Cheatsheet

Zusammenfassung der relevantesten Dinge, mit jeweils Syntax und gegebenenfalls einem Beispiel

Inhaltsverzeichnis

1 Terminal

Befehle (in Linuxumgebungen), und was sie tun:

- `cd Pfad/zum/Directory/` - **c**hange **d**irectory, navigiert zu dem angegebenen Pfad.
- `cd ..` und `cd` - cd .. springt in den direkt übergeordneten Ordner, cd in das Ursprungsverzeichnis
- `ls` - Listet alle Ordner und Dateien im aktuellen Verzeichnis auf
- `ls *.py` - Listet alle Dateien im aktuellen Ordner mit einer .py-Endung auf (.py ist durch beliebige Dateienendung ersetzbar).
- `python3 dateiname.py` - Führt das Programm dateiname.py aus. Der Befehl funktioniert nur, wenn man auch im Ordner ist, in dem die Datei liegt.
- `python3 -i dateiname.py` - Startet den **i**nteraktiven Python-Modus. So können Funktionen innerhalb des Programms auf der Konsole aufgerufen werden.
- Automatische Vervollständigung mit der 'Tab-Taste'

Für Windows-PCs findet ihr gleichbedeutende Befehle hier:

`https://www.stationx.net/windows-command-line-cheat-sheet/`

`https://docs.python.org/3/faq/windows.html`

Für MAC-PCs findet ihr gleichbedeutende Befehle hier:

`https://support.apple.com/de-de/guide/terminal/apd5265185d-f365-44cb-8b09-71a064a42125/2.14/mac/14.0`

`https://docs.python.org/3/using/mac.html#`

2 Variablen, Print, Input

Datentypen:

Typ	Beispiel
`int`	`-5, 9, 0, 1233456`
`float`	`-17.0, -3.1415, 29.5, 57.33333`
`string`	`'abcde', 'ABc3f', 'oe# hajd 98.403vcdm!!'`
`bool`	`True, False`

Operatoren:

Natürlich gilt Punkt- vor Strichrechnung und Klammerausdrücke werden bevorzugt ausgewertet.

Operator	Rückgabewert	Beispiel
`+`	Abhängig vom Eingabewert	`3+4=7, 5.6+4=9.6, 'abc'+'a'='abca'`
`-`	Integer oder Float	`7-8=-1, 8.3-7.2=1.1000000`
`*` (Multiplikation)	Abhängig vom Eingabewert	`7*2=14, -7.2*2=-14.4, 3*'he'='hehehe'`
`/` (Division)	Float	`4/2=2.0, 2.1/0.3=7.000`
`//` (ganzzahlige Division)	Integer	`5//2=2, 7.1//4.0=1`
`**` (Potenzieren)	Integer oder Float	`3**3=27, 3.0**3=27.0, 3**3.0=27.0`
`%` (Modulo/'Rest')	Integer	`7%3=1, 4%2=0, 8%3=2`
`>=, >, <=, <`	True oder False	`5.0>6.0=False, 3<=3=True`
`==`	True oder False	`('ha'=='he')=False, (7==7)=True, (7.0==7)=True, ('6'==6)=False`
`!=`	True oder False	`('ha'!='he')=True, (7!=7)=False, (7.0!=7)=False, ('6'!=6)=True`
`and`	True oder False	`(6>8 and 7==7)=False, (6<=8 and 7==7)=True ('E'!='e' and 7.0==7)=True`
`or`	True oder False	`(6>8 or 'l'=='l')=True, (6<=8 or 7==7)=True ('E'=='e' or 7.0!=7)=False`

Print und Input:

Pythoneigene Funktionen mit folgender Syntax und Funktion:

- `print()` gibt das, was in den Klammern steht aus. Darin können auch die obigen Operatoren ausgeführt werden.
- `input()` erwartet eine Eingabe der User*in. Diese wird als *string* gespeichert.

Alles in **Grün** sind Ausgaben auf der Konsole, alles in **Rot** sind Eingaben durch die User*in, und **Orange** kennzeichnet Kommentare, diese gehören nicht zum Code.

```
print("Hallo")                                  >>> Hallo
print(3==7)                                     >>> False
print(7**2)                                     >>> 49
variable1= 5
variable2= "du"
print("Hallo" variable2)                       >>> Hallodu
print("Hallo"  str(variable1))                 >>> Hallo5
print ("Hallo", variable1)                      >>> Hallo 5

eingabe=input("Bitte gib was ein: \n")          >>> Bitte gib was ein:
eingabe2=input()                                >>>
# '\n' innerhalb eines Strings fuehrt zu einem Zeilenumbruch
                                                >>> 5
                                                >>> huhu
print(2*eingabe)                                >>> 55
print(2*int(eingabe))                           >>> 10
print(eingabe2)                                 >>>huhu
```

3 if...elif...else-Bedingungen

Wichtig ist es, die Einrückungen zu beachten. Steht Code ein 'Tab' weiter, als dort, wo das `if/elif/else` beginnt, wird dieser ausgeführt, sobald die geforderte Bedingung wahr ist.

```
if (Bedingung1):
    # Bedingung1 ist wahr, fuehre den Code, der im if steht, aus
    # Tue dies oder das
else:
    # Bedingung1 ist NICHT wahr, fuehre dann den folgenden Code aus
    # Tue jenes
```

——————————Verschachtelungen sind möglich——————————

```
if(Bedingung1):
    # Bedingung 1 ist wahr
    if(Bedingung1_1):
        # Hier landet man, sind Bedingung 1 UND Bedingung1_1 wahr
    else:
        # Bedingung1 wahr, Bedingung1_1 NICHT wahr
```

——————————Mehrere Bedingungen sind möglich——————————

```
if (Bedingung1):
    # Bedingung1 ist wahr, fuehre den Code, der im if steht, aus
elif (Bedingung2):
    # Bedingung2 ist wahr, fuehre den Code, der im elif steht, aus
...
elif (BedingungN):
    # BedingungN ist wahr, fuehre den Code, der im elif steht, aus
else:
    # Keine der vorigen Bedingungen ist wahr
    # fuehre nun den Code hier aus
```

——————————Mehrere Bedingungen sind möglich——————————
—— Aber Achtung! Der untenstehende Code macht nicht dasselbe wie oben——

```
if (Bedingung1):
    # Bedingung1 ist wahr, fuehre den Code, der im if steht, aus
if (Bedingung2):
    # Bedingung2 ist wahr, Bedingung1 koennte wahr oder falsch sein
...
if (BedingungN):
    # BedingungN ist wahr,
    # aber auch alle vorigen Bedingungen koennten wahr sein
else:
    # Nur BedingungN ist nicht wahr
    # ueber die anderen weiss man nichts
```

4 Listen

Im Gegensatz zu einer Variable, lassen sich in Listen beliebig viele Werte (auch unterschiedlichen Typs) speichern.

```
L  = []                                #Leere Liste
L  = [2, 3.5, "9", 'hehe', True]  #Eine Liste mit 5 Elementen
M  = [True, False]
N  = ["a","b"]
M= [True, False,"a", "b"]        #Listen koennen addiert werden
3*N= ["a","b","a","b","a","b"]    #Und ganzzahlig malgenommen werden
```

Listenoperationen:

Operation	**Beispielliste** `L=[1,5,5,5,5,8,6,9,34]`
`L.append(23)`	`L=[1,5,5,5,5,8,6,9,34,23]`
`w=L[0], x=L[5], y=L[-1], z=L[-4]`	`w=1, x=8, y=23, z=6`
`L.reverse() / L[::-1]`	`L=[23,34,9,6,8,5,5,5,5,1]`
`L.pop()`	`L=[23,34,9,6,8,5,5,5,5]`
`L.pop(4)`	`L=[23,34,9,6,5,5,5,5]`
`n=len(L)`	`n=8`
`L.remove(34)`	`L=[23,9,6,5,5,5,5]`
`i=L.index(6)`	`i=2`
`c=L.count(5)`	`c=4`

Es können nicht nur einzelne Elemente in Listen enthalten sein, sondern auch Listen selbst:

Operation	**Beispielliste** `L=[[1,6,1],[3,4],[1,8,0,9],[2]]`
`l=L[2]`	`l=[1,8,0,9]`
`x=L[0][1], y=L[2][:-1], z=L[:-1][0]`	`x=6, y=9, z=2`

Die oben aufgeführten Operationen funktionieren weiterhin analog, mit dem Zusatz, dass sie sowohl auf die ganze Liste anwendbar sind, als auch auf ein einziges Listenelement der Liste.

Operation	**Beispielliste** `L=[[1,6,1],[3,4],[1,8,0,9],[2]]`
`L.pop(), L[0].pop()`	`L=[[1,6,1],[3,4],[1,8,0,9]], L=[[1,6],[3,4],[1,8,0,9]]`
`L.reverse(), L[1].reverse()`	`L=[[1,8,0,9],[3,4],[1,6]], L=[[1,8,0,9],[4,3],[1,6]]`
`n=len(L), m=len(L[2])`	`n=3, m=2`
`L.remove([4,3]),L[1].remove(6)`	`L=[[1,8,0,9],[1,6]], L=[[1,8,0,9],[1]]`
`L.append([1]), L[0].append(1)`	`L=[[1,8,0,9],[1],[1]], L=[[1,8,0,9,1],[1],[1]]`
`i=L.index([1,8,0,9,1]), j=L[0].index(9)`	`i=0, j=3`
`c=L.count([1]), d=L[1].count(1)`	`c=2, d=1`

5 Schleifen

For-Schleife: Eine For-Schleife geht häufig Hand in Hand mit dem `range(start, stop, step)`-Operator. Dieser iteriert ganzzahlig von `start` bis `stop-1`, in `step` Schritten. `step` ist hierbei ein optionales Argument, dessen Default-Wert 1 ist.

```
L=[]
for i in range(0,5):
    L.append(i)
print(L)                                    >>>[0,1,2,3,4]
```

```
L=[]
for i in range(1,7,2):
    L.append(i)
print(L)                                    >>>[1,3,5]
```

Nicht nur über ganze Zahlen kann iteriert werden, sondern auch über Listen:

```
L=["ding", "dang", "dong"]
for element in L:
    print(element)                          >>>ding
                                            >>>dang
                                            >>>dong
```

Eine For-Schleife kann auch benutzt werden, um zum Beispiel eine Liste zu füllen:

```
L=[i for i in range(3,7)]                   >>>L=[3,4,5,6]
L=[i*i for i in range(0,12,3)]              >>>L=[0,9,36,81]
```

While-Schleife:
Eine While-Schleife wird so lange durchlaufen, wie die angegebene Bedingung wahr ist, die grundlegende Syntax ist:

```
while (Bedingung ist wahr):
    # tue dieses oder jenes
```

```
#Ein Beispiel#
k=3
while k<5:
    print(k)
    k+=1
```

```
>>>3
>>>4
```

Häufig werden auch sogenannte Endlos-While-Schleifen benutzt. Bei diesen ist die Bedingung *immer* wahr. Genutzt werden sie zum Beispiel, wenn man auf ein bestimmtes Ereignis wartet. Sobald dieses eintritt, gibt es die Möglichkeit, mit `break` die Endlos-Schleife zu verlassen:

```
# Solange fuer das Jahr nicht 2024 eingegeben wird,
# wird die User*in immer wieder nach dem Jahr gefragt.

while True:
    jahr = int(input("Welches Jahr haben wir?\n"))
    if jahr==2024:
        print("Korrekt, danke!")
        break
```

```
>>> Welches Jahr haben wir?
>>> 2023
>>> Welches Jahr haben wir?
>>> 1924
>>> Welches Jahr haben wir?
>>> 2025
>>> Welches Jahr haben wir?
>>> 2024
>>> Korrekt, danke!
```

6 Funktionen

```
# Funktion ohne Uebergabewert und Rueckgabewert
def print_hello():
    print("Hello/Hallo/Hola")

namen=["a","b"]
for name in namen:
    print_hello()
    print(name)                                    >>> Hello/Hallo/Hola a
                                                   >>> Hello/Hallo/Hola b

# Funktion mit Uebergabewert, ohne Rueckgabewert
def print_hello(uebergebener_name):
    print("Servus/Moin/Gude"+name)

namen=["a","b"]
for name in namen:
    print_hello(name)                              >>> Servus/Moin/Gude a
                                                   >>> Servus/Moin/Gude b

# Funktion mit Uebergabewert und Rueckgabewert
def addiere_3_zahlen(a,b,c):
    return (a+b+c)
def grosses_ergebnis(zahl):
    if zahl>25:
        return True
    else:
        return False

ergebnis=addiere_3_zahlen(9,8,7)
if grosses_ergebnis(ergebnis):
    print(ergebnis, "ist eine gr Zahl")
else:
    print(ergebnis, "ist eine kl Zahl")
                                                   >>> 24 ist eine kl Zahl
```

Eine Funktion darf sich auch selber aufrufen (das nennt sich Rekursion), ein einfaches Beispiel (was sich allerdings auch mit einer Schleife gut realisieren ließe), ist die Fakultät: $n! := n \cdot (n-1) \cdot (n-2) \cdot ... \cdot 3 \cdot 2 \cdot 1$

```
def fakultaet(n):
    if n==1:
        return 1
    else:
        return n*fakultaet(n-1)

print(fakultaet(4))
```

```
>>> 24
```

7 Weiterführende Links etc.

- Eine sehr umfangreiche und gut strukturierte Seite mit vielen Erklärungen und Beispielen : `https://www.python-kurs.eu/python3_interaktiv.php`
- Programmieraufgaben in ganz unterschiedlichen Schwierigkeitsgeraden findet ihr zum Beispiel hier: `https://projecteuler.net/archives` Leider kann man nicht nach Schwierigkeitsgrad filtern.
- Das ist der Editor, den wir hier benutzt haben. Dieser läuft auf MAC, Windows und Linuxrechnern: `https://www.geany.org/`