Kurzanleitung zu Python

Alexander Ginschel

1. Pythoncode starten

Macht man, indem man python ins Terminal eingibt und mit import das Skript importiert oder mit python skriptname das Skript gleich ausführen lässt.
Bzw man will einen kurzen Befehl mit python -c ausführen: python -c ’print("hallo_welt")’

1.1. Pip und Python environments

Pip ist das Standardinstallierprogramm für python Module. Man installiert jegliches Modul mit pip install modulname.

Auf Arch Linux kann es da schon mal zu Problemen kommen, dass man nur Module installieren kann, wenn man in einer virtuellen Pythonumgebung ist, die keinen Einfluss auf die Systempythonumgebung hat.
Dazu erstellt man im gewollten Ordner die Umgebung mit python -m venv umgebungsname und aktiviert die Umgebung mit source umgebungsname/bin/activate. Nun sollte man mit pip Module installieren können. Mit exit verlässt man diese Umgebung wieder und muss sie später wieder selbst aktivieren.

2. Variablen deklarieren

```
x = 32
s = "das ist ein String"
boo = True #Das ist ein Boolean, der nicht False ist
abs(-34.5) #gibt Betrag, also 34.5 aus
type(x) #gibt Datentyp von x aus
5 // 3 #gibt ganze Zahlen beim Dividieren aus statt floats
```

Achtung! Alle Variablen außerhalb von Funktionen sind global!

2.1. Variablen konvertieren

```
x = 45
str(x) #konvertiert zu einem String
y = "234"
int(y) #konvertiert zum korespondierenden int-Wert
```

3. Eingabe und Ausgabe

```
inp = input("Gebe einen Wert ein") #Eingabewert wird in inp gespeichert
print(inp) #gibt die Eingabe ins Terminal wieder aus
print(f’{inp} hast du eingegeben’) #mit f-String
x =54
print("x ist ",x) #mit Kommata ist es möglich Variablen auszugeben
```

4. Strings

Mit Strings hat man eine Vielzahl an Methoden aus denen man frei wählen kann.

```
name = "name"
print(len(name))  #gibt Länge von name aus
print(name.find("o")) #gibt Index vom String aus, bei welchem find zum ersten Mal o findet
#da o nicht in name enthalten ist, wird -1 ausgegeben
print(name.capitalize()) #name mit erstem Buchstaben groß
print(name.upper()) #name großgeschrieben
print(name.lower()) #name klein geschrieben
print(name.isdigit()) #prüft, ob name Zahl oder ein Wort ist
print(name.isalpha()) #prüft, ob name ein Wort ist
print(name.count("o")) #zählt, wie oft o in name vorkommt
print(name.replace("a","o")) #ersetzt a mit o in name
print(name*3) #name wird dreimal ausgegeben
print(name[2:3]) #gibt den Substring "me" aus
```

5. If, while, for

5.1. if

```
x = 32
if x > 10:
        print("hi")
elif x < 10:
        print("hello")
else:
        print("x ist 10")
```

5.2. while (do-while gibts btw nicht)

```
while x > 10:
        pass #mach irgendwas
```

6. for-Schleife

```
ls = [0,1,2,3]
for i in range(0,3,1): #i geht von 0 bis 3 im Einerschritt
        print(i)
for item in ls: #foreach loop durch ls
        print(item)
```

7. Funktionen

Void-Funktionen erkennt man daran, dass sie am Ende nichts zurückgeben, wohingegen Variablenfunktionen mit return am Ende des Funktionsdurchganges einen Wert zurückgeben.

```
def foo(c=4): #wird c im Funktionsaufruf nicht angegeben, ist c gleich 4
        c+=1
        return c
foo(5)
```

7.1. Bessere Lesbarkeit für Funktionsdeklarationen

Man darf nicht vergessen, dass dies nur zur besseren Lesbarkeit für den Programmierer da ist und der Pythoninterpreter diese Informationen nicht bei der Programmausführung verwendet.

```
def foo(c : float) -> int: # c ist ein Floatdatentyp und die Ausgabe von foo ist ein Integer
        return int(x) # Das einzige, was der Interpreter wirklich bearbeitet (Integer x wird ausgegeben)
```

8. Ausnahmen

```
try:
	print("Ausgabe")
catch: #wird durchlaufen, wenn try versagt
	print("Ein Fehler ist unterlaufen")
```

9. Tupel, Set, Dictionary, List

9.1. Tupel

```
tupel = ("das","ist","ein","tupel") #ist übrigens schreibgeschützt und unerweiterbar
print(tupel[2]) #gibt "ein" aus
```

9.2. Set (mathematische Menge)

```
set = {2,5,8} #ist schreibgeschützt, erweiterbar,verkleinerbar und sortiert
set.add(6) #fügt 6 ein
set.remove(2) #entfernt 2
```

9.3. Dictionary/Hashmap

```
dictio = dict(a = "4", b = "7") #mit Konstruktor
dictio = {"a" : "4", "b" : "7"} #ohne geht auch
for x in dictio: #foreach geht hier auch
dictio["a"] = "72" #ändert Wert von a
x = dictio["a"] #Zugriff auf a
dictio["c"] = "8" #erstellt neuen Index
for key, item in dictio.items(): #gibt Liste mit Tupel für jeden Eintrag aus
	print("Schlüssel:",key,"|Item:",item) #gibt die Schose wieder aus
dictio.pop("c") #löscht c Eintrag
dictio.popitem() # löscht letzten neuen Eintrag
dictio.keys() # gibt alle Schlüssel/Einträge aus
dictio.clear() #löscht alles in dictio
```

9.4. Liste

```
liste = [1,2,4,5] #erstellt Liste
liste.insert(2, 4) #liste[2] = 4
liste.pop(1) #löscht Liste[1]
liste.pop() #löscht letzten Eintrag der Liste wie liste.pop(-1)
liste.sort() #Liste wird sortiert bzw liste.sort(reverse=True), was rückwärts sortiert
liste[:3] #alle Werte bis zum Index 3
liste[1:] #alle Werte vom Inex 1 (liste[1:3] sind alle Elemente von 1 bis 3)
liste.append(2) #letztes Element ist 2
del liste bzw liste.clear() #löscht die liste
liste[2] # gibt Liste am Index 2 aus
x = list((1,3,5)) #macht aus Tupel eine liste
```

9.4.1. List Comprehension

Syntaktisches Gefüge, das beschreibt, wie vorhandene Lsiten oder andere iterierbare Objekte verarbeitet werden, um so neue Listen zu erstellen.

```
students = [100,90,80,70,60,50,40,30,20,15,10]
passed_students = [i for i in students >= 60] #filtert die Studenten danach, ob sie bestanden haben
```

9.4.2. Numpy-arrray

```
import numpy as np
arr = np.array([[1,2,3,4,5],[6,7,8,9,10]]) #wenn eindimensional nur ein [ ]
arr[1,-1] #gibt letztes Element vom zweiten Element aus
```

10. Lambdafunktionen

x = lambda a : a +10
x(2) #gibt 12 aus

10.1. zip

zips sind eine Zuordnung von zwei oder mehr gleichlangen Containern zueinander.

```
liste1= ["eins", "zwei", "drei"]; liste2 = [1,2,3]
tupellliste = list(zip(liste1,liste2)) #erstellt aus beiden Listen eine Tupelliste
dictliste = dict(zip(liste1,liste2)) #diesmal ists eine Hashmap
```

11. Maps

map() gib ein map Objekt aus, das das Ergebnis eines Datenobjekts und einer Funktion ist. -> map(funktion, liste)

```
liste = list(map(lambda x : x*2, [1,2,3,4])) #erstellt eine verdoppelte Liste
```

12. Kleinere nette Sachen

12.1. rand

import random; print(random.randint(0, 9)) #gibt Integerwert zwischen 0 und 9 aus

12.2. Zeit messen

```
import time
start = time.time() #setzt Timer
end = time.time() #endet timer
intervall = end -start #speichert Zeitintervall
```

12.3. sleep

```
import time
time.sleep(3) #Thread schläft drei Sekunden lang
```

12.4. Mainfunktionen

if __name__ == "__main__":
Alles, was eingerückt in der oben stehenden Mainfunktion steht, wird nur ausgeführt, sofern man dieses Pythonprogramm direkt selbst ausführt und es nicht importiert. Dies ist sofern nützlich, da bei einem import nämlich alles, was in keiner Funktion steht direkt ausgeführt wird, was bei Softwarebibliotheken ziemlich beschissen ist, sofern man Code außerhalb von Funktionen "ungeschützt" hat.

12.5. import

Man nehme an man hätte im Ordner lib das Pythonskript hallo.py würde man direkt außerhalb dieses Ordner das Skript so referenzieren.
import lib.hallo as hallo
Um auf die Funktion hallo_welt in hallo.py zuzugreifen, braucht man nur das zu schreiben:
hallo.hallo_welt()

12.6. Pythonargumente

Terminalargumente

```
import sys
sys.argv[1] #greift auf das erste Argument nach python [skriptname].py zu (sys.argv[0] ist der Skriptname)
```

Schlüsselwortargumente

```
def hello(first,last):
    print("Ich grüße Sie, ",first,last,"!")
hello(last="Schulze", first="Hans")
#obwohl die Namen in der falschen Reihenfolge sind, werden sie in richtiger Reihenfolge zugeordnet
```

Funktionsargumente (der Name ist nicht entscheidend, sondern nur die Anzahl an Kleenesternen)

```
#args also arguments
def main(*args):
    args[0] #gibt erstes Element aus der Argumentliste aus
main(2,5,6,3,6)
#für kwargs (dictionaryargumente bzw keywordarguments)
def main2(**kwargs):
    kwargs["name"] #gibt Eintrag für name aus
main2(name="hans", ort="Neustadt")
```

12.7. requests

```
import requests
page = requests.get("example.com")
print(page.text) #gibt Quelltext der Seite aus
```

12.8. Dekoratoren

Mit Dekoratoren kann man Funktionen als Parameter anderer Funktionen verwenden bzw diese in die Funktion leicht einfügen.

```
def foo(func):
    def hi():
    ...
    func() #hier wird die Funktion, die als Parameter genutzt wird ausgeführt
    ...
@foo #nächste Funktion wird als Parameter von foo verwendet def eine_funktion():
    print("lol")
```

12.9. Systembefehle

```
import os
os.system("ls") #führt Systembefehle aus
```

12.10. assert

assert var > 0 #gibt Fehler aus, wenn var kleiner Null ist (so ziemlich wie bei C)

13. Dateien

Dateien schreiben

```
with open("text", "w") as f: #w = überschreiben/write, a=anfügen/append
    f.write("ein text \n")
bzw f = open("text", "a")
f.write("hi"); f.close() #bei with wird das f.close automatisch gemacht
```

Dateien lesen

```
with open("text") as f: #"r" wird standardmäßig als Parameter genommen
    f.read() #liest sofort alles aus
    f.readline(2) #liest nur die dritte Zeile aus
    zeilen = f.readlines() #speichert alle Zeilen als Liste in zeilen
    for zeile in f: #alternative Möglichkeit zum Durchiterieren aller Zeilen
```

14. Dateisystem

```
import os
#herausfinden, ob Pfad existiert
os.path.exists("/home/nutzer/datei_oder_ordner")
os.mkdir("ordner") #erstellt ordner
os.rmdir("ordner") #löscht ordner
os.makedirs("ordner/unterordner") #erstellt Ordner mit Unterordner
os.replace("text", "ordner/unterordner/text") #verschiebt text in den Unterordner
os.remove("text") #löscht text
import shutil #für betriebssystemunabhängiges Kopieren von Dateien
shutil.copyfile("quelldatei", "kopie") #kopiert Datei
```

15. Verschluesselung

```
import random
import string

chars = " " + string.punctuation + string.digits + string.ascii_letters #mögliche Zeichen zur Verschlüsselung
chars = list(chars)
key = chars.copy()
random.shuffle(key) #zufällige Reihenfolge
#ENCRYPT/Verschlüsselung
plain_text = input("Enter a message to encrypt: ")
cipher_text = ""
for letter in plain_text:
    index = chars.index(letter)
    cipher_text += key[index]
print(f"original message : {plain_text}")
print(f"encrypted message: {cipher_text}")
#DECRYPT/Entschlüsselung
cipher_text = input("Enter a message to encrypt: ")
plain_text = ""
for letter in cipher_text:
    index = key.index(letter)
    plain_text += chars[index]
print(f"encrypted message: {cipher_text}")
print(f"original message : {plain_text}")
```

16. Klassen (btw es gibt keine privaten Daten in Python)

car.py

```
class Car(Maschine): #erbt von fiktiver Klasse Maschine, bei keiner Vererbung braucht man kein ()
        beschwerdeliste = [] #die Liste ist global und aufrufbar von jeder Instanz von Car
        def __init__(self,marke="VW", farbe):   #Konstruktor
                Maschine.__init___(self) #bzw super().__init__(self) (Oberklassenkonstruktor)
                self.marke = marke
                self.farbe = farbe
        def __del__(self): #Destruktor
                print("Schrott)
```

irgendeinskript.py

```
from car import Car
car1 = car("VW","rot")
car2 = car("BMW", "schwarz")
car1.beschwerdeliste.append("zu langsam")
```

16.1. enum-Klassen

```
class Gefahr:
        niedrig,mittel,hoch = range(3)
print(Gefahr.hoch) #gibt 2 aus
```

17. regex (Reguläre Ausdrücke)

```
import re
pattern = re.compile(.*s$') #Pattern, das nach Wörtern sucht mit s am Zeilenende
pattern.search("suchstrings") #findet am Ende einen Patternmatch
liste = pattern.findall("suchstrings") #gibt Liste aller passenden Strings aus
```

reguläre ausdrücke:
. -> egal, was da steht
* -> egal, welche Anzahl von dem, was links neben dem Klene Stern ist
\+ -> mindestens ein Zeichen muss da stehen, von dem was links daneben ist
$ -> markiert Zeilenende
ˆ -> markiert Zeilenanfang
\S -> nicht Leerzeichen, kann mit * rechts kombiniert werden
\s -> Leerzeichen
\? - Zeichen ist optional
[a-z] -> jeder kleine Buchstabe
[A-Z] -> jeder große Buchstabe
[A-Za-z] -> jeder Buchstabe
[0-9] -> jede Ziffer
\. -> ein Punkt

18. Multithreading und Multiprocessing

Multithreading läuft in Python nicht wirklich gleichzeitig, sondern die Threads werden immer mal wieder gewechselt, weswegen man Multithreading für gewöhnlich beim Warten auf Eingabe oder Ausgabe verwendet.

Multiprocessing ist in Python hingegen das Ausnutzen von den mehreren Kernen eines Prozessors, um Rechenoperationen gleichzeitig ausführen zu lassen, was lange Rechenoperationen verschnellern kann, insofern man nicht mehr Prozesse erzeugt, als einem Kerne zur Verfügung stehen.

18.1. Multithreading

Hier wird eine Threadklasse erzeugt, die eine vorher definierte Funktion als Argument annimt. Man kann jedoch auch eine eigene Klasse erstellen, die von threading.Thread erbt, wobei man aber auf den Konstruktor und den Konstruktor der threading.Thread-Klasse nicht versehentlich verzichten sollte! Diese eigene Klasse wird genauso mit start() gestartet und join() mit dem Mainthread vereint wie die normale Thread-Klasse.

```
import threading
def eine_funktion():
        print("hi")
t1 = threading.Thread(target=eine_funktion, args=(1,2,3))
t1.start(); t1.join() #damit, der Mainthread nicht weiterarbeitet, solange t1 nicht fertif ist, wird er gejoint
threading.active_count() #gibt Threadanzahl aus
threading.enumerate() #gibt Threadinformationen aus
```

18.2. Multiprocessing

```
from multiprocessing import Process, cpu_count
import time
def counter(num):
    count = 0
    while count < num:
        count += 1
def main():
    print("cpu count:", cpu_count()) #gibt Anzahl der verfügbaren Kerne aus
    a = Process(target=counter, args=(500000000,))
    b = Process(target=counter, args=(500000000,))
    a.start()
    b.start()
    print("processing...")
    a.join()
    b.join()
    print("Done!")
    print("finished in:", time.perf_counter(), "seconds")
if _name_ == '__main__': #ist wichtig, damit mögliche Kinderprozesse (in diesem Fall nicht) nichts crashen
    main()
```

19. csv's lesen (Comma seperated values)

```
import csv
with open("text.csv") as f: #CSV's lesen
        for row un reader:
                print(row[1]) #gibt zweiten Eintrag der Reihe aus
with open("text.csv", "w") as f: #CSV's schreiben
        writer = csv.writer(f); writer.writerow(["ID", "name"]) #schreibt Dateiheader
        writer.writerow(["1","Ernst"]) #eigentliche Datenreihe
```

20. Graphen zeichnen

```
import matplotlib.pyplot as plt, numpy as np
xwerte=np.array([1,3,5,8]) #x-Koordinaten der Punkte
ywerte=np.array([5,8,2,30]) #y-Koordinaten der Punkte
plt.plot(xwerte,ywerte) #erstellt Graphen
plt.show() #zeigt Graphen
```

21. Web Sockets

```
import socket

#fürn Server
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM) #INET für IP und STREAM für TCP
server.bind((’127.0.0.1’,80)) # braucht ein Tupel; btw 80 ist der http Port
server.listen(1) #hört nur auf einen Client gleichzeitig
while True: #wenn man öfters Daten vom Selben empfangen will, sollte man die Schleife nach accept() packen
        (client,addr) = server.accept() #ins Tupel kommen die Daten des Client
        msg=client.recv(2048) #Bytenachrichtengröße
        msg = msg.decode() #Bytes in String
        print(str(msg,"utf8")) #gibt msg aus

#fürn Client
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM) #INET für IP und STREAM für TCP
client.connect((’127.0.0.1’,80)) # braucht ein Tupel; btw 80 ist der http Port
msg = "lol"; msg = msg.encode() #encodiert "lol", damit man es per Socket schicken kann
client.send(msg)
```

Table of Contents