Programmieren für Ingenieure

Zusammenfassung

https://pitrium.de/

Inhaltsverzeichnis

1. Erste Schritte

1.1. Python Download

- Python schon installiert?

```
C:\Users\Win10x64>py

Entweder:
> Python 3.9.6… - sehr gut
Oder:
> 'py' is not recognized as an internal or external command, operable program or batch file. – muss noch installiert werden
```

- https://www.python.org/downloads/

1.2. Hello World

- Hello World:

```
print("Hello World =)")

> Hello World =)
```

1.3. Entwicklungsumgebungen IDEs

- Texteditoren mit erweiterten Funktionalitäten – z.B.:
  - Visual Studio Code
  - PyCharm
  - Sublime Text
  - Visual Studio

1.4. Informationen finden

- https://stackoverflow.com/
- https://www.youtube.com/
- https://pypi.org/
- https://github.com/
- https://pitrium.de

2. Variablen

- Zwischenspeicher für Daten und Werte.

Datentyp	Bezeichnung	Gespeicherte Daten	Beispiel
String	str	Text	„Hello World“
Integer	int	Ganzzahl	123
Float	float	Gleitkommazahl	12,34
Boolean	bool	Wahr oder Falsch	True / False
None	NoneType	Verweis auf nichts	None
Liste	List	Sammlung von Elementen	[1, 2, 3]
Dictionary	Dict	Liste von Schlüssel – Wert Paaren	{1: 'Hello', 2: 'World', 3: '=)'}

- Namens Konventionen:
  - Camel Case: meinVariablenName
  - Pascal Case: MeinVariablenName
  - Underscores: mein_variablen_name

- Standard Funktionen
  - https://docs.python.org/3/library/functions.html

- Deklarieren, Wert zuweisen/überschreiben:

```
meineGanzzahl = 123
```

- Wert auslesen:

```
meineGanzzahl

> 123
```

- Datentyp abfragen:

```
meineGleitkommazahl = 12.3
print(type(meine Gleitkommazahl))

> <class 'float'>
```

3. Strings

- Texte
- Deklaration:

```
testString = "Hello World =)"
print(type(testString), testString)
testString2 = 'Das ist ein anderer Text.'
print(type(testString2), testString2)

> <class 'str'> Hello World =)
> <class 'str'> Das ist ein anderer Text.
```

- Escape Character:

```
print("Ich möchte gern ein \" in meinem Text haben.")

> Ich möchte gern ein " in meinem Text haben.
```

- Zeilenumbrüche:

```
print("Ich möchte gern \nin zwei Zeilen schreiben.")

> Ich möchte gern
> in zwei Zeilen schreiben.
```

- Verketten:

```
print("Will" + "kom" + "men =)")

> Willkommen =)
```

- Formatierung:

```
wer = "Felix"
wieViele = 3
print(f"{wer} hat {wieViele} Äpfel gekauft.")

> Felix hat 3 Äpfel gekauft.
```

- Zeichenanzahl:

```
print(len(testString))

> 14
```

- Numerisch / Alphanumerisch:

```
print("abc".isalpha())
print("123".isalnum())

> True
> True
```

- Großschreibung / Kleinschreibung:

```
print(testString.lower())
print(testString.upper())

> hello world =)
> HELLO WORLD =)
```

- Zerlegen:

```
print("1, 2, 3, 4, 5".split(", "))

> ['1', '2', '3', '4', '5']
```

- Ersetzen:

```
print(testString.replace("World", "WRLD"))

> Hello WRLD =)
```

- Teilen / Slicing / Indexing:

```
testString = “Hello World =)”
print(testString[6])
print(testString[0])
print(testString[-1])
print(testString[12:15])          # [Start (einschließlich) : Ende (ausschließlich)]

> W
> H
>)
> =)
```

- Enthalten Abfrage:

```
if "World" in “Hello World =)”:
        print("World ist ein Bestandteil von Hello World =).")

> World ist ein Bestandteil von Hello World =).
```

4. Integer, Float

- Zahlen und Berechnungen
- Deklaration:

```
zahl1 = 2
zahl2 = 5.7
print(f"{zahl1} \t {type(zahl1)}")
print(f"{zahl2} \t {type(zahl2)}")

> 2       <class 'int'>
> 5.7     <class 'float'>
```

- Grundrechenarten:

```
print(f"Addieren: {zahl1} + {zahl2} = {ergebnis}")
print(f"Subtrahieren: {zahl1} - {zahl2} = {zahl1 - zahl2}")
print(f"Multiplizieren: {zahl1} * {zahl2} = {zahl1 * zahl2}")
print(f"Dividieren: {zahl1} / {zahl2} = {zahl1 / zahl2}")

> Addieren: 2 + 5.7 = 7.7
> Subtrahieren: 2 - 5.7 = -3.7
> Multiplizieren: 2 * 5.7 = 11.4
> Dividieren: 2 / 5.7 = 0.3508771929824561
```

- Berechnungs Reihenfolge: Klammern, Punktrechnung, Strichrechnung:

```
print(f"1 + 1 * 2 = {1 + 1 * 2}")
print(f"(1 + 1) * 2 = {(1 + 1) * 2}")

> 1 + 1 * 2 = 3
> (1 + 1) * 2 = 4
```

- Potenzieren:

```
print(f"Potenzieren: {zahl1} ** {zahl2} = {zahl1 ** zahl2}")
print(f"Wurzel: {zahl1} = {zahl1 ** 0.5}")

> Potenzieren: 2 ** 5.7 = 51.98415336679908
> Wurzel: 2 = 1.4142135623730951
```

- Runden:

```
import math

print(round(2.1))
print(round(2.9))
print(round(2.5))
print(round(2.56789, 2))
print(math.ceil(2.1))
print(math.floor(2.9))
print(math.floor(10**2 * 2.999) / 10**2)

> 2
> 3
> 2
> 2.57
> 3
> 2
> 2.99
```

5. Boolean

- Wahrheitswerte
- Deklaration:

```
wahr = True
falsch = False
print(f"wahr = {wahr} \t {type(wahr)}")
print(f"falsch = {falsch} \t {type(falsch)}")

> wahr = True   <class 'bool'>
> falsch = False          <class 'bool'>
```

- Vergleiche:

```
x = 2
y = 5
print(f"x < y --> {x < y}")
print(f"x <= y --> {x <= y}")
print(f"x > y --> {x > y}")
print(f"x >= y --> {x >= y}")
print(f"x == y --> {x == y}")
print(f"x != y --> {x != y}")

> x < y --> True
> x <= y --> True
> x > y --> False
> x >= y --> False
> x == y --> False
> x != y --> True
```

- Verknüpfungen:

```
print(f"Wahr UND Wahr --> {True and True}")
print(f"Wahr UND Falsch --> {True and False}")
print(f"Wahr ODER Wahr --> {True or True}")
print(f"Wahr ODER Falsch --> {True or False}")
print(f"NICHT Wahr UND Wahr --> {not True and True}")
print(f"NICHT Falsch UND Wahr --> {not False and True}")

> Wahr UND Wahr --> True
> Wahr UND Falsch --> False
> Wahr ODER Wahr --> True
> Wahr ODER Falsch --> True
> NICHT Wahr UND Wahr --> False
> NICHT Falsch UND Wahr --> True
```

6. List

- Sammlung von Elementen
- Deklaration:

```
listeLeer = []
print(listeLeer)
print(type(listeLeer))

> []
> <class 'list'>
```

- Deklaration mit Inhalt:

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

> ['a', 'b', 'c']
> [9, 21, 0, 1, 2, 3, 4, 5, 6, 7]
> [1.1, 2.2, 3.3, 4.4, 5.5]
> [[1, 3], [3, 7]]
```

- List Comprehension:

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

initialeListe = [zahl for zahl in range(10)]
print(initialeListe)

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

- Elemente hinzufügen:

```
listeVonIntegers.append(9)     # Element an Ende der Liste anhängen
print(listeVonIntegers)
listeVonIntegers.insert(2, 8)  # Element an indizierte Stelle der Liste anhängen
                               # liste.append(Element, Index)
print(listeVonIntegers)
print(listeVonIntegers + [10, 11, 12, 13])      # Listen verknüpfen

> [9, 21, 0, 1, 2, 3, 4, 5, 6, 7, 9]
> [9, 21, 8, 0, 1, 2, 3, 4, 5, 6, 7, 9]
> [9, 21, 8, 0, 1, 2, 3, 4, 5, 6, 7, 9, 10, 11, 12, 13]
```

- Teilen / Slicing / Indexing:

```
print(listeVonIntegers[0])     # erstes Element
listeVonIntegers[0] = 99                # Überschreibt den Wert an der Stelle
mit Index 0 mit dem Wert 99
print(listeVonIntegers[0])
print(listeVonIntegers[-1])    # letztes Element
print(listeVonIntegers[2])     # Element an indizierter Stelle (3. Element)
print(listeVonIntegers[1:3])   # Elemente von Start (einschließlich) bis Ende
(ausschließlich)
print(listeVonIntegers[:3])    # Elemente von Start bis Ende (ausschließlich)
print(listeVonIntegers[2:])    # Elemente von Start (einschließlich) bis Ende

> 9
> 99
> 9
> 8
> [21, 8]
> [99, 21, 8]
> [8, 0, 1, 2, 3, 4, 5, 6, 7, 9]
```

- Elemente entfernen:

```
listeVonIntegers.remove(7)                # erstes Element mit Wert 7 entfernen
print(listeVonIntegers)
listeVonIntegers.remove(listeVonIntegers[1])   # Element mit Index 1
entfernen
print(listeVonIntegers)
print(listeVonIntegers.pop())             # entfernt und gibt das letzte Element
der Liste zurück
print(listeVonIntegers)

> [99, 21, 8, 0, 1, 2, 3, 4, 5, 6, 9]
> [99, 8, 0, 1, 2, 3, 4, 5, 6, 9]
> 9
> [99, 8, 0, 1, 2, 3, 4, 5, 6]
```

- Kennzahlen von Listen:

```
print(len(listeVonIntegers))                    # Anzahl der Elemente
print(min(listeVonIntegers))                    # Minimalwert
print(max(listeVonIntegers))                    # Maximalwert
print(sum(listeVonIntegers))                    # Summe
print(statistics.mean(listeVonIntegers))        # Durchschnitt
print(statistics.median(listeVonIntegers))      # Median
print(statistics.stdev(listeVonIntegers))       # Standardabweichung
print(statistics.variance(listeVonIntegers))    # Varianz

> 9
> 0
> 99
> 128
> 14.222222222222221
> 4
> 31.889566388467003
> 1016.9444444444443
```

- Methoden für Listen:

```
print(listeVonIntegers.index(2))    # gibt Index von erstem Element mit Wert 2
zurück
print(listeVonIntegers.count(99))    # gibt Anzahl von Elementen mit Wert 99
zurück
listeVonIntegers.sort()          # sortiert Elemente der Liste (aufsteigend)
print(listeVonIntegers)
listeVonIntegers.reverse()        # kehrt Reihenfolge der Elemente in der Liste
um
print(listeVonIntegers)

> 4
> 1
> [0, 1, 2, 3, 4, 5, 6, 8, 99]
> [99, 8, 6, 5, 4, 3, 2, 1, 0]
```

- Prüfen ob Element enthalten ist:

```
print(21 in listeVonIntegers)
print(0 in listeVonIntegers)

> False
> True
```

- Iterieren - Elemente von Liste durchlaufen:

```
for element in listeVonIntegers:
        print(element)

> 99
> 8
> ...
> 1
> 0
```

- Kopie von / Referenz auf eine Liste erstellen:

```
print(listeVonIntegers)

# 1) Referenz auf bereits bestehende Liste erstellen
kopierteListe = listeVonIntegers
kopierteListe.append(99)
print(kopierteListe)
print(listeVonIntegers)

# 2) Kopie einer bereits bestehenden Liste erstellen
kopierteListe2 = listeVonIntegers[:]
kopierteListe2.append(77)
print(kopierteListe2)
print(listeVonIntegers)

> [99, 8, 6, 5, 4, 3, 2, 1, 0]
> [99, 8, 6, 5, 4, 3, 2, 1, 0, 99]
> [99, 8, 6, 5, 4, 3, 2, 1, 0, 99]
> [99, 8, 6, 5, 4, 3, 2, 1, 0, 99, 77]
> [99, 8, 6, 5, 4, 3, 2, 1, 0, 99]
```

7. Dictionary

- Liste aus Elementen mit Schlüsselwort und Wert
- Deklaration:

```
dictionaryLeer = {}
print(dictionaryLeer)
print(type(dictionaryLeer))

> {}
> <class 'dict'>
```

- Deklaration mit Inhalt:

```
mitarbeiter = {1: ["Felix", "1988-09-21", 90000], 2: ["Mareike", "1992-11-24",
59000]}
print(mitarbeiter)
print(mitarbeiter.keys())
print(mitarbeiter.values())

> {1: ['Felix', '1988-09-21', 90000], 2: ['Mareike', '1992-11-24', 59000]}
> dict_keys([1, 2])
> dict_values([['Felix', '1988-09-21', 90000], ['Mareike', '1992-11-24', 59000]])
```

- Elemente hinzufügen:

```
mitarbeiter[3] = ["Olivia", "1988-03-19", 45000]
print(mitarbeiter)

> {1: ['Felix', '1988-09-21', 90000], 2: ['Mareike', '1992-11-24', 59000], 3:
['Olivia', '1988-03-19', 45000]}
```

- Werte abfragen mit Slicing / Indexing:

```
mitarbeiter[3][2] = 52000      # Herzlichen Glückwunsch zur Gehaltserhöhung
print(mitarbeiter)

> {1: ['Felix', '1988-09-21', 90000], 2: ['Mareike', '1992-11-24', 59000], 3:
['Olivia', '1988-03-19', 52000]}
```

- Werte mit Funktion abfragen:

```
print(mitarbeiter.get(3))
print(mitarbeiter[3])          # gibt Fehlermeldung zurück wenn
Schlüsselwort nicht gefunden wurde
print(mitarbeiter.get(5))      # .get() Funktion gibt None zurück wenn
Schlüsselwort nicht gefunden wurde

> ['Olivia', '1988-03-19', 52000]
> ['Olivia', '1988-03-19', 52000]
> None
```

- Element entfernen:

```
del mitarbeiter[1]
print(mitarbeiter)

> {2: ['Mareike', '1992-11-24', 59000], 3: ['Olivia', '1988-03-19', 52000]}
```

- Prüfen ob Element enthalten ist:

```
print(1 in mitarbeiter)
print(3 in mitarbeiter)

> False
> True
```

- Iterieren - Elemente von Dictionary durchlaufen:

```
for schluesselwort, wert in mitarbeiter.items():
        print(f"Schlüsselwort: {schluesselwort} - Wert: {wert}")

> Schlüsselwort: 2 - Wert: ['Mareike', '1992-11-24', 59000]
> Schlüsselwort: 3 - Wert: ['Olivia', '1988-03-19', 52000]
```

8. None

- Verweis auf None Objekt
- Deklaration:

```
variable = None
print(variable)
print(type(variable))

> None
> <class 'NoneType'>
```

9. Datentypen umwandeln

- Implizite Umwandlung (ohne Gefahr von Informationsverlust):

```
ganzzahl = 1
gleitkommazahl = 2.3
summe = ganzzahl + gleitkommazahl
print(summe)

> 3.3
```

- Explizite Umwandlung (mit Gefahr von Informationsverlust):

```
# Informationsverlust
ganzzahl = 1
gleitkommazahl = 2.9
ganzzahl2 = int(gleitkommazahl)
summe = ganzzahl + ganzzahl2
print(summe)

> 3
```

10. Abfragen - if, elif, else

Wenn (diese Bedingung erfüllt ist):
    dann wird dieser Code ausgeführt.
Oder wenn (diese Bedingung erfüllt ist)
    dann wird dieser Code ausgeführt.
Sonst:
    wird dieser Code ausgeführt.

- Einfache Abfrage:

```
bedingung1 = True
bedingung2 = True

if bedingung1:
   print("bedingung1 ist True.")
elif bedingung2:
   print("bedingung1 ist False und bedingung2 ist True.")
else:
   print("bedingung1 und bedingung2 sind False.")

> bedingung1 ist True.
```

- Verknüpfung von Abfragen:

```
if x < y and y < z:
   print("x < y and y < z")
else:
   print("Bedingung nicht erfüllt.")

> x < y and y < z
```

11. while Schleifen

Während (diese Bedingung erfüllt ist):
wird dieser Code ausgeführt.

Geeignet, wenn du nicht genau einschätzen kannst wie oft die Schleife durchlaufen werden muss bis die Eingangsbedingung nicht mehr erfüllt ist.

- Einfache while Schleife:

```
i = 0
while i < 5:
    print(f"Bedingung True - i = {i}")
    i = i + 1

> Bedingung True - i = 0
> Bedingung True - i = 1
> Bedingung True - i = 2
> Bedingung True - i = 3
> Bedingung True - i = 4
```

- while Schleife mit Abbruchbedingug:

```
while True:
    print(f"Immer noch True - i = {i}")
    if i == 9:
        break
    i += 1

> Immer noch True - i = 5
> Immer noch True - i = 6
> Immer noch True - i = 7
> Immer noch True - i = 8
> Immer noch True - i = 9
```

- Ausführung einer Endlosschleife unterbrechen:
  - Strg + c

```
while True:
    print(“Hello World =)“)

### ERGEBNIS / AUSGABE ###
> Hello World =)
> Hello World =)
> Hello World =)
…
> Traceback (most recent call last):
>   File "<stdin>", line 1, in <module>
> KeyboardInterrupt
```

12. for Schleifen

Für jeden Wert der VARIABLE im Bereich (von – bis):
wird dieser Code ausgeführt.

Geeignet, wenn du bereits weißt wie oft deine Schleife durchlaufen werden soll.

- Einfache for Scheife:

```
for i in range(5):
    print(f"Bedingung True - i = {i}")

> Bedingung True - i = 0
> Bedingung True - i = 1
> Bedingung True - i = 2
> Bedingung True - i = 3
> Bedingung True - i = 4
```

- Variationen der range() Funktion:

```
for i in range(1,10,2):
    print(f"Bedingung True - i = {i}")
for i in range(0,-5,-1):
    print(f"Bedingung True - i = {i}")

> Bedingung True - i = 1
> Bedingung True - i = 3
> Bedingung True - i = 5
> Bedingung True - i = 7
> Bedingung True - i = 9
> Bedingung True - i = 0
> Bedingung True - i = -1
> Bedingung True - i = -2
> Bedingung True - i = -3
> Bedingung True - i = -4
```

- Elemente von iterierbaren Datentypen durchlaufen (z.B. String, List, …):

```
testString = "Hello World =)"
for zeichen in testString:
        print(zeichen)

> H
> e
>...
> d

> =
>)
```

13. 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

14. Funktionen

- Wiederverwendbare Code Abschnitte mit Namen
  - Funktion deklarieren:

```
def NameDerFunktion():
    print("Code meiner Funktion wird ausgeführt.")

NameDerFunktion ()

> Code der Funktion wird ausgeführt.
```

  - Funktionsparameter:

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

Addieren(2, 3)

> 5
```

  - Rückgabewerte:

```
def RechteckFlaecheUndUmfang(laenge, breite):
    """DocString um die Funktionsweise einer Funktion zu beschreiben"""
    flaeche = laenge * breite
    umfang = 2 * (laenge + breite)
    return flaeche, umfang

laenge = 2
breite = 3
flaeche, umfang = RechteckFlaecheUndUmfang(laenge, breite)
print(f"Länge = {laenge}; Breite = {breite}; Fläche = {flaeche}; Umfang =
{umfang}")

> Länge = 2; Breite = 3; Fläche = 6; Umfang = 10
```

15. Module

- Sammlung von Code
  - Module importieren:

```
import time
from datetime import datetime
import math as m

for i in range(3):
    print(f"{datetime.now()} - {m.pi}")
    time.sleep(1)

> 2020-04-23 19:07:08.299398 - 3.141592653589793
> 2020-04-23 19:07:09.314949 - 3.141592653589793
> 2020-04-23 19:07:10.315151 - 3.141592653589793
```

  - Eigene Module / Dateien importieren:

```
import NameDerDatei


NameDerDatei.NameDerSagHalloFunktion("Mia")

> Hallo Mia =)
```

  - Priorität für die Suche nach Modulen/Dateien:
    1. Python Standard Modul
    2. *.py Datei mit entsprechendem Namen im gleichen Verzeichnis
    3. Verzeichnisse die in PYTHONPATH Variable gespeichert sind

  - Modul installieren das im Python Paketmanager (pip) verfügbar ist:

```
pip install NameDesModuls
```

  - Liste installierter Module anzeigen:

```
pip list
```

16. Logging

- Konfigurieren:

```
import logging

logging.basicConfig(filename="LogFileName.log",
level=logging.getLevelName(settings.find("LoggingLevel").text),
format="%(asctime)s; %(levelname)s; %(message)s")
```

- Einträge schreiben:

```
logging.debug("Debug logging ...")
logging.info("Prozessdaten Auswetung gestartet.")
logging.warning("Warnung logging ...")
logging.error("Error logging ....")
logging.critical("Kritisches logging ...")
```

LogFileName.log

```
2021-08-19 10:51:26,132; INFO; Prozessdaten Auswetung gestartet.
2021-08-19 10:51:26,304; WARNING; Warnung logging ...
2021-08-19 10:51:26,445; ERROR; Error logging ....
2021-08-19 10:51:26,772; CRITICAL; Kritisches logging ...
```

17. Exceptions - Fehlerhandhabung

- Syntax:

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

    except TypeError as ex:
        print(ex)
    except Exception as ex:
        print(ex)

    finally:
        print("Code der immer ausgeführt wird, unabhängig davon ob Exception
ausgelöst wurde oder nicht.")


TestFunktion(10, 2)

> Test Funktion wir ausgeführt.
> 5
> Code der immer ausgeführt wird, unabhängig davon ob Exception ausgelöst
wurde oder nicht.

TestFunktion(1, None)

> Test Funktion wir ausgeführt.
> unsupported operand type(s) for /: 'int' and 'NoneType'
> Code der immer ausgeführt wird, unabhängig davon ob Exception ausgelöst
wurde oder nicht.
```

18. Häufige 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 numpyy

> ModuleNotFoundError: No module named 'numpyy'
```

  - 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
```

- 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
```

19. Einfache 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

20. Anwendung

1. Ziel so detailliert wie möglich definieren
2. Forschung und Entwicklung
   - Wurde die gleiche oder eine sehr ähnliche Aufgabe bereits gelöst? (GitHub)
   - Gibt es bereits Anleitungen, Forschungsarbeiten, Videos zu diesem Thema? (Google)
3. In Bestandteile / kleinere Unteraufgaben zerlegen
4. Einfachen Prototyp bauen
   - einzelne Bestandteile entwickeln und nach und nach zusammensetzen (StackOverflow, Modul Dokumentation))
   - schnell zu entwickeln, einfach zu verstehen und anzupassen
5. Optimierung und Praxis Vorbereitung
6. Stabilisieren, Instandhaltung

21. Ausführung

1. Ausführung in der IDE
    - während der Entwicklung oder Debugging
    - F5 in den meisten IDEs

2. Ausführung der .py Datei
    - auf Rechner mit Python
    - *.py Datei doppelt anklicken

3. Ausführung über Eingabeaufforderung
    - Um Fehlermeldungen bei Programmabstürzen zu sehen
    - Eingabeaufforderung (cmd)
        i. Navigieren: cd C:\Pfad\Zu\Deinem\Programm
        ii. Ausführen: HelloWorld.py
        iii. TAB zum Auto-vervollständigen von Namen

4. Ausführung mit Hilfe einer .bat Datei
    - Wenn mehrere Befehle nacheinander ausgeführt werden sollen

```
REM In richtigen Ordner navigieren
cd /d
D:\x\xxx\Projects\Pitrium\Kurse\ProgrammierenFuerIngenieure\Beispielprog
ramm_Prozessdaten_Optimiert

REM Daten kopieren
DateienKopieren_4.py

REM Daten prozessieren
DatenProzessieren_4.py
```

5. Ausführung als .exe Datei
    - auf Rechnern ohne Python
    ```
    pip install pyinstaller

    pyinstaller -F -i "Icons\DateienKopieren.ico" "DateienKopieren_4.py"
    ```

    - Als Batch Datei:
    ```
    REM In richtigen Ordner navigieren
    cd /d
    D:\x\xxx\Projects\Pitrium\Kurse\ProgrammierenFuerIngenieure\Beispielprog
    ramm_Prozessdaten_Optimiert

    REM Exe Dateien erzeugen
    REM pip install pyinstaller
    pyinstaller -F -i "Icons\DateienKopieren.ico" "DateienKopieren_4.py"

    pyinstaller -F -i "Icons\DatenProzessieren.ico" "DatenProzessieren_4.py"

    PAUSE
    ```

6. Geplante Ausführung mit Aufgabenplanung
    - Automatische Zeit- oder Ereignis- gesteuerte Ausführung

7. Ausführung auf Raspberry Pi
    - Energieeffizienter 24/7 Rechner
    - gut kontrollierbare Testumgebung vor Umzug in Cloud Server
    - privater 24/7 Rechner (Cloud ist auch nur ein fancy Wort für: Rechner der irgendjemand anderem gehört)

    - IP Adresse finden:
    ```
    ping raspberrypi.local
    ```

    - SSH Verbindung starten:
    ```
    ssh pi@IP_ADRESSE
    ```

    - VNC installieren, starten, verbinden:
    ```
    sudo apt update
    sudo apt install realvnc-vnc-server

    vncserver :1 -geometry 1920x1080

    raspberrypi.local::5900
    ```

8. Ausführung auf Google oder Amazon Server
    - Zuverlässige, günstige und beliebig skalierbare (CPU, RAM, GPU) 24/7 Ausführung