesem Zusammenhang alles, was nicht »falsch« ist.

»Falsch« wiederum sind Werte, die numerisch Null sind (egal, welche Typen - ganzzahlig, Gleitkomma oder komplex), und einige andere Werte, die Sie noch nicht gesehen haben.

Vergleichsoperatoren liefern immer einen der Werte `False` (0) oder `True` (1).

Beide Operanden müssen denselben Typ haben.

Numerische Operanden werden notfalls umgewandelt - etwa von ganzen in Gleitkomma-Zahlen-, bis sie vergleichbar werden.

Operator	Bedeutung
<code>==</code>	Gleichheit
<code>!=</code>	Ungleichheit
<code><></code>	Ungleichheit (Python 2, verpönt)
<code><</code>	Kleiner als
<code><=</code>	Kleiner als oder gleich
<code>></code>	Größer als
<code>>=</code>	Größer als oder gleich
<code>is</code>	Objektidentität
<code>is not</code>	Objekt-Nichtidentität

Vergleichsoperatoren in Python

else und elif

Das if-Kommando ist nicht nur dazu zu gebrauchen, eine Kommandofolge auszuführen, wenn eine Bedingung erfüllt ist.

Sie können auch alternativ eine Kommandofolge ausführen, wenn die Bedingung »wahr« ist, und eine andere, wenn sie »falsch« ist. Dazu dient else:

```
if n % 2 == 0:  
    print("n ist gerade")  
else:  
    print("n ist ungerade")
```

```
if n % 2 == 0: print ("n ist gerade")  
else: print ("n ist ungerade")
```

(Wichtig: Auch hinter dem else steht ein Doppelpunkt.) !

Verschachtelten ifs und else s

Natürlich können Sie ifs (und else s) auch verschachteln. Sie müssen nur die Einrückung konsequent einhalten. Zum Beispiel:

```
if x <= 0:
    print("x ist Null oder negativ")
else:
    if x < 1000:
        print("x ist kleiner als 1000")
    else:
        if x < 1000000:
            print("x ist kleiner als 1000000")
        else:
            print("x ist gigantomatisch riesengroß!")
```

Elif statt else if

```
if x <= 0:
    print("x ist Null oder negativ")
elif x < 1000:
    print("x ist kleiner als 1000")
elif x < 1000000:
    print("x ist kleiner als 1000000")
else:
    print("x ist gigantomanisch riesengroß!")
```

Hier werden die Bedingungen nach dem if bzw. den elifs nacheinander geprüft, bis sich eine als »wahr« herausstellt. Die Kommandofolge, die dieser Bedingung folgt, wird ausgeführt.

Ist keine der Bedingungen »wahr«, dann wird die Kommandofolge hinter dem else ausgeführt.

Natürlich muss es nicht zwingend ein else geben.

Gibt es keins, dann passiert in dem Fall, dass keine der if / elif-Bedingungen »wahr« war, eben einfach nichts

Case und Switch- Anweisungen

Die CASE- oder switch-Anweisung, die viele Programmiersprachen für Fallunterscheidungen gemäß dem Wert einer Variablen unterstützen, werden Sie in Python vergeblich suchen.

Sie können sie aber durch eine if-elif-Kette

Schleifen mit while

Nachdem Sie jetzt gelernt haben, wie Fallunterscheidungen mit if funktionieren, ist der nächste Schritt, Ihnen zu zeigen, wie Schleifen funktionieren.

Die einfachste Form von Schleife, die in Python zur Verfügung steht, ist der Fallunterscheidung prinzipiell gar nicht unähnlich:

```
i = 1  
while i <= 5:  
    print i  
    i = i + 1
```



```
1  
2  
3  
4  
5
```

```
while i <= 5: print i; i = i + 1
```


While - VS if Schleife

Der einzige Unterschied zwischen if und while besteht darin, dass bei if nach der Ausführung der Kommandofolge mit dem Rest des Programms weitergemacht wird.

Bei while dagegen wird zum Anfang der Schleife zurückgesprungen und die Bedingung nochmals ausgewertet.

Das wiederholt sich, bis die Auswertung der Bedingung »falsch« ergibt- und dann wird (wie bei if) die Kommandofolge übersprungen und mit dem Rest des Programms fortgefahren.

Wenn die Bedingung schon bei der allerersten Auswertung »falsch« ergibt, dann wird die Kommandofolge einfach übersprungen und mit dem Rest des Programms weitergemacht.

Man spricht von einer »abweisenden« Schleife.

$i = i + 1$ Ausdruck Mathematisch gesehen unsinnig oder?

Der Python-Operator `=` dient nicht dazu , eine mathematische Gleichung aufzustellen, sondern eine Abkürzung ist für:

1. Bestimme den Wert des Ausdrucks auf der rechten Seite
2. schreibe ihn in die Variable auf der linken Seite«,

dann ergibt es schon mehr Sinn.

Unter dem Strich erhöht der Ausdruck den in der Variablen `i` gespeicherten Wert um 1.

Dafür gibt es übrigens eine Kurzschreibweise:

```
i = i + 1
```

ist dasselbe wie

```
i += 1
```

Die Kurzschreibweise gibt es auch für die anderen Operatoren: `-=`, `*=`, `/=`, `//=`, `%=` und `**=`.

Die Bedeutung ist jeweils sinngemäß dieselbe.

Diese »erweiterten Zuweisungen« funktionieren nicht nur für Zahlen, sondern zum Beispiel auch für Zeichenketten:

Kurzschreibweise für erweiterten Zuweisungen

Bsp. Zeichenketten

Hier die einzige Vorbedingung ist, dass die zugrundeliegenden Operatoren wie + oder * für den betreffenden Datentyp definiert sind.

```
>>> s = "123"  
>>> s += "X"  
>>> s  
'123X'  
>>> s *= 3  
>>> s  
'123X123X123X'
```

break und continue


Break:

Hin und wieder kommt es vor, dass eine Schleife vorzeitig abgebrochen werden muss (zum Beispiel weil eine Fehlersituation aufgetreten ist).

In Python dient dazu das break-Kommando.

Es beendet die innerste umgebende Schleife:

```
s0 = 'Habe nun, ach! Philosophie, Juristerei und Medizin'  
s1 = ''  
i = 0  
while i < len(s0):  
    s1 += s0[i]          # i-tes Zeichen von s0  
    if s0[i] == '!': break  
  
    i += 1  
print s1
```



Habe nun, ach!

break und continue

Continue:

Zusätzlich zum break gibt es auch noch das Kommando continue (ahnl. Wie goto in C).

Mit continue können Sie aus der Mitte einer Schleife heraus wieder an deren Anfang springen und den nächsten Durchlauf beginnen - wobei bei while dafür zuallerst die Bedingung getestet wird.

Zusammenfassung

- Python nutzt Einrückung zur Darstellung der Programmstruktur.
- In Python gibt es einen Booleschen Datentyp, der aber bis auf die Werte True und False dem ganzzahligen Datentyp entspricht.
- Python unterstützt diverse Vergleichsoperatoren, die Boolesche Ergebnisse liefern.
- Die logischen Operatoren sind not, and und or.
- Python erlaubt die bedingte Ausführung von Programmteilen mit if und else.
- Ketten von Vergleichen lassen sich oft mit elif vereinfachen.
- Python unterstützt keine CASE- bzw. switch-Verzweigung.