3 Erste Datentypen

3.1 Zahlen

Python kann verschiedene Arten von Zahlen darstellen und speichern. Wir werden uns hier auf die Zahlen beschränken, welche in der Programmierung oft benutzt werden. Dies sind ganze Zahlen und Fliesskommazahlen. Python kann auch mit Brüchen, komplexen Zahlen oder Dezimalzahlen mit einer festen Anzahl Dezimalstellen rechnen. Diese werden wir aber hier nicht genauer anschauen.

IntegerGanze Zahlen Ganze Zahlen sind sowohl negative als auch positive Zahlen ohne Nachkommastellen. Mit diesen kann Python sehr gut umgehen. Im Gegensatz zu anderen Programmiersprachen sind sie in der Grösse nicht eingeschränkt. Ihr könnt beliebig grosse ganze Zahlen speichern. [2]

FloatFliesskommazahlen Fliesskommazahlen gibt es in den meisten Programmiersprachen. Sie werden benutzt, um Dezimalzahlen zu speichern. Die Arbeit mit Fliesskommazahlen bietet aber auch einiges an Stolpersteinen. Vor allem wenn die Zahlen betragsmässig sehr gross werden oder sehr nahe bei 0 liegen, kann es Probleme mit der Genauigkeit geben - wie wir weiter unten selbst beobachten werden.
Die folgenden Zahlen sind in Python alle zulässig. Sobald ein Punkt, ein e oder ein E vorkommt werden sie als Fliesskommazahl interpretiert:

```
# Ganze Zahlen (Integer)
0
12
1220200399948882888338278
-5
-116920130986472233456294786617302641572474603438 08

# Fliesskommazahlen (Float)
0.0
23.0
0.234
-1e-1
12E4
```

Eine kurze Erklärung braucht der Buchstabe e oder E. Beide Schreibweisen stehen für das gleiche und sind dir vielleicht schon vom Taschenrechner bekannt. Es handelt sich dabei um die wissenschaftliche Schreibweise von Zahlen mit Zehnerpotenzen. So steht zum Beispiel 1.2E4 für $1,2 \cdot 10^4$ oder -1e-1 für $-1 \cdot 10^{-1}$.

[2] Dies ist natürlich nur theoretisch korrekt. Der Arbeitspeicher deines Computers ist beschränkt. Das heisst, wenn dieser voll ist, kann die Zahl nicht mehr grösser werden. Aber um den Arbeitspreicher heutiger Rechner zu füllen, braucht es unvorstellbar grosse Zahlen.

Siehe auch
Brüche:
Das Modul fractions unterstützt auch Bruchrechnen. Falls man präzise und ausschliesslich mit rationalen Zahlen rechnet, können damit die Probleme der Fliesskommazahlen umgangen werden.
Dezimalzahlen:
Das Modul decimal kann zum präziseren - jedoch langsameren - Rechnen mit Dezimalzahlen benutzt werden.

3.1.1 Aufgaben

1. Probiere die unten stehenden Operatoren mit verschiedenen Zahlen aus und notiere dir, was sie tun. Setze für x und y verschiedene Zahlen ein, bis du herausgefunden hast, was die Operatoren tun.

```
>>> x + y
>>> x - y
>>> x % y
>>> -x
>>> x * y
>>> x / y
>>> x**y
>>> x // y
```

2. Was ist der Rückgabewert des unten stehenden Programms? Gehe es zuerst im Kopf durch und probiere es anschliessend aus, in dem du das Programm mit PyCharm in einer neuen Datei abspeicherst.

```
1 x = 2
2 y = 3
3 z = x + y
4 x = z
5 y = x
6 print(x) 5
7 print(y) 5
8 print(z) 5
```

3. Probiere die folgenden Rechnungen in der Python-Konsole aus. Welches Ergebis erwartest du? Woran liegt es, dass du nicht das erwartete Ergebnis erhältst?

warscheinlich weil es in der konsole nur int werte ausgeben kann

```
>>> 0.1 + 0.1 + 0.1 - 0.3
>>> 5 + 10**40 - 10**40
>>> 5.0 + 10**40 - 10**40
```

4. Angenommen, du hast pro Semester vier Prüfungen in einem Fach. Nun sind drei dieser Prüfungen vorbei und du möchtest wissen, welche Note du in der vierten Prüfung haben musst, um deinen Wunschschnitt zu erreichen.

   Schreibe ein Programm, welches dir diese Frage beantwortet. Benutze vier Variablen um die drei Noten und den Wunsch-Durchschnitt abzuspeichern und lasse das Programm daraus die letzte Note berechnen, welche du brauchst, um den Wunsch-Durchschnitt zu erreichen. Diese kannst du mit dem print() Befehl ausgeben.

5. Schreibe ein Programm, welches eine Zeitangabe in Stunden, Minuten und Sekunden in die Anzahl Sekunden insgesamt umrechnet.

6. Eine Werkstatt verlangt für die Benutzung einer Maschine eine Grundgebühr von 60 Euro sowie 35 Euro Stunde. Man überlege sich, welche Daten einzugeben und welche Daten zu berechnen sind und erstelle ein passendes Programm.

7. Ein Kleinunternehmen stellt seinen Kunden für Transporte einen Lieferwagen und einen Lastwagen zur Verfügung. Für Transporte mit dem Lieferwagen werden 1.60 Euro pro Kilometer verrechnet, für den Lastwagen beträgt der Tarif 2.80 Euro pro Kilometer. Welche Gebühr hat ein Kunde zu bezahlen, wenn der Lieferwagen 85 km und der Lastwagen 120 km zurückgelegt hat?

8. Ein Weinhändler verkauft Rotwein zu 18 Euro pro Flasche, Roséwein zu 13 und Weisswein zu 12 Euro pro Flasche. Ein Kunde bestellt (beispielsweise) 12 Flaschen Rotwein, 6 Flaschen Rosé und 24 Flaschen Weisswein. Schreibe ein Programm, welches den Gesamtpreis berechnet.

3.2 Zeichenketten

Beim Programmieren möchte man nicht nur mit Zahlen arbeiten. Man möchte auch Text abspeichern können, um dem Benutzer etwas mitzuteilen oder um den Text zu verarbeiten.
Diesen Datentyp nennt man Zeichenketten oder auf englisch String. Er wird so genannt, weil wir nicht nur Text darin abspeichern können, sondern beliebige Zeichen wie Satzzeichen, Zahlen. Sogar Leerschläge, Tabulatoren und Zeilenumbrüche werden vom Computer als Zeichen behandelt.
Wir haben zwei Möglichkeiten Zeichenketten in Python darzustellen. So kann der String „Hallo Welt“ wie folgt dargestellt und z.B. in einer Variable abgespeichert werden:

```
1 'Hallo Welt'
2 "Hallo Welt"
```

Dies dient dazu, dass auch ein String, welcher ’ oder ënthält, ausgedrückt werden kann:

```
1 'Eine Zeichenkette, welche "Anführungszeichen" enthält'
```

Teilweise haben wir Zeichenketten, welche auch Zahlen darstellen können. So ist zum Beispiel ’2.73’ eine Fliesskommazahl oder ’42’ eine ganze Zahl. Python kann damit jedoch nicht rechnen, da alles zwischen Anführungs- und Schlusszeichen nur als Abfolge von Zeichen interpretiert wird.
In diesem Fall müssen wir die Zeichenketten in Zahlen konvertieren. Dies geschieht mit dem float() Befehl für Fliesskommazahlen respektive mit dem int() Befehl für Integer.

```
>>> int('42') # Dies gibt die Zahl ohne Anführungszeichen zurück
42
>>> float('23.22') # Dies gibt eine Fliesskommazahl zurück
23.22
```

Haben wir hingegen eine Zahl, können wir mit dem Befehl str() daraus eine Zeichenkette machen.

```
>>> str(7)
'7'
>>> str(2.5)
'2.5'
```

Dies funktioniert unabhängig davon, ob es sich um eine ganze Zahl oder eine Fließkommazahl handelt.

3.2.1 Aufgaben

1. Probiere die Operationen mit verschiedenen Strings in der Pythonkonsole aus und notiere dir, was der entsprechende Befehlt tut. Speichere dafür zuerst unter den Variablen string_eins und string_zwei zwei Zeichenketten, zum Beispiel so:

```
>>> string_eins = "Hallo schöne, neue Welt"
>>> string_zwei = "Hallo Mars, ich bin ein Marsroboter"
```

   (a) Probiere nun die folgenden Befehle aus. Setze im folgenden Befehlanstelle der Zahl 3 auch andere Zahlen ein.

```
>>> string_eins[3]
>>> string_eins.capitalize()
>>> string_eins.lower()
```

   (b) Probiere die folgenden zwei Befehle auch mit anderen Buchstaben anstelle von 'e' aus.

```
>>> string_eins.count('e')
>>> string_eins.find('e')
>>> string_eins + string_zwei
```

> Weitere Befehle um mit Strings zu arbeiten findest du hier:
> http://docs.python.org/release/3.1.5/library/stdtypes.html#string-methods

2. Benutzereingaben: Das folgende Programm liest eine Eingabe vom Benutzer ein und speichert dies in der Variable eingabe. Der Befehl input() liest immer Zeichenketten – nicht etwa Zahlen – ein.

```
1 eingabe = input("Gib einen Text ein: ")
2 # Ab hier kann das Programm nun den in der Variable eingabe
3 # gespeicherten Text benutzen ...
```

   Erweitere das Programm so, dass auch die folgenden Schritte ausgeführt werden:

   (a) Der erste Buchstabe der Benutzereingabe wird in Grossbuchstaben konvertiert.

   (b) Dem Text wird ein Ausrufezeichen angehängt.

   (c) Ergebnis wird mit print() auf dem Bildschirm ausgegeben.

3. Erkläre den Unterschied der folgenden Code-Zeilen. Was passiert hier? Kann Python nicht rechnen oder gibt es für dieses Verhalten eine Erklärung?

```
>>> '23' + '7'
'237'
>>> 23 + 7
30
```

4. Schreibe ein Programm, welches vom Benutzer zwei ganze Zahlen einliest und anschließend die Summe der beiden Zahlen ausgibt.

3.3 Listen

Oft reichen Integer, Float und String Datentypen nicht aus, um die notwendigen Daten zu speichern. Meist wissen wir nämlich im Voraus nicht, wie viele Datensätze gespeichert werden sollen. Wenn du dich an die Aufgabe zur Berechnung der Noten zurück erinnerst, sind wir davon ausgegangen, dass du im Semester vier große Prüfungen hast. Dies ist aber natürlich von Fach zu Fach verschieden, und es wäre vorteilhaft, wenn unser Programm eine im Voraus unbekannte Anzahl Noten speichern könnte.
Python bietet für solche Problemstellungen verschiedene Datentypen. Die einfachste Struktur sind die Listen, welche wir in diesem Kapitel genauer anschauen wollen.[3]
In Python können in einer Liste beliebige Datentypen gemischt gespeichert werden - es können sogar Listen in Listen gespeichert werden. Eine Liste beginnt mit einer geöffneten, eckigen Klammer [. Anschliessend werden die Elemente mit Kommas getrennt aufgelistet. Am Schluss wird die Liste wieder mit] geschlossen.
Dies ist ein Beispiel einer Liste:

```
>>> liste = [3, 'King Arthur', ['Rabbit', 3.4], 2.44]
```

Auf die einzelnen Elemente der Liste kann anschließend genau wie bei Zeichenketten über die Nummer des Elements in eckigen Klammern zugegriffen werden. Die Elemente werden auch hier, wie bei den Strings, ab 0 nummeriert.

```
>>> liste[0]
3
>>> liste[1]
'King Arthur'
>>> liste[2]
['Rabbit', 3.4]
```

Falls wir Listen geschachtelt haben - also als Element einer Liste wieder eine Liste gespeichert, können wir mehrere eckige Klammern hintereinander setzen, um auf die einzelnen Elemente zuzugreifen.
Im obigen Beispiel ist an der Stelle 2 eine Liste gespeichert. Um auf die Elemente zuzugreifen, müssen wir in den ersten eckigen Klammern angeben, dass wir die Liste an Stelle 2 meinen und in den zweiten eckigen Klammern geben wir dann an, welches Element in der Unterliste wir herausholen möchten:

```
>>> liste[2][0]
'Rabbit'
>>> liste[2][1]
3.4
```

[3] Weiter gibt es noch Key-Value Speicher, welche in Python „Dictionaries“ heissen. Oder auch Mengen und Tupel im mathematischen Sinn.

Die eckigen Klammern können auch benutzt werden, um einen Wert in einer Liste zu überschreiben. Wir benutzen den Ausdruck, welcher auf ein Element zugreift (z.B. liste[1]) genau wie eine Variable, unter der wir mit = einen Wert speichern:

```
>>> liste
[3, 'King Arthur', ['Rabbit', 3.4], 2.44, 4]
>>> liste[0] = 11  # An der Stelle 0 wird nun 11 gespeichert.
>>> liste
[11, 'King Arthur', ['Rabbit', 3.4], 2.44, 4]
```

Wie bei anderen Datentypen haben Listen in Python viele nützliche Methoden. In den Aufgaben unten findest du wiederum eine Aufgabe, bei der du solche Methoden ausprobieren kannst.

Eine vollständige Auflistung findest du auch hier in der Dokumentation unter
`http://docs.python.org/3/tutorial/datastructures.html#more-on-lists`

3.3.1 Aufgaben

1. Speichere die Elemente 'Schwalbe', 'Kokosnuss', 13, 'Spam' und 3.14 in einer Liste mit dem Namen liste ab und versuche herauszufinden, was die folgenden Methoden machen.

   ```
   >>> liste[2] = 666
   >>> len(liste)
   >>> liste.append('Ni')
   >>> liste.extend([4, 5, 3.14])
   >>> liste.insert(2, 'Taube')
   >>> liste.count(3.14)
   >>> liste.index(3.14)
   >>> liste.remove(3.14)
   >>> liste.pop()
   >>> liste.reverse()
   >>> sum([1,3,5])
   ```

2. Lies das folgende Programm und versuche zu erraten, was die Ausgabe ist. Probiere es anschließend aus und suche nach einer Erklärung des Verhaltens.

   ```
   1 liste_a = ['Hallo', 'schönes', 'Wetter']
   2 liste_b = liste_a
   3 liste_b[1] = 'schlechtes'
   4 print(liste_a[0], liste_a[1], liste_a[2])
   ```

   Das Verhalten dieses Programms ist der Grund, warum wir uns in Python Variablen nicht als Speicherplätze sondern als Namensschilder für Objekte vorstellen. Kannst du dies erklären?

3. In einer früheren Aufgabe hast du ein Programm erstellt, welches eine feste Anzahl Noten einliest und anschliessend die notwendige Berechnung macht. Wir möchten dieses Programm nun anpassen, so dass die Noten in einer Liste gespeichert werden.

   Dies hilft uns dann im nächsten Kapitel, dass wir eine beliebige, vom Benutzer wählbare, Anzahl Noten speichern können.

Abgewandelt von `https://pythonbuch.com/datentypen.html`

Nehmen wir an, dass die Funktion temp_funktion() in einem späteren Programmverlauf nie mehr gebraucht wird. So können wir uns die Namensvergabe sparen und ändern denn Code folgendermassen:

```
def make_list(f,groesse):
    ergebnis = []
    for i in range(groesse):
        ergebnis.append(f(i))
    return ergebnis

a = make_list(lambda x: x + 42, 5)
    print(a)
```

Dieses Programm hat den gleichen Output wie das erste, jedoch mussten wir nicht eine Funktion definieren, welche wir später sowieso nicht gebraucht hätten.

6.4.1 Aufgaben

1. Welche der folgenden Eingaben sind zulässig? Welche nicht und wieso? Wie müsste es richtig sein? Überlege zuerst und tippe es danach zur Kontrolle ein.

```
>>> a = lambda arg1, arg2: arg1 + arg2
>>> print(a(0,2))
>>> a = (lambda arg1, arg2: arg1 + arg2)(0,2)
>>> print(a(0,2))
>>> a = lambda arg1, arg2: arg1 + arg2
>>> print(a(2+2))
>>> a = lambda arg1, arg2: arg1 + arg2
>>> b = lambda x: a(2, x)
>>> print(b(3,4))
```

2. Gegeben ist folgendes Programm:

```
def f2(d):
    return 2**d
def f3(d):
    return 3**d
def f5(d):
    return 5**d
def f7(d):
    return 7**d
def add_function(f,g):
    return f(2) + g(2)

print(add_function(f2, f3))
print(add_function(f5, f7))
```

Schreibe das Programm mit dem lambda-Operator so um, dass die Zeilen 1-12 weggelassen werden können, jedoch der gleiche Output produziert wird.

Abgewandelt von `https://pythonbuch.com/funktion.html`