Eine Kurzeinführung in Python 3

– DRAFT –

Uwe Ziegenhagen

8. März 2020

Inhaltsverzeichnis

1 Was ist eigentlich Python und warum sollte ich es können? 7

2 Installation 8
2.1 Installation von Python 3.8 unter Windows 8
2.2 Installation von Anaconda unter Windows 8

3 Wie nutze ich Python? 9
3.1 Python im interaktiven Modus 9
3.2 Python im Batch-Modus 10
3.3 Ein- und Ausgabe 11
3.4 Datentypen 12

4 Stringverarbeitung 14
4.1 Von hier aus... 14

5 Funktionen 15

6 Anwendungen 16

Abbildungsverzeichnis

Tabellenverzeichnis

List of Listings

1 11 This is below the code. 6

3.1 Im interaktiven Modus 9
3.2 Überblick der Import-Methoden 10
3.3 Eingabe und Ausgabe 11
3.4 sdfdfd 12
3.5 sdfdfd 12
3.6 sdfdfd 12

aaa

```
1 int('Hello World')
```

```
import this
```

```
# -*- coding: utf-8 -*-

def hello_world():
    print('Hello World')

hello_world()
```

.

```
# -*- coding: utf-8 -*-

def hello_world():
    print('Hello World')

hello_world()
```

Code 1: 11 This is below the code.

Über dieses Dokument

Mit diesem Dokument versuche ich, auf wenigen Seiten einen Überblick über Python 3 zu geben. Ich habe nicht das Ziel, alle Aspekte der Sprache umfassend zu behandeln, ich möchte vielmehr den Einstieg in die Programmierung mit Python ermöglichen und zeigen, was man alles mit Python und ausgewählten Python-Modulen machen kann.

Aus Gründen der Einfachheit werden daher einzelne Aspekte etwas verkürzt dargestellt, die geneigten Leserinnen und Leser mögen es mir nachsehen.

Fehler jedoch bitte ich zu melden, am besten per E-Mail an ziegenhagen@gmail.com

Köln, den 8. März 2020

Dieses Dokument entstand mit LaTeX und der CM Bright Schrift.

1 Was ist eigentlich Python und warum sollte ich es können?

Python ist eine Programmiersprache, die Anfang der 1990er Jahre von Guido van Rossum am Centrum Wiskunde & Informatica in Amsterdam in den Niederlanden entwickelt wurde. Python gilt als sehr gut lesbar und lernbar und eignet sich daher auch gut als erste Programmiersprache.

Python ist dabei eine sogenannte interpretierte Sprache, was heißt, dass man keinen Compiler wie beispielsweise in C und C++ benötigt, um ein Programm laufen zu lassen. Python-Programme sind daher meist etwas langsamer als kompilierte Programme, im praktischen Alltag spielt dies jedoch nur selten eine Rolle.

Lange Zeit gab es zwei parallele Entwicklungsstränge von Python, Python 2 und Python 3. Da Python 2 jetzt Anfang 2020 offiziell den Status „deprecated“, also „überholt“, hat, betrachten wir in diesem Dokument nur Python 3.

2 Installation

- Unter Linux ist Python üblicherweise vorinstalliert, oft ist jedoch noch – zumindest Stand Anfang 2020 – Python 2.7 das Standard-Python, nicht Python 3.
- Unter Windows und Mac OS X ist standardmäßig kein Python installiert, hier muss man also selbst eine Python-Distribution installieren.
- Neben dem Standardpython, das man sich von `https://www.python.org/downloads` herunterladen kann, gibt es noch alternative Distributionen.
- Die bekannteste Distribution ist Anaconda Python, siehe `https://www.anaconda.com/distribution`
- Als Alternative zu Anaconda ist WinPython (`https://winpython.github.io/`) empfehlenswert.

2.1 Installation von Python 3.8 unter Windows

2.2 Installation von Anaconda unter Windows

3 Wie nutze ich Python?

Man kann Python auf verschiedene Arten benutzen:

1. interaktiv wie einen Taschenrechner
2. im Batch-Modus, bei dem die Befehle alle in einer Datei stehen und dann in „einem Rutsch“ (englisch „Batch“) abgearbeitet werden
3. über andere Methoden wie Jupyter und iPython

3.1 Python im interaktiven Modus

In diesem Kapitel betrachten wir kurz die interaktive Nutzung, in den weiteren Kapiteln werden wir dann näher auf die anderen Nutzungsarten eingehen.

Listing 3.1 zeigt ein Beispiel für die Arbeit im interaktiven Modus, in den Python wechselt, wenn man die `python.exe` startet. In diesem Modus kann man Python wie einen Taschenrechner benutzen.

```
1 Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916 32 bit
     (Intel)] on win32
2 Type "help", "copyright", "credits" or "license" for more information.
3 >>> 2+2*3
4 8
5 >>> 2**2
6 4
7 >>> 4**0.5
8 2.0
9 >>>
```

Listing 3.1: Im interaktiven Modus

Zu beachten ist dabei, dass:

1. Python kann beliebig genau mit (ganzen?) Zahlen rechnen
2. Nicht alle Funktionen eines Taschenrechners stehen bereit, ohne dass man ein entsprechendes Modul geladen hat. Dazu gleich mehr.

Mehr Funktionalität im „Taschenrechner“ erhält man, wenn man eines der vielen Module lädt, die Python in seiner Standardbibliothek mitbringt. Nehmen wir beispielsweise die Wurzelfunktion `sqrt()` aus dem `math` Modul. Es gibt verschiedene Methoden, ein Modul zu laden, daher wollen wir diese kurz vorstellen:

`import math` lädt das Modul `math`, zum Aufruf der Funktionen gibt man „math.“ und den Funktionsnamen ein, Beispiel: `math.sqrt(4)`.

`import math as m` stellt die Funktionen von `math` unter der Abkürzung `m` bereit, Beispiel `m.sqrt(4)`. Für viele Module haben sich dabei Standardabkürzungen herausgebildet, so steht `pd` für `pandas`, `np` für `numpy`, etc.

`from math import sqrt` lädt die Funktion in den „allgemeinen Speicher“, ein Präfix ist beim Aufruf nicht mehr nötig. Beispiel `sqrt(4)` Mit dieser Methode sollte man sorgsam umgehen, da verschiedene Module gleiche Funktionsnamen nutzen können. In der Praxis ist daher die zweite Methode empfehlenswert.

3.2 Python im Batch-Modus

Ab jetzt wechseln wir in den Batch-Modus von Python, was nichts anderes heißt als dass wir unseren Python-Code in eine Textdatei schreiben und diese der `python.exe` als Argument übergeben. Python verarbeitet dann die Datei „in einem Rutsch“.

Listing 3.2 zeigt einen Überblick über die verschiedenen Methoden.

```
1 import math
2 print(math.sqrt(4))
3
4 import math as m
5 print(m.sqrt(4))
6
7 from math import sqrt
8 print(sqrt(4))
```

Listing 3.2: Überblick der Import-Methoden

Listing zeigt auch noch einen wichtigen Unterschied zum interaktiven Modus: während man im interaktiven Modus sofort eine Ausgabe der Ergebnisse erhält, muss bzw. kann man im Batch-Modus die Ein- und Ausgabe von Werten, etc. explizit steuern.

```
import math
print(math.sqrt(4))

import math as m
print(m.sqrt(4))
```

```
from math import sqrt
print(sqrt(4))
```

.

3.3 Ein- und Ausgabe

Für die Ausgabe nutzt man in Python den `print()`-Befehl, für die Eingabe `input()`, Listing 3.3 zeigt entsprechende Beispiele, die wir uns noch ein wenig genauer anschauen müssen:

1. Variablen beginnen mit einem Buchstaben oder Unterstrich, Python unterscheidet zwischen Groß- und Kleinschreibung. Die Variable „alter“ ist daher eine andere als die Variable `Alter`.
2. Zeichenketten setzt man in Python mit einfachen oder doppelten Anführungszeichen. Möchte man Anführungszeichen innerhalb einer Zeichenkette ausgeben, so nimmt man jeweils diejenigen, die nicht den String begrenzen.
3. Der `print()` Befehl akzeptiert auch mehrere Argumente, mit Komma getrennt. Standardmäßig werden sie bei der Ausgabe mit einem Leerzeichen getrennt, dies lässt sich durch Angabe des Parameters `sep` ändern.
4. `input()` liest immer nur Zeichenketten (Strings) vom Nutzer ein, selbst wenn man Zahlen eingibt. Möchte man mit der Zahl weiterrechnen, so muss man den String explizit in eine Zahl umwandeln. Dies gilt auch für die verkettete Ausgabe, hier muss die errechnete Zahl auch wieder explizit in einen String umgewandelt werden, bevor man sie ausgeben kann.

```
1  print('Hallo Welt!')
2
3  print('Hallo', 'Welt')
4
5  name = input('Geben Sie Ihren Namen ein: ')
6
7  alter = input('Geben Sie Ihr Alter ein: ')
8
9  print('Hallo, ' + name)
10
11 print('In einem Jahr sind Sie ' + str(int(alter)+1) + ' Jahre alt' )
```

Listing 3.3: Eingabe und Ausgabe

```
1 Hallo Welt!
2 Hallo Welt
```

```
3
4  Geben Sie Ihren Namen ein: Uwe
5
6  Geben Sie Ihr Alter ein: 43
7  Hallo, Uwe
8  In einem Jahr sind Sie 44 Jahre alt
```

3.4 Datentypen

Zwei Datentypen haben wir im Listing 3.3 bereits kennengelernt, `string` und `integer`. String steht für Zeichenketten, integer für ganze Zahlen. Neben diesen beiden gibt es noch zwei weitere primitive Datentypen im Python-Kern: `float` und `boolean`, dazu später mehr. Floats nutzt man für „Zahlen mit Komma“ und `boolean` für logische Wahrheitswerte. Bei Kommazahlen ist zu beachten, dass anstelle des Kommas der Punkt `.` den ganzzahligen Teil vom Dezimalteil abtrennt. Man schreibt also `3.1415927` anstelle von `3,1415927`.

In seinen tausenden von Modulen hält Python noch diverse andere Datentypen bereit, an einigen werden wir im Laufe des Skripts noch vorbeikommen.

Möchte

```
1  # -*- coding: utf-8 -*-
2
3  def hello_world():
4      print('Hello World')
5
6  hello_world()
```

Listing 3.4: sdfdfd 


```
1  # -*- coding: utf-8 -*-
2
3  def hello_world():
4      print('Hello World')
5
6  hello_world()
```

Listing 3.5: sdfdfd

```
1  # -*- coding: utf-8 -*-
2
3  def hello_world():
4      print('Hello World')
5
6  hello_world()
```

Listing 3.6: sdfdfd

4 Stringverarbeitung

In diesem Kapital werden wir uns mit Strings, also mit Zeichenketten, beschäftigen. Python bietet eine Vielzahl von Funktionen dazu an, die wichtigsten davon werden wir uns anschauen.

4.1 Von hier aus...

In diesem Skript können wir nicht auf Reguläre Ausdrücke eingehen. Wer sie nicht kennt: damit lassen sich Herausforderungen lösen wie die Validierung von E-Mail-Adressen oder das kontextsensitive Verändern von Links. Eine gute Webseite, um Reguläre Ausdrücke mit Python zu üben, findet sich unter .

5 Funktionen

In diesem Kapitel geht es darum, eigene Funktionen zu schreiben.

6 Anwendungen