Themen

weitere Themen siehe Seite B-3

Operatoren

Operatoren werden in Programmiersprachen wie Python genutzt, um Variablen Werte zuzuweisen und um in Variablen gespeicherte Werte zu verarbeiten.

Zuweisungsoperatoren

Das Gleichheitszeichen dient als Zuweisungsoperator. Dank des Gleichheitszeichens ist es möglich, einer Variablen einen Wert zuzuweisen.

Berechnungsoperatoren

Die Berechnungsoperatoren werden für die bekannten Grundrechenarten verwendet. Daneben können mit dem Operator ** Zahlen potenziert werden.

Der Operator // liefert als Ergebnis der Division eine ganze Zahl. Auch beim Modulo-Operator % wird eine Division ausgeführt. Als Ergebnis erhält man den Rest der Division.

+	Addition	10 + 3 = 13
-	Subtraktion	10 - 3 = 7
*	Multiplikation	10 * 3 = 30
**	Potenzieren	10 ** 3 = 1000
/	Division	10 / 3 = 3.33333333333
//	Ganzzahldivision	10 // 3 = 3
%	Modulo-Operator	10 % 3 = 1

Vergleichsoperatoren

Vergleichsoperatoren werden in Abfragen genutzt, um die Bedingung zu formulieren. Das Ergebnis dieser Abfragen lautet entweder `true` oder `false`.

>	größer als	<=	kleiner als oder gleich
<	kleiner als	==	gleich
>=	größer als oder gleich	!=	ungleich

Logische Operatoren

Mit Hilfe der logischen Operatoren `and`, `or` und `not` lassen sich mehrere Bedingungen miteinander verknüpfen.

Das Ergebnis ist wahr (`true`), wenn ...	
and	... beide Bedingungen erfüllt sind.
or	... eine von mehreren Bedingungen erfüllt ist.
not	... die betreffende Bedingung nicht erfüllt ist.

Teilmengenoperatoren

Der Teilmengenoperator `in` wird eingesetzt, um zu überprüfen, ob ein Element Teil einer Menge ist. Beispielsweise lässt sich damit feststellen, ob eine Liste ein bestimmtes Wort oder eine bestimmte Zahl enthält. Als Ergebnis erhält man `true` oder `false`.

```
"A" in ["A","B","C"]
"A" not in ["A","B","C"]
```

Rangfolge der Operatoren

Beim Rechnen kennen wir die Regel „Punktrechnung vor Strichrechnung“. Auch bei den Operatoren in Python gibt es eine Reihenfolge, in der sie ausgewertet werden. Die folgende Tabelle zeigt die Operatoren und die Priorität ihrer Abarbeitung.

Priorität		
	**	Potenzieren
	*, /, //, %	Multiplikation, Division, Modulo
	+, -	Addition, Subtraktion
	<, <=, >, >=, !=, ==	Vergleichsoperatoren
	in, not in	Teilmengenoperatoren

Verkettung mit Hilfe von Operatoren

Mit Hilfe der Operatoren , oder + können Werte kombiniert werden. Man spricht dabei auch vom „Verketten“ der Werte.

Möchte man Zahlen und Text miteinander über den Operator + verketten, muss man die Zahl mit Hilfe der Funktion `str` in eine Zeichenkette umwandeln.

```
name = "Anna"
alter = 15
print name,"ist",alter,"Jahre alt."
print name+"ist"+str(alter)+"Jahre alt."
```

```
Anna ist 15 Jahre alt.
Annaist15Jahre alt.
```

Es fällt auf, dass die Werte mit dem Operator + ohne Leerzeichen verkettet werden, während bei Verwendung des Operators , automatisch ein Leerzeichen eingefügt wird.

Operatoren

Aufgabe 1
Welche Ausgaben sind nach Ablauf dieser Programmzeilen im Ausgabefenster zu erwarten?

a)	`print 100 + 25`	`125`	e)	`print 3 * 5 + 8`	`23`
b)	`print 2 ** 5`	`32`	f)	`print 3 ** 3 % 5`	`2`
c)	`print 80 // 3`	`26`	g)	`print 60 / 5 - 2`	`10.0`
d)	`print 25 % 4`	`1`	h)	`print 3 * 15 // 4`	`11`

Aufgabe 2
Welche Ausgaben sind nach Ablauf dieser Programmzeilen im Ausgabefenster zu erwarten?

a)	`print 5 in [2, 4, 6, 8]`	`False`
b)	`print "AB" in "RHABARBER"`	`True`
c)	`print 10 not in [10, 20, 30, 40]`	`False`
d)	`print "T" in ["H", "O", "R", "B"]`	`False`
e)	`print "N" not in "STUTTGART"`	`True`
f)	`print 1 in [-2, -1, 0, 1, 2]`	`True`
g)	`print 10**2 not in [500, 1000, 1500]`	`True`
h)	`print 10 % 3 in [3, 6, 9]`	`False`

Aufgabe 3
Welche Ausgaben sind nach Ablauf dieser Programmzeilen im Ausgabefenster zu erwarten?

a)	`print 7>5 and 14>7`	`True`
b)	`print 3<8 or 3>8`	`True`
c)	`print not 10<12`	`False`
d)	`print 20>10 and not 10>5`	`False`

Operatoren

Aufgabe 4
Welche Ausgabe ist nach Ablauf dieses Programms im Ausgabefenster zu erwarten?

```
berg = "Feldberg"
hoehe = 1277
einheit = "Meter"
gebirge = "Schwarzwald"

print "Der",berg,"im",gebirge
print "ist"+str(hoehe)+einheit+"hoch."
```

```
Der Feldberg im Schwarzwald
ist1277Meterhoch.
```

Aufgabe 5
Schreibe ein Programm, das den Namen und den Wohnort abfragt und beides in der Form „X wohnt in Y“ wieder ausgibt.

```
name = input("Gib deinen Namen ein")
ort = input("Gib deinen Wohnort ein")

print name,"wohnt in",ort

oder

print name+" wohnt in "+ort
```

Beispiel:

```
Linus wohnt in Wangen im Allgäu
```

Aufgabe 6
Schreibe ein Programm, das
- fünf Variablen mit den folgenden Werten anlegt und
- sie mit dem +-Operator verkettet und ausgibt

Texte: Im Jahr
leben
Milliarden Menschen auf der Erde

Zahlen: 2020
7.8

```
text1 = "Im Jahr "
jahr  = 2020
text2 = " leben "
bevoelkerung  = 7.8
text3 = " Milliarden Menschen auf der Erde"

print text1+str(jahr)+text2+str(bevoelkerung)+text3
```

```
Im Jahr 2020 leben 7.8 Milliarden Menschen auf der Erde
```

for-Schleifen

Schleifen werden in Programmen genutzt, um Anweisungen mehrmals ausführen zu lassen. Jede Schleife enthält eine Bedingung, die vor jeder Wiederholung geprüft wird, damit die Schleife nicht endlos läuft.

In Python gibt es zwei Arten von Schleifen, die so genannten for-Schleifen und while-Schleifen.

for-Schleifen

Eine for-Schleife wird verwendet, wenn bekannt ist, wie häufig die zu wiederholenden Anweisungen ausgeführt werden sollen.

Die Steuerung von for-Schleifen erfolgt durch Variablen, die als Zähler dienen. for-Schleifen werden deshalb auch als Zählschleifen bezeichnet.

Die Zählvariablen in for-Schleifen werden üblicherweise mit Kleinbuchstaben wie i, j, k etc. benannt.

for-Schleifen können auch verschachtelt werden. Das heißt, dass in eine äußere for-Schleife mit der Zählvariablen i eine oder mehrere for-Schleifen mit den Zählvariablen j, k usw. eingebettet werden können.

```
for i in (1, 2):
    print i
    for j in ("A", "B"):
        print j
```

Nach dem einleitenden `for` folgt die Bedingung, bei deren Erfüllung die enthaltenen Anweisungen ausgeführt werden. Diese Zeile wird mit einem Doppelpunkt abgeschlossen, die nachfolgenden Anweisungen werden eingerückt.

In unserem Beispiel kann die Zählvariable i nacheinander die Werte 1 und 2 annehmen. In jedem Durchlauf der äußeren Schleife wird die zweite Schleife durchlaufen. Deren Zählvariable j kann die Werte A und B annehmen. Entsprechend erhält man für unser Beispiel die folgende Ausgabe:

```
1
A
B
2
A
B
```

range()-Funktion

In einfachen Fällen wie in unserem Beispiel lassen sich alle möglichen Werte, die die Zählvariablen annehmen können, direkt in der Bedingung notieren. Ist die Anzahl der möglichen Werte größer, benötigt man eine Funktion, die eine Liste dieser Werte erzeugt.

Hierfür kann die Funktion `range()` genutzt werden. Sie kann mit einem, zwei oder drei Parametern genutzt werden, abhängig davon, welche Werte die Liste enthalten soll. Als Parameter sind ganze Zahlen zulässig, die aber sowohl positiv als auch negativ sein dürfen.

Mit einem Parameter erzeugt man eine Liste ganzer Zahlen, die stets mit Null beginnt und in Einerschritten bis zur Zahl unterhalb des Parameterwertes reicht. Der Parameter selbst ist nicht Teil der Liste.

```
print range(4)
print range(2,6)
print range(4,16,4)
print range(2,-6,-2)
```

```
[0, 1, 2, 3]
[2, 3, 4, 5]
[4, 8, 12]
[2, 0, -2, -4]
```

Für eine Liste, die nicht mit Null beginnt, muss man die Funktion `range()` mit zwei Parametern verwenden. Das Ergebnis beginnt in diesem Fall mit dem ersten Parameter und endet eins unter dem zweiten Parameter.

Der dritte Parameter in der Funktion `range()` ermöglicht Listen, die nicht in Einserschritten ansteigen. Der erste der drei Parameter ist zugleich der Startwert der Liste. Der dritte Parameter definiert die Schrittweite. Der zweite Parameter markiert den Wert, der gerade nicht mehr Teil der Liste ist. Durch die Verwendung des dritten Parameters sind so auch absteigende Listen möglich.

In for-Schleifen lässt sich mit Hilfe der `range()`-Funktion ein Zahlenbereich erzeugen, den die Zählvariable i durchläuft. In unserem Beispiel wären das 10 Wiederholungen von i = 0 bis i = 9.

```
for i in range(10):
```

for-Schleifen

Aufgabe 1
Welche Ausgaben sind nach Ablauf dieser Programmzeilen im Ausgabefenster zu erwarten?

a)
```
print range(1,10,2)
```

b)
```
print range(-2,-10,-4)
```

c)
```
print range(3)
```

d)
```
print range(15,20)
```

e)
```
print range(-5,0)
```

f)
```
print range(12,4,-2)
```

g)
```
print range(10,90,15)
```

a)
```
[1, 3, 5, 7, 9]
```

b)
```
[-2, -6]
```

c)
```
[0, 1, 2]
```

d)
```
[15, 16, 17, 18, 19]
```

e)
```
[-5, -4, -3, -2, -1]
```

f)
```
[12, 10, 8, 6]
```

g)
```
[10, 25, 40, 55, 70, 85]
```

Aufgabe 2
Schreibe ein Programm, mit dessen Hilfe die Lottozahlen 6 aus 49 ermittelt und ausgegeben werden können.

```
from random import *
seed()

print sample(range(1,50),6)
```

Aufgabe 3
Lass den Computer im übertragenen Sinne 6000 mal würfeln und anschließend ausgeben, wie häufig die einzelnen Augenzahlen gefallen sind.

- Weise dazu sechs Variablen den Wert 0 zu.
- Verwende eine for-Schleife, um die Anzahl der gewürfelten Zahlen zu zählen.
- Erhöhe nach jedem Würfeln den Wert der Variable, die zur jeweiligen Augenzahl gehört.
- Gib nach dem Durchlaufen der Schleife aus, wie häufig die einzelnen Ziffern gefallen sind

Beispiellösung:

```
from random import *
seed()

anzahl1 = 0
anzahl2 = 0
anzahl3 = 0
anzahl4 = 0
anzahl5 = 0
anzahl6 = 0

for i in range(6000):
    augenzahl = randint(1,6)

    if augenzahl == 1:
        anzahl1 = anzahl1 + 1
    elif augenzahl == 2:
        anzahl2 = anzahl2 + 1
    elif augenzahl == 3:
        anzahl3 = anzahl3 + 1
    elif augenzahl == 4:
        anzahl4 = anzahl4 + 1
    elif augenzahl == 5:
        anzahl5 = anzahl5 + 1
    else:
        anzahl6 = anzahl6 + 1

print "Eins:",anzahl1
print "Zwei:",anzahl2
print "Drei:",anzahl3
print "Vier:",anzahl4
print "Fünf:",anzahl5
print "Sechs:",anzahl6
```

for-Schleifen

Aufgabe 4

Schreibe Programme, die die Zeilen- und Spaltenzahl abfragen und anschließend die folgenden Muster ausgeben.

Die Programme sollen beliebige Zeilen- und Spaltenanzahlen erlauben und entsprechend größere oder kleinere Figuren erzeugen.

Beispiellösungen:

a)
```
OOOOO
OOOOO
OOOOO
OOOOO
OOOOO
```

```
groesse = input("Anzahl Zeilen/Spalten")
ausgabe = ""
for zeile in range(groesse):
    for spalte in range(groesse):
        ausgabe = ausgabe + "O"
    print ausgabe
    ausgabe = ""
```

b)
```
O----
-O---
--O--
---O-
----O
```

```
groesse = input("Anzahl Zeilen/Spalten")
ausgabe = ""
for zeile in range(groesse):
    for spalte in range(groesse):
        if spalte == zeile:
            ausgabe = ausgabe + "O"
        else:
            ausgabe = ausgabe + "-"
    print ausgabe
    ausgabe = ""
```

c)
```
OOOOO
-OOOO
--OOO
---OO
----O
```

```
groesse = input("Anzahl Zeilen/Spalten")
ausgabe = ""
for zeile in range(groesse):
    for spalte in range(groesse):
        if spalte >= zeile:
            ausgabe = ausgabe + "O"
        else:
            ausgabe = ausgabe + "-"
    print ausgabe
    ausgabe = ""
```

d)
```
OOOOO
O---O
O---O
O---O
OOOOO
```

```
zeilen = input("Anzahl Zeilen eingeben")
spalten = input("Anzahl Spalten eingeben")
ausgabe = ""
for zeile in range(zeilen):
    for spalte in range(spalten):
        if zeile == 0 or spalte == 0 or zeile == zeilen-1 or spalte == spalten-1:
            ausgabe = ausgabe + "O"
        else:
            ausgabe = ausgabe + "-"
    print ausgabe
    ausgabe = ""
```