Python Einführung

In Jupyter Notebooks kann man sowohl Text im markdown-Format (https://www.markdownguide.org/basic-syntax/) als auch Python-Code schreiben. Markdown bietet viele Möglichkeiten zur Textformatierung wie Listen, Tabellen und Bilder. Zur Einführung kommen jetzt ein paar Zeilen Code.

In [1]:

```
# Kommentare werden in Python mit einem 'hash' (#) geschrieben.
# Kommentare werden nicht als Code ausgeführt.
```

Variablen

Alle Daten werden in Variablen gespeichert. Eine Variable ist im Prinzip nichts anderes als ein einfacher Name für beliebig komplexe Daten. Dieser Name darf nicht mit einer Zahl beginnen, kann aber sonst beliebig gewählt werden. Der Name darf auch Zahlen enthalten. Sinnvoll ist ein Name, der den Inhalt der Variablen beschreibt, z.B. `masse_kg` für eine Masse in Kilogramm. In Python ist es üblich, die Variablennamen klein zu schreiben und mehrere Wörter im Name durch einen Unterstrich zu trennen, z.B. `masse_tomaten_kg` . Die Zuweisung von Daten zu einer Variablen geschieht mit einem Gleichheitszeichen.

In [1]:

```
masse_tomaten_kg = 1.25
```

In der Variablen `masse_tomaten_kg` ist jetzt der Wert `1.25` gespeichert. Wenn wir nochmal auf den Wert zugreifen müssen, können wir dies über die Variable machen. Falls sich der Wert ändert, weil wir ein paar Tomaten gegessen haben, können wir der Variable einfach den neuen Wert zuweisen. Dieser neue Wert wird dann überall dort verwendet, wo wir auf die Variable zugreifen.

In [2]:

```
print(masse_tomaten_kg)
masse_tomaten_kg = 0.8
print(masse_tomaten_kg)
```

```
1.25
0.8
```

primitive Datentypen

Daten werden Datentypen zugeordnet. Der Datentyp bestimmt mögliche Werte und Eigenschaften der Daten. Jede Variable hat einen Datentyp. Wenn einer Variable ein neuer Wert zugewiesen wird, kann sich auch der Datentyp ändern.

In Python gibt es drei primitive Datentypen, die als Variablen gespeichert werden können.

- integer numbers (int)
- floating point numbers (float)
- string

Zwei dieser drei Typen sind numerisch, können also zum Rechnen verwendet werden. Der Unterschied ist, dass ints immer ganzzahlig sind und floats Kommata enthalten (kann auch .0 sein). Strings werden durch einfache oder doppelte Anführungszeichen gekennzeichnet.

In [2]:

```
# int
a = 5

# float
b = 3.14
c = 42.0  # hat zwar keine Nachkommestellen, aufgrund des Kommas ist es aber ein float

# string
s = "Hallo Welt"
t = 'Hallo Wilhelmshaven'

text = 'Jonas\' sagt "Hallo"'
```

print()

Mit der `print()` -Funktion kann Text auf der Konsole ausgegeben werden. Als Argument muss einfach der Text übergeben werden. Man kann auch mehrere Argumente übergeben, die dann mit einem Leerzeichen als separator nacheinander ausgegeben werden.

In [5]:

```
# Verketten von Strings
print("Hallo Welt " + t + str(b))

# mit Komma getrennt
print("Hallo Welt", t, b)
```

```
Hallo Welt Hallo Wilhelmshaven3.14
Hallo Welt Hallo Wilhelmshaven 3.14
```

input()

Mit der `input()` -Funktion kann Text vom Benutzer abgefragt und in eine Variable gespeichert werden. Der Rückgabewert von `input()` ist immer ein String, auch wenn nur Zahlen eingegeben werden. Wenn für Rechnungen ein Zahlenwert benötigt wird, kann der String einfach umgewandelt werden. Dies kann aber zu Fehlern führen, wenn etwas anderes als Zahlen eingegeben wird.

In [7]:

```
user_var = input("Bitte gib etwas ein: ")
zahl = float(input("Bitte gib eine Zahl ein: "))

print(user_var, zahl)
```

```
Bitte gib etwas ein: Hallo
Bitte gib eine Zahl ein: 4
Hallo 4
```

Mathematische Operatoren

In Python können numerische Werte nach den Regeln der Mathematik miteinander verrechnet werden. Dafür werden diese Rechenzeichen verwendet:

Operator	Rechenart
+	Addition
-	Subtraktion
*	Multiplikation
/	Division
//	Ganzzahldivision
**	Potenz
( )	Klammern
%	modulo

Bei den meisten Rechenoperationen gilt: Wenn alle verrechneten Werte Integers sind, ist das Ergebnis auch ein Integer. Wenn mindestens einer der verrechneten Werte ein Float ist, dann ist das Ergebnis auch ein Float. Ausnahme: Bei der Division ist das Ergebnis **immer** ein Float, auch wenn Dividend und Divisor beide Integers sind.

Die Ganzzahldivision hat als Ergebnis immer eine ganze Zahl. Wenn die Division eine Kommazahl ergibt, wird diese abgerundet, unabhängig von den Nachkommastellen! Aber auch hier gilt: Wenn der Dividend oder der Divisor (oder beide) Floats sind, ist das Ergebnis auch ein Float; nur wenn beide Integers sind, ist das Ergebnis auch ein Integer.

In [3]:

```
1 14 // 3
```

Out[3]:

4

Der `modulo` -Operator gibt den Rest bei einer Division zurück. Beispiel:

In [105]:

```
1 14 % 3
```

Out[105]:

2

Wenn man 14 durch 3 teilt, erhält man 4 mit einem Rest von 2. Dieser Rest wird ausgegeben. Damit kann auch geprüft werden, ob eine Zahl durch eine andere teilbar ist: der Rest muss dann 0 sein.

Wenn eine numerische Variable mit einer Zahl verrechnet werden soll und das Ergebnis wieder in dieselbe Variable gespeichert werden soll, kann die Rechenoperation mit einem Gleichheitszeichen verknüpft werden:

In [89]:

```
a = 2
a = a + 5  # zu a wird 5 addiert und wieder in a gespeichert
print("a =", a)

b = 2
b += 5  # dies ist identisch mit b = b + 5
print("b =", b)
```

```
a = 7
b = 7
```

Das geht mit jeder Rechenoperation:

In [91]:

```
a = 4
a -=3
print(a)

b = 2
b *= 3
print(b)

c = 12
c /= 3
print(c)

d = 2
d **= 3
print(d)

e = 11
e %= 3
print(e)
```

```
1
6
4.0
8
2
```

In [8]:

```
keine_zahl = float(input())
```

```
zahl
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-8-146260922775> in <module>
----> 1 keine_zahl = float(input())

ValueError: could not convert string to float: 'zahl'
```

Booleans

Boolean (`bool`) ist ein Datentyp, der genau zwei Werte hat: `True` und `False` (manchmal auch: `1` und `0`). Booleans können als Flags (Kennzeichen) verwendet werden, z.B. `has_pencil = True` wäre ein Kennzeichen, dass eine Person einen Stift bei sich trägt. Booleans entstehen aber auch bei Vergleichen. Vergleiche können mit diesen Operatoren durchgeführt werden:

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

Beispiele:

In [3]:

```
1 5 == 5
```

Out[3]:

```
True
```

In [4]:

```
1 5 == 4
```

Out[4]:

```
False
```

In [5]:

```
1 5 != 4
```

Out[5]:

```
True
```

In [6]:

```
1 5 > 4
```

Out[6]:

```
True
```

In [7]:

```
5 <= 4
```

Out[7]:

```
False
```

Alle diese Vergleiche werden zu einem Boolean evaluiert. Booleans können mit `not` verneint werden und mit `and` und `or` verknüpft werden. Mit `and` können zwei Booleans verknüpft werden: das Ergebnis ist nur dann `True`, wenn beide Booleans `True` sind. Mit `or` können zwei Booleans verknüpft werden: das Ergebnis ist dann `True`, wenn mindestens einer der beiden Booleans `True` ist.

Beispiele:

In [8]:

```
not True
```

Out[8]:

```
False
```

In [2]:

```
True and True
```

Out[2]:

```
True
```

In [3]:

```
True and False
```

Out[3]:

```
False
```

In [4]:

```
False and False
```

Out[4]:

```
False
```

In [5]:

```
True or True
```

Out[5]:

```
True
```

In [6]:

```
1 True or False
```

Out[6]:

```
True
```

In [7]:

```
1 False or False
```

Out[7]:

```
False
```

Die drei keywords `not`, `and` und `or` können auch beliebig miteinander verkettet werden. Wie bei mathematsichen Operatoren gibt es auch bei boolschen Operatoren eine Reihenfolge, in der sie abgearbeitet werden. Zuerst `not`, dann `and`, und zum Schluss `or`. Klammern überschreiben diese Reihenfolge und haben immer Priorität. Du kannst ja mal überlegen, was bei den folgenden Ketten heraus kommt.

In [ ]:

```
1 not (True and False) and (not True or not False)
```

In [ ]:

```
1 not not True
```

In [ ]:

```
1 (4 < 5) and (5 != 7)
```

In [ ]:

```
1 ("hello" == "Hello") or (True == True) and (9 >= 4+5)
```

In [ ]:

```
1 False and False or True
```

In [ ]:

```
1 False and (False or True)
```

In [ ]:

```
1 2 + 2 == 4 and not 2 + 2 == 5 and 2 * 2 == 2 + 2
```

In [ ]:

```
1 not False or True
```

In [ ]:

```
1 not (False or True)
```

Verwendung von Strings und Numerischen als Booleans

Strings und Zahlenwerte können auch als Booleans interpretiert werden. Leere Strings werden als `False` interpretiert, alle anderen Strings werden als `True` interpretiert. Bei Zahlenwerten wird `0` als `False` interpretiert, alle anderen Zahlen werden als `True` interpretiert. Mit der Funktion `bool()` kann ein beliebiger Datentyp in einen Boolean umgewandelt werden, damit können wir prüfen, wie ein Wert interpretiert wird:

In [64]:

```
1 bool(0)
```

Out[64]:

```
False
```

In [69]:

```
1 bool(1)
```

Out[69]:

```
True
```

In [70]:

```
1 bool(-4)
```

Out[70]:

```
True
```

In [63]:

```
1 bool(0.0000001)
```

Out[63]:

```
True
```

In [66]:

```
1 bool("")
```

Out[66]:

```
False
```

In [71]:

```
1 bool("Hello World")
```

Out[71]:

```
True
```

Literale

In den Beispielen oben werden die Booleans keiner Variablen zugewiesen. Die allgemeine Bezeichnung dafür ist *Literal*. Ein Literal ist also ein direkter Wert von einem bestimmten Datentyp. Literale können auch bei Zuweisungen verwendet werden. Dafür betrachten wir folgendes Beispiel:

In [48]:

```
1 a = 5
2 b = "Hello World!"
3 c = True
```

In Zeile 1 ist `a` eine Variable und `5` ist ein numerisches Literal.
In Zeile 2 ist `b` eine Variable und `"Hello World!"` ist ein String-Literal.
In Zeile 3 ist `c` eine Variable und `True` ist ein Boolean-Literal.

Literale sind also direkte Werte, wohingegen Variablen Namen sind, unter denen Werte gespeichert sind.

In einer Python-Console werden die Werte von Literalen direkt ausgegeben. In komplexeren Programmen haben Literale keinen Effekt. Dazu aber später mehr.