4. Variablen

Variablen sind ein fundamentales Konzept in Programmiersprachen, da sie es ermöglichen, Daten temporär im Speicher zu speichern und später wieder abzurufen. Ohne Variablen könnten Programme keine Eingaben verarbeiten, Zwischenergebnisse speichern oder auf veränderte Daten reagieren. Sie fungieren als Container für Werte, die sich während der Programmausführung ändern können, was die Grundlage für dynamische und interaktive Programme bildet. Variablen machen Programme flexibel und wiederverwendbar, da derselbe Code mit unterschiedlichen Daten arbeiten kann. Darüber hinaus verbessern sie die Lesbarkeit und Wartbarkeit von Code, indem sie aussagekräftige Namen für Daten bereitstellen.

In Python sind folgende Datentypen für Variablen möglich:

- `int` für Ganzzahlen
- `float` für Fließkommazahlen
- `string` für Zeichenketten
- `boolean` für Wahrheitswerte
- `list` für Listen
- `tuple` für Tupel

Beispiel

```
1 note = 123
2 print(note)
```

```
123
```

Syntax

> **Syntax**
>
> ```
> Bezeichner = Ausdruck
> ```

Bemerkungen

- Python ist eine dynamische Sprache, d.h. Variablen müssen nicht deklariert werden. Der Typ einer Variablen wird durch den Wert bestimmt, der ihr zugewiesen wird.
- Pro Zeile nur eine Anweisung!
- linke Seite: Bezeichner = Variablennamen = Identifier

- rechte Seite: Ausdruck oder ein Wert
- Gleichheitszeichen ist der Zuweisungsoperator
- Variablen sind erst nach einer Anweisung verwendbar.
- Erst wird die rechte Seite ausgewertet und dann an die Variable zugewiesen.

> ⚠ Achtung
>
> Das Gleichheitszeichen ist der Zuweisungsoperator und nicht das mathematische Gleichheitszeichen (Vergleichsoperator).

Regeln für die Wahl des Bezeichners

- Buchstaben, Unterstriche und Ziffern
- erste Zeichen keine Ziffer
- keine Leerzeichen
- keine Python Schlüsselwörter (if, else, return, class, or,…)
- Python ist case-sensitive, d.h. Groß- und Kleinschreibung unterscheiden sich.

Ausdrücke

Ausdrücke können sein: - Operatoren - Literalen - Variablen

Die Auswertung eines Ausdrucks liefert einen Wert oder bricht mit Fehlermeldung ab. Die Auswertung bei arithmetischen Ausdrücken läuft nach folgenden Regeln ab:

1. Klammern zuerst
2. Potenzen
3. Multiplikation / Division
4. Addition / Subtraktion
5. ansonsten von links nach rechts.

Beispiele:

Zuweisung von Werten an Variablen

```
1 note = 4
```

Ausgabe der Variablen

```
1 print(note)
```

```
4
```

Mit Variablen rechnen

```
1 print(2*4**note)
```

```
512
```

Dynamische Typisierung

```
durchschnitt = 12.3
print(durchschnitt)
```

```
12.3
```

Rechnen mit unterschiedlichen Zahldatentypen

```
print(durchschnitt * note)
```

```
49.2
```

Fehler bei nicht definierten Variablen

```
print(goody)
```

```
NameError: name 'goody' is not defined
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[7], line 1
----> 1 print(goody)
NameError: name 'goody' is not defined
```

Strings sind Zeichenketten, die in Anführungszeichen stehen. Sie können mit Variablen verknüpft werden.

```
print("ham")
```

```
ham
```

Analog geht das auch mit einfachen Anführungszeichen

```
print('egg')
```

> **Info**
>
> Der PEP 8 (Python Style Guide) empfiehlt, Strings in einfachen Anführungszeichen zu schreiben.

Der +-Operator wird bei Strings für die Verkettung von Strings (Konkatenation)verwendet.

```
print('ham'+'egg')
```

```
hamegg
```

Stings lassen sich in Python auch mit ganzen Zahlen über den *-Operator verknüpfen, was zu einer Wiederholung des Strings führt.

```
print('ham'*3)
```

```
hamhamham
```

```
print('ham'*1.5)
```

```
TypeError: can't multiply sequence by non-int of type 'float'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[21], line 1
----> 1 print('ham'*1.5)
TypeError: can't multiply sequence by non-int of type 'float'
```

Mit `asserts` wird überprüft, ob der Wert eines Strings einem vorgegebenen Wert entspricht und wird für Tests verwendet. Stimmmt der Wert überein, passiert nichts, ansonsten wird eine Fehlermeldung ausgegeben.

```
assert 0* 'egg' == ""
```

```
assert 0* 'egg' == 'egg'
```

```
AssertionError:
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
Cell In[24], line 1
----> 1 assert 0* 'egg' == 'egg'
AssertionError:
```

Python ist Case-Sensitive.

```
Number = 10
print(Number)
```

```
10
```

```
print(number)
```

```
NameError: name 'number' is not defined
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[26], line 1
----> 1 print(number)
NameError: name 'number' is not defined
```