



Python Fortgeschritten

Zusammenfassung

<https://pitrium.de/>

Python Fortgeschritten

Contents

1) Willkommen	3
2) Kursübersicht.....	3
3) IDE - Visual Studio Code	3
4) Debugging.....	3
5) venv – Virtuelle Umgebung	4
6) Conversions - Datentyp Umwandlung.....	4
7) Scope – Gültigkeitsbereich	5
8) String Fortgeschritten.....	6
9) Integer, Float Fortgeschritten	8
10) None – NoneType	9
11) Listen	10
12) List Comprehensions	14
13) Tuple, Set.....	15
14) Dictionary / Hashtable / Hashmap	16
15) Operationen	18
16) Benutzer Interaktion	19
17) Funktionen Fortgeschritten.....	20
18) Module Fortgeschritten.....	21
19) logging	22
20) os, shutil – Dateien und Verzeichnisse.....	23
21) time, datetime – Zeit und Datum.....	26
22) random – Zufall	28
23) numpy – Array Berechnungen.....	30
24) pandas – Daten Aufbereitung und Analyse.....	34
25) matplotlib – Daten Visualisierung	38
26) jupyter – Live Code Notizbücher	39
27) Exceptions – Kritische Fehler.....	41
28) Typische Fehler.....	43
29) Code Optimierung	44
30) Danke! =)	44

Python Fortgeschritten

- 1) Willkommen
- 2) Kursübersicht

3) IDE - Visual Studio Code

- Python Download:
 - <https://www.python.org/downloads/>
- Visual Studio Code Download:
 - <https://code.visualstudio.com/>
- Alternativen: z.B.: PyCharm, Sublime Text, Visual Studio

4) Debugging

Zeilenweise Code Ausführung zur Fehlersuche und Funktionsprüfung

- Breakpoint – Haltepunkt, an dem die Ausführung deines Programmes pausiert
- Call Stack – Liste aller Funktionsaufrufe bis zum aktuellen Breakpoint
- Locals / Globals – Übersicht über Namen und Werte aller Variablen
- Watch – Beobachtung von beliebigen Variablen
-
- F9 Breakpoint – setzt einen Haltepunkt
- F5 Start Debugging / Continue – Startet oder setzt das Debugging fort
- F10 Step over – setzt Ausführung bis zur nächsten Zeile fort
- F11 Step into – setzt Ausführung bis zur nächsten Zeile fort und führt auch Unterfunktionen Schrittweise aus

Python Fortgeschritten

5) venv – Virtuelle Umgebung

- Eingabeaufforderung öffnen:

```
cmd
```

- zu Zielverzeichnis navigieren:

```
cd /d D:\Prad\zum\Zielverzeichnis
```

- neue virtuelle Umgebung erstellen:

```
py -3 -m venv NAME_DER_UMGEBUNG
```

- virtuelle Umgebung aktivieren:

```
cd VENV_PythonFortgeschritten\Scripts
activate.bat # in Eingabeaufforderung
activate.ps1 # in PowerShell
```

- Module installieren:

```
pip install MODUL_NAME
```

- Textdatei mit Liste von Modulinformationen erstellen:

```
pip freeze > requirements.txt
```

- Liste von Modulen aus Textdatei installieren:

```
pip install -r requirements.txt
```

- virtuelle Umgebung verlassen

```
deactivate
```

6) Conversions - Datentyp Umwandlung

Implizit – automatisch in höheren Datentyp

Explizit – durch Funktionsaufruf mit Gefahr von Informationsverlust

- explizite Konvertierung

```
testString = "12.34"
print(float(testString))
> 12.34

print(int(testString))      # Informationsverlust
> 12
```

- Python Build in Functions zur Konvertierung

- <https://docs.python.org/3/library/functions.html>

Python Fortgeschritten

7) Scope – Gültigkeitsbereich

lokale Variablen – im aktuellen Bereich erstellt

globale Variablen – in übergeordneten Bereich erstellt

- Python verwendet immer möglichst spezifische, also bevorzugt lokale Variablen
- um globale Variablen in Unterfunktionen zu ändern bevorzugt mit „return“ Rückgabewerten arbeiten
- alternativ auch Verwendung des „global“ Schlüsselwortes möglich

Python Fortgeschritten

8) String Fortgeschritten

- Teilstring enthalten Abfrage

```
print("World" in "Hello World =")  
> True
```

- Index von Teilstring

```
print("Hello World =").index("World") # Fehlermeldung wenn nicht gefunden  
> 6
```

```
print("Hello World =").find("Wrld") # -1 wenn nicht gefunden  
> -1
```

- Anzahl von Zeichen in String

```
print(len("Hello World ="))  
> 14
```

- Anzahl von Teilstrings in String

```
print("Hello World =").count("l")  
> 3
```

- Prüfung auf ausschließlich numerischen oder alphanumerischen Inhalt

```
print("Hello World =").isalnum()  
> False
```

```
print("123").isalnum()  
> True
```

```
print("abc").isalpha()  
> True
```

- Kleinschreibung, Großschreibung

```
print("Hello World =").lower()  
> hello world =)
```

```
print("Hello World =").upper()  
> HELLO WORLD =)
```

- String an bestimmten Zeichen trennen

```
print("1, 2, 3, 4, 5".split(", "))  
> ['1', '2', '3', '4', '5']
```

- Teilstring ersetzen

```
print("Hello World =").replace("World", "WRLD")  
> Hello WRLD =)
```

Python Fortgeschritten

- Über einzelne Zeichen in String iterieren

```
for zeichen in "Huhu":  
    print(zeichen)  
  
> H  
> u  
> h  
> u
```

- Slicing – Teile aus String abfragen

```
print("Hello World =")[6]  
> W  
  
print("Hello World =")[0]  
> H  
  
print("Hello World =")[-1]  
> )  
  
print("Hello World =")[12:15] # [Start (einschließlich) : Ende (ausschließlich)]  
> =)
```

Python Fortgeschritten

9) Integer, Float Fortgeschritten

- Absolutwert

```
print(abs(-21.09))  
> 21.09
```

- Runden

```
print(round(2.1))  
print(round(2.9))  
print(round(2.5))           # Abrunden bis einschließlich x.5  
print(round(2.56789, 2))    # Auf Nachkommastelle runden  
> 2  
> 3  
> 2  
> 2.57
```

- Aufrunden / Abrunden

```
import math  
  
print(math.ceil(2.1))  
print(math.floor(2.9))  
> 3  
> 2
```

- Auf beliebige Nachkommastelle aufrunden/abrunden

```
nachkommastelle = 2  
print(math.floor(10**nachkommastelle * 2.999) / 10**nachkommastelle)  
> 2.99
```


Python Fortgeschritten

- Statistische Kennzahlen einer Zahlenreihe

```
import statistics

print(min([1, 2, 2, 3, 4, 3, 2, 1]))    # Minimum
> 1

print(max([1, 2, 2, 3, 4, 3, 2, 1]))    # Maximum
> 4

print(sum([1, 2, 2, 3, 4, 3, 2, 1]))    # Summe
> 18

print(statistics.mean([1, 2, 2, 3, 4, 3, 2, 1])) # Durchschnitt
> 2.25

print(statistics.median([1, 2, 2, 3, 4, 3, 2, 1]))    # Median
> 2

print(statistics.stdev([1, 2, 2, 3, 4, 3, 2, 1]))    # Standardabweichung
> 1.0350983390135313

print(statistics.variance([1, 2, 2, 3, 4, 3, 2, 1]))    # Varianz
> 1.0714285714285714
```

10) None – NoneType

- Objekt in Python auf das verwiesen wird, um anzuzeigen, dass etwas leer oder nicht vorhanden ist

```
print(None)    # Wert
> None

print(type(None))    # Datentyp
> <class 'NoneType'>
```

Python Fortgeschritten

11) Listen

- Sammlung von Elementen
- Liste

```
listeLeer = []  
print(listeLeer)  
> []  
  
print(type(listeLeer))  
> <class 'list'>
```

- Liste mit Werten erstellen (Kurzschreibweise – siehe List Comprehensions)

```
listeVonStrings = ["a", "b", "c"]  
print(listeVonStrings)  
> ['a', 'b', 'c']  
  
listeVonIntegers = [9, 21, 0, 1, 2, 3, 4, 5, 6, 7]  
print(listeVonIntegers)  
> [9, 21, 0, 1, 2, 3, 4, 5, 6, 7]  
  
listeVonFloats = [1.1, 2.2, 3.3, 4.4, 5.5]  
print(listeVonFloats)  
> [1.1, 2.2, 3.3, 4.4, 5.5]  
  
listeVonListen = [[1, 3], [3, 7]] # Auch als Array bekannt  
print(listeVonListen)  
> [[1, 3], [3, 7]]
```

- Anzahl von Elementen in Liste

```
listeVonIntegers = [9, 21, 0, 1, 2, 3, 4, 5, 6, 7]  
print(len(listeVonIntegers))  
> 10
```

- Elemente hinzufügen

```
# Element an Ende der Liste anhängen  
print([9, 21, 0, 1, 2, 3, 4, 5, 6, 7].append(9))  
> [9, 21, 0, 1, 2, 3, 4, 5, 6, 7, 9]  
  
# Element an indizierte Stelle der Liste anhängen liste.append(Index, Element)  
print([9, 21, 0, 1, 2, 3, 4, 5, 6, 7, 9].insert(2, 8))  
> [9, 21, 8, 0, 1, 2, 3, 4, 5, 6, 7, 9]  
  
# Listen verknüpfen  
print([9, 21, 8, 0, 1, 2, 3, 4, 5, 6, 7, 9] + [10, 11, 12, 13])  
> [9, 21, 8, 0, 1, 2, 3, 4, 5, 6, 7, 9, 10, 11, 12, 13]
```

Python Fortgeschritten

- Elemente ändern

```
listeVonIntegers = [9, 21, 8, 0, 1, 2, 3, 4, 5, 6, 7, 9]
listeVonIntegers[0] = 99      # Überschreibt den Wert an der Stelle mit Index
                               0 mit dem Wert 99
print(listeVonIntegers)
> [99, 21, 8, 0, 1, 2, 3, 4, 5, 6, 7, 9]
```

- Slicing - Teile einer Liste abfragen

```
listeVonIntegers = [99, 21, 8, 0, 1, 2, 3, 4, 5, 6, 7, 9]
print(listeVonIntegers[-1])  # letztes Element
> 9

print(listeVonIntegers[2])   # Element an indizierter Stelle (3. Element)
> 8

print(listeVonIntegers[1:3]) # Elemente von Start (einschließlich) bis Ende
                             (ausschließlich)
> [21, 8]

print(listeVonIntegers[:3])  # Elemente von Start bis Ende (ausschließlich)
> [9, 21, 8]

print(listeVonIntegers[2:])   # Elemente von Start (einschließlich) bis Ende
> [8, 0, 1, 2, 3, 4, 5, 6, 7, 9]
```

- Erweitertes Slicing

```
listeVonIntegers = [99, 21, 8, 0, 1, 2, 3, 4, 5, 6, 7, 9]
print(listeVonIntegers[::-1]) # gibt jedes Element einer Liste zurück
> [9, 21, 8, 0, 1, 2, 3, 4, 5, 6, 7, 9]

print(listeVonIntegers[::2])  # gibt jedes zweite Element einer Liste zurück
> [9, 8, 1, 3, 5, 7]

print(listeVonIntegers[::-1]) # gibt jedes Element einer Liste in
                               umgekehrter Reihenfolge zurück
> [9, 7, 6, 5, 4, 3, 2, 1, 0, 8, 21, 9]

print(listeVonIntegers[1:-1:2]) # gibt jedes zweite Element zwischen dem
                                zweiten und letzten (ausschließlich) zurück
> [21, 0, 2, 4, 6]
```

Python Fortgeschritten

- Elemente entfernen

```
listeVonIntegers = [99, 21, 8, 0, 1, 2, 3, 4, 5, 6, 7, 9]
listeVonIntegers.remove(7)          # erstes Element mit Wert 7 entfernen
print(listeVonIntegers)
> [9, 21, 8, 0, 1, 2, 3, 4, 5, 6, 9]

listeVonIntegers.remove(listeVonIntegers[1]) # Element mit Index 1
entfernen
print(listeVonIntegers)
> [9, 8, 0, 1, 2, 3, 4, 5, 6, 9]

print(listeVonIntegers.pop()) # entfernt und gibt das letzte Element zurück
print(listeVonIntegers)
> 9
> [9, 8, 0, 1, 2, 3, 4, 5, 6]
```

- Statistische Kennzahlen

```
print(min(listeVonIntegers))
> 0

print(max(listeVonIntegers))
> 9

print(sum(listeVonIntegers))
> 38
```

- Element in Liste enthalten Abfrage

```
listeVonIntegers = [9, 8, 0, 1, 2, 3, 4, 5, 6]
print(21 in listeVonIntegers)
> False

print(0 in listeVonIntegers)
> True
```

- Position von Element in Liste

```
listeVonIntegers = [9, 8, 0, 1, 2, 3, 4, 5, 6]
print(listeVonIntegers.index(2)) # gibt Index von erstem Element mit Wert 2
zurück
> 4
```

- Anzahl von Elementen mit bestimmtem Wert

```
listeVonIntegers = [9, 8, 0, 1, 2, 3, 4, 5, 6]
print(listeVonIntegers.count(4)) # gibt Anzahl von Elementen mit Wert 99
zurück
> 1
```

Python Fortgeschritten

- Sortieren

```
listeVonIntegers = [9, 8, 0, 1, 2, 3, 4, 5, 6]
listeVonIntegers.sort()      # sortiert Elemente der Liste (aufsteigend)
print(listeVonIntegers)
> [0, 1, 2, 3, 4, 5, 6, 8, 9]
```

- Reihenfolge der Elemente umkehren

```
listeVonIntegers = [9, 8, 0, 1, 2, 3, 4, 5, 6]
listeVonIntegers.reverse()   # kehrt Reihenfolge der Elemente in der Liste um
print(listeVonIntegers)
> [9, 8, 6, 5, 4, 3, 2, 1, 0]
```

- Über einzelne Elemente iterieren

```
for element in [1, 2, 4]:
    print(element)
> 1
> 2
> 4
```

- Liste kopieren

```
listeVonIntegers = [9, 8, 6, 5, 4, 3, 2, 1, 0]

# 1) Referenz auf bestehende Liste erstellen
kopierteListe = listeVonIntegers
kopierteListe.append(99)
print(kopierteListe)
print(listeVonIntegers)
> [9, 8, 6, 5, 4, 3, 2, 1, 0, 99]
> [9, 8, 6, 5, 4, 3, 2, 1, 0, 99]

# 2) Kopie einer bestehenden Liste erstellen
kopierteListe2 = listeVonIntegers[:]
kopierteListe2.append(77)
print(kopierteListe2)
print(listeVonIntegers)
> [9, 8, 6, 5, 4, 3, 2, 1, 0, 99, 77]
> [9, 8, 6, 5, 4, 3, 2, 1, 0, 99]
```

Python Fortgeschritten

12) List Comprehensions

- Kompakte Schreibweise zum Erstellen von Listen

```
manuellErstellteListe = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print(manuellErstellteListe)
> [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

initialeListe = [zahl for zahl in range(10)]
print(initialeListe)
> [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- Berechnungen in List Comprehensions

```
quadriert1 = []
for zahl in initialeListe:
    quadriert1.append(zahl ** 2)
print(quadriert1)
> [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

quadriert2 = [zahl ** 2 for zahl in initialeListe]
print(quadriert2)
> [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

- Abfragen in List Comprehensions

```
ungerade1 = []
for zahl in initialeListe:
    if zahl % 2 != 0:
        ungerade1.append(zahl)
print(ungerade1)
> [1, 3, 5, 7, 9]

ungerade2 = [zahl for zahl in initialeListe if zahl % 2 != 0]
print(ungerade2)
> [1, 3, 5, 7, 9]
```

- Verschachteln

```
kombiniert1 = []
for zahl in initialeListe:
    for buchstabe in "abc":
        kombiniert1.append((zahl, buchstabe))
print(kombiniert1)
> [(0, 'a'), (0, 'b'), (0, 'c'), (1, 'a'), (1, 'b'), ..., (9, 'b'), (9, 'c')]

kombiniert2 = [(zahl, buchstabe) for zahl in initialeListe for buchstabe in "abc"]
print(kombiniert2)
> [(0, 'a'), (0, 'b'), (0, 'c'), (1, 'a'), (1, 'b'), ..., (9, 'b'), (9, 'c')]
```

Python Fortgeschritten

13) Tuple, Set

- Tuple – unveränderbare Liste

```
meinTuple = ()
print(meinTuple)
print(type(meinTuple))
> ()
> <class 'tuple'>
```

- Elemente auslesen und anpassen

```
meinTuple = (1, 2, 3) # Variable wird nur auf ein neues Objekt referenziert
print(meinTuple)
> (1, 2, 3)

print(meinTuple[2]) # Werte lassen sich auslesen
> 3

meinTuple[1] = 99 # Aber Werte können nicht geändert werden
> TypeError: 'tuple' object does not support item assignment
```

- Set – ungeordnete Sammlung einzigartiger Elemente

```
leeresSet = set()
print(leeresSet)
print(type(leeresSet))
> set()
> <class 'set'>
```

- Liste in Set umwandeln

```
meinSet = set([0, 0, 1, 2, 3, 3, 4, 4, 5, 5, 5])
print(meinSet)
> {0, 1, 2, 3, 4, 5}

meinSet = set(['H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd', ' ', '=', ''])
print(meinSet)
> {'H', 'l', 'd', '=', 'e', 'r', ' '), 'W', 'o', ' '}
```

Python Fortgeschritten

14) Dictionary / Hashtable / Hashmap

- Sammlung von Elementen die jeweils aus Schlüsselwort (key) und Wert (value) bestehen

```
dictionaryLeer = {}  
print(dictionaryLeer)  
print(type(dictionaryLeer))  
> {}  
> <class 'dict'>
```

- Dictionary erstellen, keys, values auslesen

```
mitarbeiter = {1: ["Felix", "1988", 90], 2: ["Mareike", "1992", 59]}  
print(mitarbeiter)  
> {1: ['Felix', '1988', 90], 2: ['Mareike', '1992', 59]}  
  
print(mitarbeiter.keys())  
> dict_keys([1, 2])  
  
print(mitarbeiter.values())  
> dict_values([['Felix', '1988', 90], ['Mareike', '1992', 59]])
```

- Elemente hinzufügen

```
mitarbeiter = {1: ["Felix", "1988", 90], 2: ["Mareike", "1992", 59]}  
mitarbeiter[3] = ["Olivia", "1988", 45] # dict[key] = value  
print(mitarbeiter)  
> {1: ['Felix', '1988', 90], 2: ['Mareike', '1992', 59], 3: ['Olivia', '1988', 45]}
```

- Values verändern

```
mitarbeiter = {1: ["Felix", "1988", 90], 2: ["Mareike", "1992", 59], 3: ["Olivia",  
"1988", 45]}  
mitarbeiter[3][2] = 52 # Herzlichen Glückwunsch zur Gehaltserhöhung  
print(mitarbeiter)  
> {1: ['Felix', '1988', 90], 2: ['Mareike', '1992', 59], 3: ['Olivia', '1988', 52]}
```

- Elemente auslesen

```
mitarbeiter = {1: ["Felix", "1988", 90], 2: ["Mareike", "1992", 59], 3: ["Olivia",  
"1988", 45]}  
print(mitarbeiter.get(3))  
> ['Olivia', '1988', 52]  
  
print(mitarbeiter[3]) # Fehlermeldung wenn Schlüsselwort nicht gefunden  
> ['Olivia', '1988', 52]  
  
print(mitarbeiter.get(5)) # None wenn Schlüsselwort nicht gefunden  
> None
```


Python Fortgeschritten

- Elemente entfernen

```
mitarbeiter = {1: ["Felix", "1988", 90], 2: ["Mareike", "1992", 59], 3: ["Olivia",  
"1988", 45]}  
del mitarbeiter[1]  
print(mitarbeiter)  
> {2: ['Mareike', '1992', 59], 3: ['Olivia', '1988', 52]}
```

- Keys enthalten Abfrage

```
mitarbeiter = {2: ["Mareike", "1992", 59], 3: ["Olivia", "1988", 45]}  
print(1 in mitarbeiter)  
> False  
  
print(3 in mitarbeiter)  
> True
```

- Über Keys und Values iterieren

```
mitarbeiter = {2: ["Mareike", "1992", 59], 3: ["Olivia", "1988", 45]}  
for schluessselwort, wert in mitarbeiter.items():  
    print(f"Schlüsselwort: {schluessselwort} - Wert: {wert}")  
> Schlüsselwort: 2 - Wert: ['Mareike', '1992', 59]  
> Schlüsselwort: 3 - Wert: ['Olivia', '1988', 52]
```

Python Fortgeschritten

15) Operationen

- in – Prüfung ob Element in iterierbarer Datenstruktur enthalten ist

```
print("a" in "Hallo =")
> True

print(2 in [1, 2, 3, 4, 5])
> True

print(2 not in [1, 2, 3, 4, 5])
> False

print("=" in ["Hello", "World", "="])
> True

print("e" in ["Hello", "World", "="])
> False

print(2 in {1: ["Felix", "1988", 90], 2: ["Mareike", "1992", 59]})
> True

print(59000 in {1: ["Felix", "1988", 90], 2: ["Mareike", "1992", 59]})
> False

print(2 in 123)          # int ist eine einzige Zahl und kann nicht iteriert werden
> TypeError: argument of type 'int' is not iterable
```

- is – Prüfung ob es sich um das gleiche Objekt handelt
- == Prüfung ob Variablen den gleichen Wert haben

```
# 1) Referenz auf bestehende Liste
listeVonIntegers = [1, 2, 3]
kopierteListe = listeVonIntegers
print(kopierteListe == listeVonIntegers)
print(kopierteListe is listeVonIntegers)
> True
> True

# 2) Kopie einer Liste
kopierteListe2 = listeVonIntegers[:]
print(kopierteListe2 == listeVonIntegers)
print(kopierteListe2 is listeVonIntegers)
> True
> False
```

Python Fortgeschritten

- Kurzschreibweise zum dazu addieren, subtrahieren, multiplizieren, dividieren

```
zahl1 = 21
zahl1 = zahl1 + 3
print(zahl1)
> 24

zahl1 += 3
print(zahl1)
> 27

zahl1 *= 2
print(zahl1)
> 54

i = 0
while i < 3:
    print(i)
    i += 1  # In Python gibt es keinen "++" Operator
> 0
> 1
> 2
```

16) Benutzer Interaktion

- Text an Benutzer ausgeben

```
print("Text an Benutzer ausgeben")
> Text an Benutzer ausgeben
```

- Eingabe von Benutzer einlesen

```
benutzerEingabe = input()
< 12.34

print(benutzerEingabe)
> 12.34

print(type(benutzerEingabe))  # Datentyp immer string - ggf. Konvertierung
> <class 'str'>
```

Python Fortgeschritten

17) Funktionen Fortgeschritten

- Dokumentation mit Docstrings innerhalb von `"""` und `"""`

```
def NeueZufallszahl(kleinsterWert, groessterWert):  
    """DocStrings zur Beschreibung der Funktion."""  
    return random.randrange(kleinsterWert, groessterWert+1)
```

- Standardwerte für Funktionsparameter mit `=`

```
def NeueZufallszahl(kleinsterWert=0, groessterWert=9):  
    return random.randrange(kleinsterWert, groessterWert+1)  
  
print(NeueZufallszahl())  
> 4
```

- Entpacken

```
def FunktionMitMehrerenRueckgabewerten():  
    return "string1", "string2", 3  
  
variable1, variable2, variable3 = FunktionMitMehrerenRueckgabewerten()  
print(variable1, variable2, variable3)  
> string1 string2 3
```

Python Fortgeschritten

18) Module Fortgeschritten

- Eigene Dateien als Modul importieren

```
import NameVonDateiImGleichenVerzeichnis
```

- zuerst wird Python Standard Modul mit entsprechendem Namen gesucht
 - deshalb Dateien nicht wie Standard Module benennen
- anschließend nach Daten mit entsprechendem Namen im Ausführverzeichnis
- zuletzt in Verzeichnissen aus Python PATH

```
import sys
print(sys.path)

### ERGEBNIS / AUSGABE ###
['',
 'C:\\Users\\pk\\AppData\\Local\\Programs\\Python\\Python37\\python37.zip', ...
 'C:\\Users\\pk\\AppData\\Local\\Programs\\Python\\Python37\\lib\\site-packages\\Pythonwin']
```

- `__name__` - spezial Variable

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

- Wenn Datei als Hauptprogramm ausgeführt wird, dann wird der Variable der Wert „`__main__`“ zugewiesen
- Wenn Datei importiert wird, dann wird der Variable der Name der Datei als Wert zugewiesen

Python Fortgeschritten

19) logging

- Einfaches Logging

```
import logging

logging.basicConfig(filename="LogFileName.log", level=logging.INFO,
format="%(asctime)s; %(levelname)s; %(message)s")

logging.debug("Debug logging Nachricht...")
logging.info("Info logging Nachricht...")
logging.warning("Warning logging Nachricht...")
logging.error("Error logging Nachricht...")
logging.critical("Exception logging Nachricht...")
```

- Formatierungs Optionen
 - <https://docs.python.org/3/library/logging.html>
- Logging mit logger Objekt

```
def GetLogger():
    """Neuen Logger erstellen."""
    logger = logging.getLogger("LoggerName")
    logger.setLevel(logging.INFO)
    fileHandler = logging.FileHandler(filename="LogFileName.log")
    formatter = logging.Formatter("%(asctime)s; %(name)s;
%(levelname)s; %(message)s")
    fileHandler.setFormatter(formatter)
    logger.addHandler(fileHandler)

    return logger

# logger Objekt erstellen
logger = GetLogger()

logger.info("Info logging mit logger Objekt ...")
```

Python Fortgeschritten

20) os, shutil – Dateien und Verzeichnisse

- os, shutil Module nutzen

```
import os
import shutil
```

- Datei öffnen, schreiben, schließen

```
# Datei öffnen um zu schreiben
textDatei = open("NeueTextdatei.txt", "w") # alternativ "x" - nur erstellen
textDatei.write("Hallo Welt =)")
textDatei.close()

textDatei = open("NeueTextdatei.txt", "w") # überschreibt existierende Datei
textDatei.write("Hallo Welt =)")
textDatei.close()

textDatei = open("NeueTextdatei.txt", "a") # fügt etwas zu existierender
Datei hinzu
textDatei.write("Noch mal Hallo Welt =)")
textDatei.close()

# Datei auslesen
textDatei = open("NeueTextdatei.txt", "r")
print(textDatei.readlines())
textDatei.close()
> ['Hallo Welt =)Noch mal Hallo Welt =)']
```

- Datei Metadaten auslesen

```
print(os.stat("NeueTextdatei.txt").st_size) # Dateigröße
> 35

print(datetime.fromtimestamp(os.stat("NeueTextdatei.txt").st_mtime)) #
Änderungszeitpunkt
> 2020-10-28 11:55:03.267638
```

- Datei mit Metadaten kopieren

```
shutil.copy2("NeueTextdatei.txt", "NeueTextdatei2.txt")
```

- Datei löschen

```
os.remove("NeueTextdatei.txt")
```

Python Fortgeschritten

- r Strings
 - Slashes / - URIs (z.B.: URLs: <https://pitrium.de/index.php>) und Pfade von Unix Betriebssystemen (z.B.: `/home/peter/Documents/file.txt`)
 - Backslashes \ - Pfade in Windows Betriebssystemen (z.B.: `C:\Users\peter\Documents\file.txt`) und Escape Zeichen (z.B.: `\n` oder `\t`)
 - Windows Pfade in Python

```
winPfad = "C:\\Users\\peter\\Documents\\file.txt"
winPfad = r"C:\Users\peter\Documents\file.txt"
```

- Aktuelles Verzeichnis auslesen

```
print(os.getcwd())
> D:\Projects\Pitrium\Kurse\PythonF
```

- Inhalt von Verzeichnis auflisten

```
print(os.listdir(r"C:\Windows"))
> ['addins', 'appcompat', 'apppatch', 'AppReadiness', 'assembly', ... ]
```

- Verzeichnis erstellen

```
os.mkdir("NeuerOrdner")
os.makedirs(r"NeuerOrdner2\NeuerUnterordner")
```

- Verzeichnis umbenennen

```
os.rename("NeuerOrdner", "UmbenannterOrdner")
```

- Verzeichnis löschen

```
os.rmdir("UmbenannterOrdner") # löscht leere Ordner
shutil.rmtree("NeuerOrdner2") # löscht Ordner und alle Unterelemente
```

- Über alle Dateien in Verzeichnis und Unterordnern iterieren

```
for pfad, unterordner, dateien in os.walk(os.getcwd()):
    for datei in dateien:
        print(datei)
> BenutzerInteraktion.py
> CodeOptimieren.py
...
```

- Größe von Verzeichnis abfragen

```
print(os.path.getsize(os.getcwd()))
> 8192
```

- Pfad aus Verzeichnis und Dateinamen zusammensetzen

```
dateiPfad = os.path.join(r"D:\Pfad\Zu\Datei", "DateiName.txt")
print(dateiPfad)
> D:\Pfad\Zu\Datei\DateiName.txt
```


Python Fortgeschritten

- Verzeichnis und Dateinamen aus Pfad auslesen

```
print(os.path.basename(r"D:\Pfad\Zu\Datei\DateiName.txt"))  
> DateiName.txt  
  
print(os.path.dirname(r"D:\Pfad\Zu\Datei\DateiName.txt"))  
> D:\Pfad\Zu\Datei\
```

- Pfad in Verzeichnis und Dateinamen aufsplitten

```
print(os.path.split(r"D:\Pfad\Zu\Datei\DateiName.txt"))  
> ('D:\\Pfad\\Zu\\Datei', 'DateiName.txt')
```

- Prüfung ob Verzeichnis oder Pfad existiert

```
print(os.path.exists(r"D:\Pfad\Zu\Datei\DateiName.txt"))  
> False
```

Python Fortgeschritten

21) time, datetime – Zeit und Datum

- time und datetime Module nutzen

```
import datetime
import time
```

- Aktuelle Zeit abfragen

```
uhrzeit = time.time() # in Sekunden seit 1970-01-01
print(uhrzeit)
print(type(uhrzeit))
> 1603903842.6046727
> <class 'float'>

zeitstempel = datetime.datetime.now() # als datetime objekt
print(zeitstempel)
print(type(zeitstempel))
> 2020-10-28 17:50:43.219656
> <class 'datetime.datetime'>
```

- Teile von Zeitstempel ausfragen

```
print(zeitstempel.year)
print(zeitstempel.month)
> 2020
> 10
```

- Zeitstempel in String umwandeln

```
print(zeitstempel.strftime("%B %Y"))
> October 2020
```

- Format Optionen

- <https://docs.python.org/3/library/datetime.html>

Parameter	Funktion	Beispiel
%a	Tag - Kurzform	Wed
%A	Tag - Name	Wednesday
%d	Tag	31
%b	Monat - Kurzform	Dec
%B	Monat - Name	December
%m	Monat	12
%Y	Jahr	2018
%H	Stunde 0-23	17
%M	Minute 0-59	41
%S	Sekunde 0-59	8
%f	Millisekunde 000000-999999	548513
%z	UTC Versatz	100
%Z	Zeitzone	CST

Python Fortgeschritten

- Zeitdifferenz ausrechnen

```
print(time.time() - uhrzeit)
> 198.61301374435425

zeitstempel2 = datetime.datetime.now()
zeitdifferenz = zeitstempel2 - zeitstempel
print(zeitdifferenz)
> 0:05:50.236848
```

- Zeitstempel in Sekunden umrechnen

```
sekunden = zeitdifferenz.total_seconds()
print(sekunden)
> 350.236848
```

- Zeitstempel erstellen

```
# Datetime aus Integers
zeitstempel = datetime.datetime(2020, 9, 2)
print(zeitstempel)
> 2020-09-02 00:00:00

# Datetime aus String
zeitstempel = datetime.datetime.strptime("20200202_202202",
"%Y%m%d_%H%M%S")
print(zeitstempel)
> 2020-02-02 20:22:02
```

- Programmausführung pausieren

```
for i in range(10):
    print(time.time())
    time.sleep(1.2)

> 1603905325.6317656
> 1603905326.831873
...
```

Python Fortgeschritten

22) random – Zufall

- random Modul nutzen

```
import random
```

- Zufallszahl zwischen 0 und 1

```
for i in range(3):  
    print(random.random())
```

```
> 0.4792442884459146  
> 0.558269773269404  
> 0.4400202026917327
```

- Zufallszahl aus beliebigem Bereich

```
# Float  
for i in range(3):  
    print(random.uniform(1, 10))    # 1 <= x < 10
```

```
> 3.509527308743771  
> 4.51264463775494  
> 6.660958175092066
```

```
# Integer  
for i in range(3):  
    print(random.randint(1, 10))    # 1 <= x <= 10
```

```
> 6  
> 7  
> 3
```

- Zufallselement aus iterierbarer Datenstruktur

```
for i in range(4):  
    print(random.choice("ABCDEFGHIJKLMNOPQRSTUVWXYZ"))
```

```
> H  
> U  
> H  
> U
```

Python Fortgeschritten

- Mischen von Elementen einer iterierbaren Datenstruktur

```
liste = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
for i in range(3):
    random.shuffle(liste)
    print(liste)

> [9, 5, 1, 8, 6, 2, 3, 4, 0, 7]
> [4, 1, 5, 9, 7, 6, 8, 3, 0, 2]
> [7, 2, 0, 4, 3, 6, 1, 8, 9, 5]
```

- Stichprobe aus iterierbarer Datenstruktur

```
liste = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
for i in range(3):
    print(random.sample(liste, 3))

> [4, 1, 0]
> [2, 0, 4]
> [6, 9, 2]
```

Python Fortgeschritten

23) numpy – Array Berechnungen

- <https://numpy.org/devdocs/user/quickstart.html#functions-and-methods-overview>
- numpy installieren und importieren

```
pip install numpy

import numpy as np
```

- ndarray erstellen

```
numpyArray = np.array([0, 1, 2, 3, 4, 5])
print(numpyArray)
> [0 1 2 3 4 5]

numpyArray2 = np.arange(6)
print(numpyArray2)
> [0 1 2 3 4 5]

print(type(numpyArray))
> <class 'numpy.ndarray'>
```

- Dimension und Form

```
# 0d Array - Scalar
scalar = np.array(21)
print(scalar)
print(scalar.ndim)
print(scalar.shape)
> 21
> 0
> ()

# 1d Array - Liste / Vektor
liste = np.array([0, 1, 2, 3])
print(liste)
print(liste.ndim)
print(liste.shape)
> [0 1 2 3]
> 1
> (4,)
```

Python Fortgeschritten

```
# 2d Array - Matrix / Tabelle
matrix = np.array([ [0, 1],
                   [2, 3],
                   [4, 5]])

print(matrix)
print(matrix.ndim)
print(matrix.shape)
> [[0 1]
   [2 3]
   [4 5]]
> 2
> (3, 2)

# 3d Array - Tensor
tensor = np.array([ [[255, 255, 255], [0, 0, 0]],
                   [[0, 0, 0], [255, 255, 255]]])

print(tensor)
print(tensor.ndim)
print(tensor.shape)
> [[[255 255 255]
   [ 0  0  0]]
   [[ 0  0  0]
   [255 255 255]]]
> 3
> (2, 2, 3)
```

- auf Elemente zugreifen

```
print(liste[1])
print(matrix[1][0])
print(tensor[1][1])
> 1
> 2
> [255 255 255]
```

- Arithmetische Berechnungen

```
print(numpyArray)
print(numpyArray + numpyArray2)
print(numpyArray ** 2)
print(numpyArray < 2)
> [0 1 2 3 4 5]
> [ 0  2  4  6  8 10]
> [ 0  1  4  9 16 25]
> [ True  True False False False False]
```

Python Fortgeschritten

```
matrixA = np.array([ [1, 1],  
                    [0, 1]])  
matrixB = np.array([ [2, 0],  
                    [3, 4]])  
print(matrixA * matrixB)    # Elementweises Produkt  
> [[2 0]  
> [0 4]]
```

- werden für jedes Element des Arrays durchgeführt

- Matrix Produkt

```
print(matrixA.dot(matrixB))  
> [[5 4]  
> [3 4]]
```

- Umformungen

```
print(matrix)  
> [[0 1]  
> [2 3]  
> [4 5]]  
  
flachesArray = matrix.ravel() # .reshape(-1)  
print(flachesArray)  
> [0 1 2 3 4 5]  
  
transponiertesArray = matrix.T  
print(transponiertesArray)  
> [[0 2 4]  
> [1 3 5]]  
  
neuesArray = matrix.reshape(2, 3) # neues Array mit modifizierter Form  
print(neuesArray)  
> [[0 1 2]  
> [3 4 5]]  
  
matrix.resize(2, 3) # passt Form des Arrays selbst an  
> [[0 1 2]  
> [3 4 5]]
```


Python Fortgeschritten

- Arrays zusammensetzen

```
print(np.vstack((matrixA, matrixB))) # row_stacks für 2d Arrays
> [[1 1]
> [0 1]
> [2 0]
> [3 4]]

print(np.hstack((matrixA, matrixB))) # column_stacks für 2d Arrays
> [[1 1 2 0]
> [0 1 3 4]]
```

- Iterieren

```
for i in matrix:
    print(i)

> [0 1 2]
> [3 4 5]

for i in matrix:
    for k in i:
        print(k)

> 0
> 1
> 2
> 3
> 4
> 5
```

- Abfragen

```
print(numpyArray)
> [0 1 2 3 4 5]

print(np.any(numpyArray > 3))
> True

print(np.all(numpyArray > 3))
> False

print(np.where(numpyArray * 2 > 3))
> (array([2, 3, 4, 5], dtype=int64),)
```

Python Fortgeschritten

24) pandas – Daten Aufbereitung und Analyse

- https://pandas.pydata.org/pandas-docs/stable/user_guide/index.html
- installieren und importieren

```
pip install pandas
```

```
import pandas as pd
```

- DataFrame

```
matrix = np.array([[0, 1], [1, 1], [2, 2], [3, 3], [4, 5], [5, 8], [6, 13], [7, 21], [8, 34], [9, 55]])
pythonDaten = pd.DataFrame(matrix)
print(type(pythonDaten))
> <class 'pandas.core.frame.DataFrame'>
```

```
print(pythonDaten.shape)
> (10, 2)
```

```
print(pythonDaten.head())
> 0 1
> 0 0 1
> 1 1 1
> 2 2 2
> 3 3 3
> 4 4 5
```

```
print(pythonDaten.tail())
> 0 1
> 5 5 8
> 6 6 13
> 7 7 21
> 8 8 34
> 9 9 55
```

Python Fortgeschritten

- Daten Laden

```
# Aus .csv Dateien
csvDaten = pd.read_csv(r"Daten\bp-stats-review-2020-all-data.csv", sep=";")
print(type(csvDaten))
> <class 'pandas.core.frame.DataFrame'>

print(csvDaten.shape)
> (10, 33)

print(csvDaten.head())
> Thousand tonnes of Lithium content 1995 1996 1997 ... Unnamed: 32
> 0 Argentina ^ ^ ^ ... NaN
> 1 Australia 2.2 3.9 1.6 ... NaN
> 2 Brazil ^ ^ ^ ... NaN
> 3 Chile 2.0 2.7 4.6 ... NaN
> 4 China 0.3 2.8 2.9 ... NaN
> [5 rows x 33 columns]

print(csvDaten.tail())
> Thousand tonnes of Lithium content 1995 1996 1997 ... Unnamed: 32
> 5 Portugal 0.2 0.2 0.2 ... NaN
> 6 US 3.5 4.0 4.0 ... NaN
> 7 Zimbabwe 0.5 0.5 0.7 ... NaN
> 8 Rest of World 0.7 0.7 1.6 ... NaN
> 9 Total World 9.5 14.8 15.6 ... NaN
> [5 rows x 33 columns]

print(csvDaten.columns)
> Index(['Thousand tonnes of Lithium content', '1995', '1996', '1997', '1998',
        '1999', '2000', ... 'Unnamed: 32'], dtype='object')
```

- Spalten mit NaN löschen

```
nanSpaltenGeloescht = csvDaten.dropna(axis="columns", how="all")
print(nanSpaltenGeloescht.head())
> Thousand tonnes of Lithium content 1995 ... 2019
> 0 Argentina ^ ... 6.4
> 1 Australia 2.2 ... 40.7
> 2 Brazil ^ ... 0.3
> 3 Chile 2.0 ... 16.6
> 4 China 0.3 ... 7.5
> [5 rows x 26 columns]
```

Python Fortgeschritten

- Werte ersetzen

```
werteErsetzt = nanSpaltenGeloeschst.replace("^", 0)
print(werteErsetzt.head())

### ERGEBNIS / AUSGABE ###
> Thousand tonnes of Lithium content 1995 ... 2019
> 0          Argentina  0 ... 6.4
> 1          Australia 2.2 ... 40.7
> 2           Brazil   0 ... 0.3
> 3           Chile   2.0 ... 16.6
> 4           China   0.3 ... 7.5
> [5 rows x 26 columns]
```

- Slicing

```
laender = nanSpaltenGeloeschst["Thousand tonnes of Lithium content"]
print(type(laender))
> <class 'pandas.core.series.Series'>

print(laender.head())
> 0  Argentina
> 1  Australia
> 2   Brazil
> 3   Chile
> 4   China
> Name: Thousand tonnes of Lithium content, dtype: object

laender2019 = nanSpaltenGeloeschst[["Thousand tonnes of Lithium content",
"2019"]]
print(type(laender2019))
> <class 'pandas.core.frame.DataFrame'>

print(laender2019.head())
> Thousand tonnes of Lithium content 2019
> 0          Argentina  6.4
> 1          Australia 40.7
> 2           Brazil   0.3
> 3           Chile  16.6
> 4           China   7.5
```

Python Fortgeschritten

- Daten filtern

```
mehrAls1kt = laender2019[laender2019["2019"] > 1]
print(type(mehrAls1kt))
> <class 'pandas.core.frame.DataFrame'>

print(mehrAls1kt.head())
> Thousand tonnes of Lithium content 2019
> 0          Argentina  6.4
> 1          Australia 40.7
> 3           Chile 16.6
> 4           China  7.5
> 5          Portugal  1.2
```

- Daten sortieren

```
sortiert = mehrAls1kt.sort_values(by=["2019"], ascending=False)
print(sortiert.head())
> Thousand tonnes of Lithium content 2019
> 9          Total World 77.0
> 1          Australia 40.7
> 3           Chile 16.6
> 4           China  7.5
> 0          Argentina  6.4
```

- DataFrame speichern

```
sortiert.to_excel(r"Daten\LithiumFoerderung2019.xlsx")
# evtl. pip install openpyxl
sortiert.to_csv(r"Daten\LithiumFoerderung2019.csv")
```

Python Fortgeschritten

25) matplotlib – Daten Visualisierung

- <https://matplotlib.org/users/index.html>
- installieren und importieren

```
pip install matplotlib

from matplotlib import pyplot as plt
```

- Linien Diagramm

```
# Datenreihen definieren
xDaten = [zahl for zahl in range(-3, 8)]
yDaten = [zahl ** 2 for zahl in range(-3, 8)]
yDaten2 = [zahl * 5 for zahl in range(-3, 8)]

# Datenreihen zu Diagramm hinzufügen
plt.plot(xDaten, yDaten)
plt.plot(xDaten, yDaten2)

# Diagramm Beschriftungen
plt.title("Diagramm Titel")
plt.xlabel("x Achse")
plt.ylabel("y Achse")

# Diagramm anzeigen
plt.show()
```

- Torten Diagramm

```
# Daten laden
csvDaten = pd.read_csv(r"Daten\LithiumFoerderung2019.csv", sep=",")
datenOhneSumme = csvDaten[csvDaten["Thousand tonnes of Lithium content"] != "Total World"]
laender = datenOhneSumme["Thousand tonnes of Lithium content"]
foerderung = datenOhneSumme["2019"]

# Datenreihen zu Diagramm hinzufügen
plt.pie(foerderung, labels = laender)

# Diagramm anzeigen
plt.show()
```

Python Fortgeschritten

26) jupyter – Live Code Notizbücher

- <https://jupyter.org/documentation>
- installieren

```
pip install jupyter
```

- jupyter starten

```
jupyter notebook
```

- notebook öffnen
 - <http://127.0.0.1:8888>
- Markdown

```
# Überschriften
```

```
normaler Text
```

```
*italic Text*
```

```
**fetter Text**
```

```
1. erster Stichpunkt
```

```
1. zweiter Stichpunkt
```

```
* Stichpunkt
```

```
* Stichpunkt2
```

```
|erste Spalte|zweite Spalte|dritteSpalte|  
|-----|:-----:|-----:|  
|unzentriert|zentriert|rechtsbündig|  
|Hello|World|=)|
```

```
Horizontale Linie:
```

```
***
```

```
Verlinkter Text:
```

```
[Link zu pitrium.de](https://www.pitrium.de)
```

```
Eingebettetes Bild:
```

```
![Bild](https://pitrium.de/wp-content/uploads/2020/05/250x307Logo_blue_transparentBG.png)
```

```
Leerzeile:
```

```
Zeile vor Leerzeilen. <br> <br> <br>
```

```
Zeile nach Leerzeilen.
```

Python Fortgeschritten

Einrückung: `
`

 eingerückter Text

Zelle ausführen: STRG + ENTER

- Kernel installieren

```
pip install ipython
```

```
ipython kernel install --name=KERNELNAME
```

jupyter neu starten

- Installierte Kernels anzeigen

jupyter kernelspec list

- Kernel deinstallieren

```
jupyter kernelspec uninstall KERNELNAME
```


Python Fortgeschritten

27) Exceptions – Kritische Fehler

- Exception fangen

```
import logging

logging.basicConfig(filename="LogFileName.log", level=logging.INFO,
format="%(%asctime)s; %(levelname)s; %(message)s")

def TestFunktion(variable1, variable2):
    try:
        print("Test Funktion wir ausgeführt.")
        print(int(variable1 / variable2))

    except TypeError as ex:
        logging.exception(ex)
        print(ex)
    except Exception as ex:
        logging.exception(ex)
        print(ex)

    finally:
        print("Code wird immer ausgeführt.")

TestFunktion(10, 2)
> Test Funktion wir ausgeführt.
> 5
> Code wird immer ausgeführt.

TestFunktion(1, None)
> Test Funktion wir ausgeführt.
> unsupported operand type(s) for /: 'int' and 'NoneType'
> Code wird immer ausgeführt.
```

Python Fortgeschritten

- Exception auslösen

```
def EigeneExceptionAusloesen(exceptionWennNone):
    try:
        # Prüfung auf None
        if exceptionWennNone is not None:
            print("Funktion wird ausgeführt.")
        else:
            raise Exception("Argument darf nicht None sein.")

    except Exception as ex:
        logging.exception(ex)
        print(ex)

EigeneExceptionAusloesen(2)
> Funktion wird ausgeführt.

EigeneExceptionAusloesen(None)
> Argument darf nicht None sein.
```

Python Fortgeschritten

28) Typische Fehler

- Tippfehler / Variable nicht definiert

```
a = 1
b = 2
print(a + n)
> NameError: name 'n' is not defined
```

- zu wenig / zu viele Parameter übergeben

```
def Addieren(zahl1, zahl2):
    print(zahl1 + zahl2)

Addieren(1, 2, 3)
> TypeError: Addieren() takes 2 positional arguments but 3 were given
```

- Objekt None

```
text1 = "Hello World ="
text2 = None
print(text1 + text2)
> TypeError: can only concatenate str (not "NoneType") to str
```

- Modul nicht gefunden

```
import numpy
> ModuleNotFoundError: No module named 'numpy'
```

- entweder Schreibfehler oder Modul nicht installiert

- Falsche Einrückung

```
for i in [1, 2, 3]:
    for k in [1, 2, 3]:
        print(i)

    print(i + k)
> IndentationError: unexpected indent
```

- Fehlende Klammer

```
for i in [1, 2, 3]:
    print(((i**2/2))
> SyntaxError: unexpected EOF while parsing
```

Python Fortgeschritten

- Endlosschleife

```
i = 1
while i < 10:
    print(i)

> 1
> 1
> ...
```

- Abbruch mit STRG + c

- Off by 1

```
i = 1
while i < 10:
    print(i)
    i += 1

> 1
> 2
> ...
> 9
```

- Auch zu beachten: Indexierung bei iterierbaren Datentypen fängt immer bei 0, und nicht wie intuitiv bei 1 an

- = statt ==

```
i = 21
if i = 21:
    print("i ist 21")

> SyntaxError: invalid syntax
```

29) Code Optimierung

- Überflüssige Berechnungen vermeiden
- Geschachtelte Schleifen minimieren
- Listen vor dem Suchen von Elementen sortieren
- Dictionaries anstatt Listen verwenden, wenn genug Arbeitsspeicher zur Verfügung steht
 - Space – Time Tradeoff

30) Danke! =)