Hochschule Bremerhaven

Fakultät II – Management und Informationssysteme

Informatik

Infrastruktur

Grundlagen Crashkurs in Python

von

Jan-Niklas Pauls
Studiengang: Informatik

02.06.2019

Inhaltsverzeichnis

1 Einleitung

1.1 Was erwartet euch in diesem Guide?

Diese Kurzanleitung wurde von einem Studenten für Studenten geschrieben und dient jediglich zum kurzen Einstieg in Python. Es wird versucht alle wichtigen Grundlagen mit simplen Beispielen verständlich zu erklären. Ich bin selber kein Experte und möchte diesen Guide etwas légerer machen als man es aus professionellen Guides gewohnt ist. In diesem Guide setzte ich allgemeine Grundkenntnisse voraus. Keine Angst, man muss im ersten Semester nicht alles bestanden haben um hier alles zu verstehen.

1.2 Was ist Python?

Python ist eine universelle und höhere Progammiersprache. Ihr Anspruch ist es knapp und gut lesbar zu sein. So verzichtet sie auf geschweifte Klammern um Blöcke zu definieren und strukturiert sich durch das korrekte Einrücken mit dem Tabulator. Zudem unterstüzt Python mehrere Programmierparadigmen. Unter anderem die Objektorientierung, Aspektorientierung oder die funktionelle Programmierung. Python wird auch oft als Skriptsprache verwendet und lässt sich daher auch gut mit Vorkenntnissen in "bash"verstehen. Die Entwickler legten bei der Entwicklung viel Wert auf eine überschaubare und leicht zu erweiternde Standartbibliothek. Auf Grund dieses Konzeptes ist es sehr einfach Python in anderen Sprachen als Modul einzubetten und zu erweitern. So lassen sich Routinen auch in der Programmiersprache 'C' aufrufen. Ebenso lassen sich Plug-Ins für Software wie z.B. Blender, LibreOffice, Maya oder Gimp in Python schreiben.

1.3 Kurzinformationen

- Erscheinungsjahr: 1991
- Entwickler: Python Software Foundation
- Aktuelle Version: 3.7.3 (25. März 2019)
- Typisierung: stark, dynamisch
- Betriebsystem: Plattformübergreifend (großer Vorteil zu Bash)
- Link zur Python-Website: https://www.python.org/

2 Installation

2.1 Windows

1. Öffne den Browser und navigiere zur Download Seite auf "python.org".
2. Wähle unter 'Stable Releases' die 'latest version' aus.
3. Starte nach dem Download den Installer und klicke dich durch die Installation.

2.2 Linux (Ubuntu ab 16.10)

Es ist gut möglich, dass dein System bereits eine Python Version installiert hat. Um dieses zu Überprüfen kannst du folgende Commands im Terminal ausführen: (Dieser Schritt ist nicht notwendig!)

- python – version
- python2 –version
- python3 –version

Als nächstes können wir mit der Installation beginnen. Dafür geben wir folgendes in deinem Terminal ein:

1. sudo apt-get update
2. apt-get install python3.6

2.3 MacOS

1. Installiere 'Homebrew'
2. Öffen dein Terminal
3. Tippe "brew install python3"

Alle Installationvorgänge lassen sich unter diesem Link nochmal genau nachlesen: klick mich.

3 Grundlagen

3.1 Variablen

3.1.1 Normale Variablen

Variablen werden in Python wie folgt initialisiert:

```
x = 5 # int
y = "John" # string
z = ["Oliver", "Hendrik", "Ulrike"] # list of strings
```

Wie hier schon auffällt müssen wir in Python keine Typen angeben um eine Variable zu deklarieren. Ebenso ist es möglich den Typen einer Variable "X"durch Neuzuweisung zu verändern.

```
x = 5 # x is an int
x = "John" # x is now a string
```

Die Bennenung von Variablen ist in Python frei wählbar. So kann man einfach nur "x"nutzen, jedoch bietet es sich an die Bedeutung einer Variable in Ihrem Namen einfliesen zu lassen. Das ist aber jedem selbst überlassen.

```
automarkevonpeter = "Mercedes Benz"
```

Ausgaben können wir in Python ähnlich wie in bash durchführen. Hierzu nutzen wir nicht den bekannten Befehl 'echo', sondern den Befehl 'print'. Wie man einen Wert ausgeben kann, zeige ich hier:

```
x = "15.06.2019"
print(x) # print the value of x
```

Wie aus bash bekannt, kann man auch hier Ausgaben miteinander Verknüpfen.

```
x = "15.06.2019"
print("Der Koggethon wird dieses Jahr am " + x + " stattfinden.")
```

Ebenso sind mathematische Operationen wie die Addition, Subtraktion, Multiplikation und Division möglich. Man kann dieses entweder bei der Erstellung einer Variable oder in einer Ausgabe durch 'print' durchführen.

```
x = 2
y = 5
xmulty = x*y
print("x * y = " + x*y) # would print "x * y = 10"
```

Jedoch würde das Verwenden von mathematischen Operatoren mit einem Zahlenwert und einem String zu einem Error führen. Also sollte soetwas vermieden werden:

```
x = 2
z = "Pauls"
xmulty = x+z # ERROR
print("x + z = " + x+z) # ERROR
```

3.1.2 Globale Variablen

Globale Variablen werden nicht viel anderes verwendet als normale Variablen. Jedoch sind sie sehr wichtig im Umgang mit Funktionen. Sollte man also eine Variable wie oben verändern wollen, so wird diese verändert. Jedoch können Funktionen nun auf Variablen von sich selbt zugreifen. Somit wäre es nicht möglich Werte von außen ohne Parameter aufzurufen und diese zu verändern. Abhilfe schaffen globale Variablen. Diese werden extra in Funktionen importiert.

```
x-outside = 10 # create var "x-outside"
result = None # create var "result" with no vaule

def mult(x): # def function called "mult"
  global result # import var "result" and let the function write on her
  result= x * 10 # mult and write the result in var "result"

print(result) # print "None" <-- is no string (None = Empty)"
mult(x-outside) # run function with parameter "x-outside"
print(result) # print "100"
```

3.2 Funktionen

3.2.1 Wie erstelle ich eine Funktion?

Wie vielleicht ein wenig weiter oben bereits erkannt, haben wir schon 2 Funktionen kennengelernt. So ist unser 'print()' und das selbst definierte 'mult()' eine Funktion gewesen. Zuallererst zeige ich also nochmal wie man eine Funktion selber definiert. Dieses geschieht durch das Keyword 'def', dem Namen und dem Zeichen 'Doppelpunkt'. Der Inhalt wird in Python nur mit einem Tabulator hereingeschoben. Die Funktion endet wenn sie keinen weiteren Tabulator findet.

```
def firstfunc(): # def <name>():
  print("HelloWorld") # inside the function

print("HelloWorld") # outside the function
```

3.2.2 Wie rufe ich eine / meine Funktion auf?

Um eine oder meine Funktion auszurufen schreiben wir einfach nur den Namen der Funktion und fügen "()"hinzu. Sollte die Funktion Parameter erwarten und wir keine übergeben, wird eine Fehlermeldung beim Ausführen erscheinen.

```
def firstfunc():
  print("HelloWorld")

print("HelloWorld") # call print
firstfunc() # call firstfunc
```

3.2.3 Wie übergebe ich Parameter an eine Funktion?

Dieses geschieht indem beim Aufruf der Funktion in die Klammern ein Wert oder eine Variable geschrieben wird. Jedoch muss die Funktion auch einen Parameter erwarten, sonst kommt es zur Fehlermeldung.

```
x = 10

def firstfunc(parameter):
  print("Hello: " + parameter)

firstfunc("1234") # Hello: 1234
firstfunc("abcd") # Hello: abcd
firstfunc(x) # Hello: 10
```

3.2.4 Default-Parameter

Der Default-Parameter wird genutzt um Funktionen auch ausführbar zu machen, wenn kein Parameter übergeben worden ist.
Ein Beispiel hier:

```
def firstfunc(parameter = "Köchin"):  # set default as "Köchin"
  print("Der Pfeffer wurde gestohlen von " + parameter)

firstfunc("Haselmaus") # normal with parameter
firstfunc("Hammerrochen") # normal with parameter
firstfunc() # use default parameter
```

3.2.5 Liste als Parameter

In Python wird sehr viel in Listen bzw. Arrays gearbeitet, deswegen können und möchte man öfters Listen als Parameter an Funktionen übergeben. Dieses funktioniert wie mit einer Variablen.

```
abc = ["a" , "b" , "c"]

def firstfunc(list):
  print(list)

firstfunc(abc)
```

3.2.6 Nutzung von return

Lässt den Wert an die nächste höhere Instanz zurückgeben.

```
def firstfunc(x):
  return x * 10

print(firstfunc(2)) # 20
print(firstfunc(10))  # 100
```

3.2.7 Rekursion

Auch in Python ist die Rekurson möglich. Also aufpassen und Endlosrekursionen vermeiden.

```
def recursion(k):
  if(k>0):
    result = k+recursion(k-1)
    print(result)
  else:
    result = 0
  return result

recursion(6)
```

3.3 Standartelemente

3.3.1 If-Bedingung

Beispiel einer If-Bedingung:

```
if (y>x):
  result = x
  print(result)
elif (y=x):
  result = 0
  print(result)
else:
  result = y
  print(result)
```

3.3.2 while-Schleife

Beispiel einer while-Schleife:

```
n = 1000
s = 0
i = 1
while i <= n:
  s = s + i
  i = i * 1

print("Summe: " + s)
```

3.3.3 for-Schleife

Beispiel einer for-Schleife:

```
fächersem2 = ["SWE" , "Infrastuktur" , "Prog2" , "Mathematik2" ,
              "Rechenarchitektur"]

def firstfunc(list):
  for x in list:
     print(x) # print every single vaule in a single line

firstfunc(fächersem2)
```

3.3.4 Liste von Listen

Man kann in Python auch Listen von Listen erstellen. Dadurch lassen sich in Python sehr gut Abhängigkeiten und Komplexitäten darstellen.

```
fleisch = ["Rind" , "Schwein" , "Fisch"]
gemüse = ["Gruke" , "Tomate" , "Paprika"]
obst = ["Banane" , "Orange" , "Kirsche"]
käse = ["Gouder" , "Edammer" , "Morzarella"]

milchprodukte = [ käse , "Yoguhrt"]

produkte = [fleisch , gemüse , obst , milchprodukte]

print(produkte)
```

3.3.5 Dictionaries als assosziative Arrays

Ähnlich wie ein assoziatives Array in PhP funktioniert hier das 'dictionary'.

```
rechenarchitektur = {
  "dozent": "H. L.",
  "labor": ture,
  "stunden": 2
}
print(rechenarchitektur) # print the full dictionary
print(rechenarchitektur["dozent"]) # print the value of "dozent"

x = rechenarchitektur.get("dozent") # get value of "dozent" from rech.
```

4 Packete

Python ist sehr modular, somit lässt sich immer genau für den jeweiligen Anwendungsfall das/die passende Paket/Bibliothek importieren. Daher ist es an sich sehr schlank und man hat nie mehr dabei als man wirklich benötigt. Viele Pakte sind jedoch für einige Zwecke unerlässlich und sehr mächtige Werkzeuge. Die 3 meiner Meinung nach wichtigsten Bibliotheken im Umgang mit Datenanalyse wirst du unten kennenlernen.

4.1 Wie importiere ich Pakete und mache sie nutzbar?

Es gibt an sich zwei wesentliche Unterschiede. Entweder möchte ich ein ganzen Paket importieren oder nur ein einzelnes Objekt davon. Man kann in Python direkt auf ein importiertes Paket zugreifen, deswegen werden häufig genutze Pakete mit einem äsäbgekürzt und sind nun unter dieser Variablen abrufbar.
Hier findet der Import eines ganzen Paketes statt:

```
import pandas as pd
import numpy as np
```

Hier findet der Import eines einzelnen Objektes eines großen Packages statt:

```
from PIZ import Images, ImageDraw
```

Der Vorteil beim Import von nur einzelen Objekten eines Packages ist, das man viele nicht notwendige Objekte nicht importiert.

4.2 Pandas

Pandas ist eine Bibliothek für Python die Hilfsmittel für die Verwaltung und Analyse von Daten anbietet. Besonders geeignet ist Pandas für den Zugriff auf numerische Tabellen und Zahlenreihen. Ein Beispiel wäre das Auswerten und Analysieren von .csv und .tsv Datein. Häufig wird für einen schnellen Zugriff auf die Daten ein Datafram erstellt.

4.2.1 Wie importiere ich Pandas?

```
import pandas as pd
```

4.2.2 dataframe

Ein Dataframe ist eine 2-Dimensionale Datenstruktur mit Zeilen und Spalten. Operationen können auf Zeilen, sowie Spalten angewendet werden.

4.2.3 Wie erstelle ich ein Dataframe?

Wir können ein Dataframe auf 3 Weise erstellen: Mit einem Dictionary:

```
import pandas as pd
d = {'col1': [1, 2], 'col2': [3, 4]}
df = pd.DataFrame(data=d)
```

Mit Numpy (Python Bibliothek):

```
import pandas as pd
import numpy as np
df2 = pd.DataFrame(np.array(1, 2, 3],
                            4, 5, 6],
                            7, 8, 9]]),
                            columns='a', 'b', 'c'])
```

Durch einen Import einer Datei:

```
  import pandas as pd
  fpath = "/home/jpauls/test.csv"
  df = pd.read_csv(fpath, sep='\t', index=False)
```

4.2.4 Aufrufen eines Dataframes

```
import pandas as pd
fpath = "/home/jpauls/test.csv"
df 0 pd-read\_csv(fpath, sep='\t', index=False)
print(df)
# or
df
```

4.2.5 Existiert eine Spalte mit dem Namen ”x”?

Prüfe ob eine Spalte mit dem Index 'x' existiert:

```
if 'x' in df.columns:
  ...
```

Gegenbeispiel:

```
if 'x' not in df.columns:
  ...
```

4.2.6 Setze eine Spalte

Die 2 steht hier für den Index an der die Spalte eingefügt werden soll. Der 'name' steht für den Namen unter der die Spalte gefunden werden kann. Das "Pauls"definiert einen Default-Wert und das True erlaubt Duplikate.

```
df.insert(2, "name", "Pauls", True)
```

4.2.7 Schreibe das Dataframe zurück in eine Datei

```
fpath = "/home/jpauls/text.csv"
df.to_csv(fpath, sep='\t', index=False)
```

4.2.8 Finde eine Gruppe an Zeilen oder Spalten über den Namen

```
df.loc['viper']
```

4.2.9 Finde alle Zeilen die in der Spalte X den Wert Y haben

```
value = 15.06.2019
df.loc[df['veranstalltungsdaten'] == value]
```

4.3 Numpy

Numpy dient zur einfachen Handhabung von Vektoren und Matrizen und generell mehrdimensionalen Arrays. Zudem bietet es Möglichkeiten zu numerischen Berechungen.

4.3.1 Wie importiere ich Numpy?

```
import numpy as np
```

4.3.2 Array-Erzeugung

```
x = np.array([1, 2, 3])
# or
x = np.arange(10) # create a array from 0 to 9
```

4.3.3 Basis Operationen

```
a = np.array([1, 2, 3, 6])
b = np.linspace(0, 2, 4)
c = a - b
print(c)
==> array([ 1. , 1.33333333, 1.66666667, 4. ])

a**2
print(a)
==> array([ 1, 4, 9, 36])
```

4.3.4 Universelle Funktionen

```
# linespace generate values (from , to, number)
a = np.linspace(-np.pi, np.pi, 100)
b = np.sin(a)
c = np.cos(a)
```

4.4 MatPlotLib

4.4.1 Wie importiere ich matplotlib?

```
import matplotlib.pyplot as plt
```

4.4.2 Abspeichern von Plots in der HS-Infrastruktur

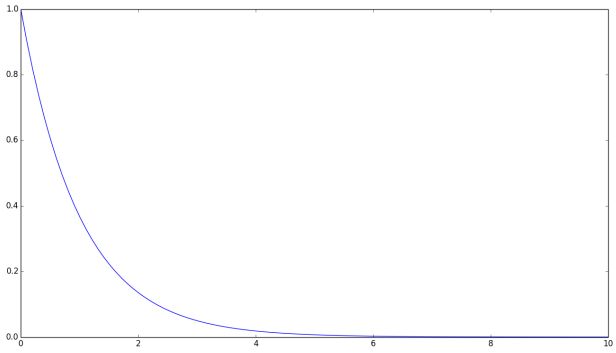
```
import matplotlib as mpl
import matplotlib.pyplot as plt
from numpy.random import rand

mpl.use('Agg') # Agg gives acces to the figure canvas as an RGB string
a = rand(100)
b = rand(100)
plt.scatter(a,b)
plt.savefig("name.png")
```

[Siehe 1, Quelle: O. Radfelder]

4.4.3 Kurven

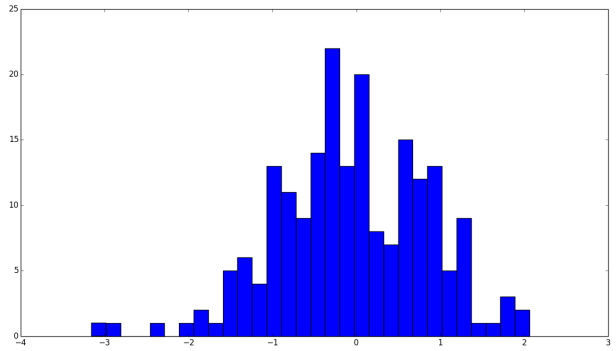
```
import matplotlib.pyplot as plt
import numpy as np
a = np.linspace(0,10,100)
b = np.exp(-a)
plt.plot(a,b)
plt.show()
```



[Siehe 2, Wikipedia Code+Image]

4.4.4 Historgramm

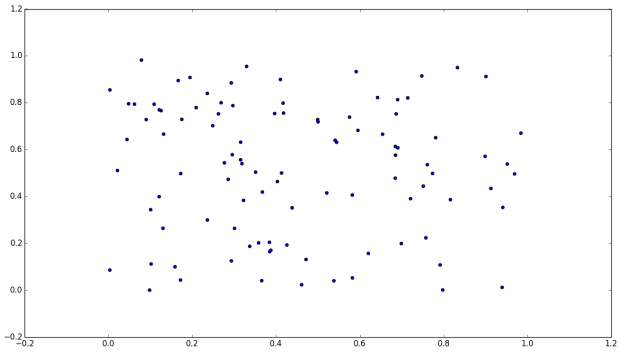
```
import matplotlib.pyplot as plt
from numpy.random import normal,rand
x = normal(size=200)
plt.hist(x,bins=30)
plt.show()
```



[Siehe 2, Wikipedia Code+Image]

4.4.5 Streudiagramm

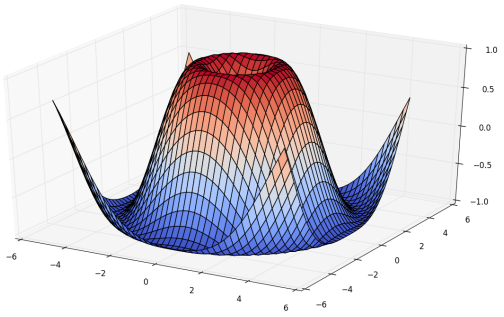
```
import matplotlib.pyplot as plt
from numpy.random import rand
a = rand(100)
b = rand(100)
plt.scatter(a,b)
plt.show()
```



[Siehe 2, Wikipedia Code+Image]

4.4.6 3D Plot

```
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import numpy as np
fig = plt.figure()
ax = fig.gca(projection='3d')
x = np.arange(-5, 5, 0.25)
y = np.arange(-5, 5, 0.25)
x, y = np.meshgrid(x, y)
r = np.sqrt(x**2 + y**2)
z = np.sin(r)
surf = ax.plot_surface(x, y, z, rstride=1, cstride=1, cmap=cm.coolwarm)
plt.show()
```



[Siehe 2, Wikipedia Code+Image]

5 Schlussworte

Dieser Crashkurs ist nur ein kurzer Einblick in Python und drei seiner vielen Bibliotheken welche Python so mächtig machen. Ich hoffe jedoch das du beim Lesen dieses Dokumentes einen guten Einblick erhalten konntest und mit dem Wissen bereits deine ersten Schritte in Python erfolgreich beschreiten kannst. Gerade in der Wissenschaft und Analyse von Daten gewinnt Python immer mehr an Beliebtheit. Also halte dih ran. Bei Rückmeldungen bin ich hier erreichbar: jpauls@studenten.hs-bremerhaven.de

Literatur

[1] *Email von Prof. Radfelder*. Email. Eingesehen am 03.06.2019.

[2] *Wikipedia/Matplotlib/Beispiele*. `https://de.wikipedia.org/wiki/Matplotlib`. Eingesehen am 02.06.2019.