Python-Cheatsheet

Lukas Pietzschmann
12.01.2022

Abstract

Dieses kurze Cheat-Sheet fasst die wichtigsten Konzepte von Python3 mithilfe prägnanter Beispiele zusammen. Vor allem werden hier allerdings für die Übung relevante Dinge vorgestellt.

Variablen

Im Gegensatz zu vielen anderen Programmiersprachen müssen in Python Variablen nicht explizit deklariert werden. Variablen sind ab ihrer ersten Initialisierung verwendbar.

Typen müssen dabei nie explizit angegeben werden! Folgendes Beispiel zeigt einige relevante Typen:

```
s = "ich bin ein String"
i = 42       #int
f = 3.1415   #float
```

Funktionen

Funktionen sind wie auch Variablen ungetypt. Sie können mit dem Schlüsselwort **def** erstellt werden.

```
def fib(n):
  if n == 1 or n == 0:
    return 0
  else
    return fib(n-1) + fib(n-2)

fib(100) # ⊥
```

Anonyme Funktionen (lambdas)

Wie viele modernen Sprachen unterstützt auch Python Konzepte der funktionalen Programmierung. Lambdas sind eines davon.

Eine (anonyme) Funktion kann dabei mit dem Schlüsselwort **lambda** erzeugt werden.

```
fun = lambda a, b: a + b
fun(1, 2) # 3
```

Da Lambdas ausschließlich Ausrücke und keine Statements enthalten dürfen und dadurch recht eingeschränkt sind, werden sie häufig dazu verwendet simple Funktionen an andere Funktionen als Argument zu übergeben.

```
map(lambda elem: print(elem), [1, 2, 3])
# 1 2 3
```

Listen

Listen sind in Python nicht in ihrer Größe beschränkt und können Daten mit unterschiedlichen Typen enthalten.

```
l = [1, 2]
l.append(3)
print(l)    # [1, 2, 3]
l.pop()
print(l)    # [1, 2]
```

Tupel

Tupel sind recht ähnlich zu Listen, mit dem Unterschied, dass sie nach ihrer Erstellung nicht mehr verändert werden können.

```
t = (1, 2, 3)
```

Operationen auf Listen und Tupeln

Tupel und Listen (und viele anderen Datenstrukturen) teilen sich einige Operationen die auf ihnen zulässig sind. Hier eine kleine Auswahl der wichtigsten:

Element-Zugriff
Mit dem Klammer-Operator `[]` können Elemente anhand ihres Indexes ausgewählt werden. Dabei wird ab 0 indiziert.

```
l = [1, 2, 3]
print(l[0]) # 1
```

Slicing
Soll nicht nur ein Element, sondern gleich mehrere abgefragt werden, kann ebenfalls der Klammer-Operator mit einem Slice verwendet werden. Die Syntax sieht folgendermaßen aus: `[<start>:<ende>:<schrittweite>]`. Dabei ist jeder Parameter Optional

```
l = [1, 2, 3]
print(l[1:]) # [2, 3]
print(l[:1]) # [1]
print(l[::2]) # [1, 3]
```

Iteration
Soll eine Operation für jedes Element einer Liste oder eines Tupels ausgeführt werden, kann eine **for**-Schleife verwendet werden.

```
l = [1, 2, 3]
for i in l:
  print(i) # 1 2 3
```

Imports

Um andere Dateien zu importieren kann das Schlüsselwort **import** verwendet werden.

```
import math
print(math.pi) # 3.14...
```

Sollen nicht alle Symbole der Datei, sondern nur eine ausgewählte Menge importiert werden, kann zusätzlich **from** verwendet werden.

```
from math import pi
print(pi) # 3.14...
```

Diverse Details

Code-Struktur

Python geht den ungewöhnlichen Weg und kennzeichnet Blöcke an Code nicht durch geschweifte Klammern, sondern durch Einrückung. Die Anzahl der Tabs (oder Leerzeichen) ist dabei erstmal egal, sollte aber konsistent bleiben.

Main-Funktion

Python besitzt keine echte Main-Funktion. Code wird einfach von der ersten Zeile ab gelesen und ausgeführt. Folgender Trick kann allerdings eine Main-Funktion halbwegs gut emulieren:

```
def main():
  print("Hallo Welt")

if __name__=="__main__":
  main()
```

Topkonvertierung

In der Übung wird es einige Stellen geben an denen Strings zu Zahlen umgewandelt werden sollen. Hier das Vorgehen:

```
s = "420"
i = int(s)
```

Das Umwandeln von Typen funktioniert in der Regel immer über deren Konstruktoren.

Hilfreiche Funktionen und Methoden

Hier eine kleine Auswahl nützlicher Funktionen und Methoden:

print

Die `print`-Funktion gibt die übergebenen Argumente als Text auf der Standardausgabe aus.

split

Diese Methode kann auf Strings aufgerufen werden und teilt den String an dem übergebenen Trenner.

```
s = "Hallo Welt!"
elems = s.split(" ")
for e in elems:
  print(e) # "Hallo" "Welt"
```

upper, lower

Diese Methode (ebenfalls aus der String-Klasse) geben einen neuen String zurück, der nur aus Groß-, beziehungsweise Kleinbuchstaben besteht.

```
s = "LoL"
print(s.lower()) # "lol"
print(s.upper()) # "LOL"
```

join

`join` ist wieder eine Methode die auf Strings angewendet werden kann. Die bekommt eine Liste an beliebigen Elementen übergeben und konkateniert diese mit dem String als Trenner.

```
print(";".join([1, 2, 3])) # "1;2;3"
```

range

Diese Funktion kann dabei helfen eine „*traditionelle*“ For-Schleife zu emulieren. Sie kann wie folgt verwendet werden:

```
for i in range(1, 3):
  print(i) # 1 2 3
```