Python Syntax

Vorkurs Bauingenieurwesen - Informatik

Rechnen

Zeichen	Beschreibung	Beispiel	Ergebnis
`+`	Plus	`4 + 5`	`9`
`-`	Minus	`12 - 7`	`5`
`*`	Mal	`3 * 6`	`18`
`/`	Geteilt (exakt)	`5 / 2`	`2.5`
`//`	Geteilt (abgerundet)	`5 // 2`	`2`
`%`	Rest	`5 % 2`	`1`
`**`	Hoch	`10 ** 3`	`1000`

Variablen

Eine Variable ist ein Name für ein Speicherort.

```
# lege einen neuen Speicherort mit name "friedhelm" an,
# speichere dort vorerst die Zahl 3.
friedhelm = 3

# überschreibe was an Speicherort "friedhelm" steht
# mit dem Ergebnis der Rechnung (also der Zahl 26)
friedhelm = 4 * 5 + 6

# speichere in "friedhelm" das was
# vorher dort gespeichert war plus 4 (also 30)
friedhelm = friedhelm + 4
```

Typen

Typ	Beschreibung	Beispiel
`int`	ganze Zahl	`3`
`float`	Kommazahl	`3.0`
`bool`	Wahr oder Falsch	`False`
`str`	Zeichenkette	`"Hallo Freunde!"`

Entscheiden

Zeichen	Beschreibung	Beispiel	Ergebnis
`==`	gleicher Wert	`"hallo" == "hallo"`	`True`
`!=`	ungleicher Wert	`"hello" != "Hallo"`	`True`
`<`	kleiner als	`4 < 5`	`True`
`>`	größer als	`4 > 5`	`False`
`<=`	kleiner oder gleich	`7 <= 7`	`True`
`>=`	größer oder gleich	`4 >= 5`	`False`
`not`	Gegenteil	`not True`	`False`
`and`	Und	`True and False`	`False`
`or`	Oder	`True or False`	`True`

Kontrollfluss

Bedingte Ausführung

Grundsätzlich wird ein Programm immer Zeile für Zeile ausgeführt. Will man eine Zeile nur in bestimmten Fällen ausführen, nutzt man die `if`-Bedingung.
Nur, wenn der Ausdruck nach `if` und vor `:` zum Wert `True` ausgewertet wird, werden die weiter eingerückten Anweisungen darunter ausgeführt.

```
if 3 > 4:
    print("Hilfe, die Mathematik ist kaputt!")
    print("ruft die Feuerwehr!")
print("Diesen Text sehen wir immer.")
```

Ausgabe:

```
Diesen Text sehen wir immer.
```

Wenn die Bedingung Wahr ist mache das eine, wenn nicht das andere:

```
if 3 > 4:
    print("Hilfe, die Mathematik ist kaputt!")
    print("ruft die Feuerwehr!")
else:
    print("Puuh, die Mathematik ist noch heil.")
print("Diesen Text sehen wir immer.")
```

Ausgabe:

```
Puuh, die Mathematik ist noch heil.
Diesen Text sehen wir immer.
```

Unterscheide mehrere Einzelfälle:

```
if 3 == 4:
    print("Ganze Zahlen sind kaputt!")
elif 3.0 > 4.0:
    print("Ganze Zahlen sind heil.")
    print("Kommazahlen sind kaputt!")
elif "hallo" == "tschüss":
    print("Ganze Zahlen sind heil.")
    print("Kommazahlen sind heil.")
    print("Zeichenketten sind kaputt!")
else:
    print("Entwarnung")
print("Diesen Text sehen wir immer.")
```

Ausgabe:

```
Entwarnung
Diesen Text sehen wir immer.
```

Wiederholen von Anweisungen

Will man *fast* die selben Anweisungen mehrfach ausführen, nutzt man Schleifen. Die `while`-Schleife funktioniert dabei zu Beginn wie eine `if`-Bedingung: Nur wenn der Logikausdruck zwischen `while` und `:` zum Wert `True` ausgewerted wird, springt das Programm in den weiter eingerückten Bereich und führt diese Zeilen aus.
Im Gegensatz zur `if`-Bedingung wird dieser Prozess nach Ausführung des eingerückten Bereiches wiederholt, bis der Logikausdruck das erste Mal `False` ist.
(Über die `for`-Schleife reden wir später.)

Version 1:

```
x = 2
while x < 6:
    print(x)
    x = x + 1
print("fertig!")
```

Version 2:

```
for x in range(2, 6):
    print(x)
print("fertig!")
```

Ausgabe:

```
2
3
4
5
fertig!
```

Funktionen

Die meisten Funktionen beim Programmieren sind von Außen identisch zu mathematischen Funktionen. Es wird ein (oder mehrere) Argument(e) von der Definitionsmenge auf die Bildmenge abgebildet. Definitionsmenge und Bildmenge sind in einem Programm bestimmte Typen. Als Beispiel wird die selbe Funktion f in der Mathematik geschrieben als

$$f\colon \mathbb{Z} \to \mathbb{Z}, \quad x \mapsto 2 \cdot x + 5$$

und in Python als

```
def f(x: int) -> int:
    return 2 * x + 5
```

Die Menge der ganzen Zahlen $\mathbb{Z}$ entspricht in Python dem Typ `int`.
Eine äquivalente Definition von f in Python wäre

```
def f(x: int) -> int:
    a = 2 * x # der name a existiert nur, solange f berechnet wird
    b = a + 5 # der name b existiert nur, solange f berechnet wird
    return b # der wert von b wird zurückgegeben
```

Die Variablen `a` und `b` existieren nur, bis der Computer an der Zeile `return b` ankommt. Erreicht der Computer eine Zeile die mit `return` beginnt, gilt die Funktion als fertig berechnet und der Ausdruck rechts davon (in diesem Fall `b`) wird ausgewertet und als Funktionswert zurückgegeben. Das Schlüsselwort `def` zeigt dem Computer an, dass an dieser Stelle eine neue Funktion beginnt.
So wie Python den Ausdruck `3 * 2 + 4` zu dem Wert `10` auswertet, kann auch die Funktion `f` angewendet werden. Der Ausdruck `f(7) - 1` etwa wird zu `18` berechnet.
In einer Funktion selbst kann eine beliebige Abfolge von Anweisungen ausgeführt werden:

```
def fakultaet(n: int) -> int:
    ergebnis = 1 # diese Variable existiert nur, solange fakultaet berechnet wird.
    while n > 0:
        ergebnis *= n
        n -= 1
    return ergebnis # beende Funktion und gebe ergebnis zurück
    n = 7 # diese Zeile wird nie ausgeführt
```

Die Funktion `fakultatet` berechnet die Fakultät einer Zahl. In Matheschreibweise ist die Fakultät (geschrieben mit einem Ausrufezeichen) definiert als

$$n! = n \cdot (n-1) \cdot \ \ldots \ \cdot 3 \cdot 2 \cdot 1$$