# Programmieren

Hier gehts rein ums Programmieren

- Python Programmieren
- HTML Schriftarten und Bilder
- Powershell Befehle
- CMD Windows
- Powershell Masterclass
- Powershell Programmbeispiele
- Machine Learning (Udemy Kurs)
- Tools mit Python entwickeln
- Checksumme vergleichen
- Bash Skripting
    - Benutzereingabe
    - Parameter übergeben
    - Zufall Random
    - For-Schleife mit Zufallspasswort Random und While-Schleife
    - Case - Kontrollstruktur
    - For-Schleife
    - Datei per Skipt verschlüsseln
- Shell programmieren
- Flutter
    - Flutter Programmierung
- Dart
    - Grundlegendes zu Flutter
    - Dart
- Flutter, Dart App Programmierung

- Flutter Befehle

# Python Programmieren

Übersicht über Befehle und Beispiele

## Skript in Linux mit Pythoninterpreter starten

```
#!/usr/bin/env python
```

# Operatoren

```
< kleiner
> größer
<= kleinergleich
>= größergleich
== gleich
!= ungleich
in = Enthält eine Liste einen bestimmten Wert
not = Negiert die Aussage
// = geteilt mit Runden
```

** = Hochgestellt ( x ** 2 = Quadrat von x)

# Boolean definieren:

Um ein true oder false zu setzen muss das folgendermaßen geschehen:

```
True
False
```

Wobei der erste Buchstabe groß geschrieben werden muss

Typ einer Variablen anzeigen lassen

```
type(name)
```

ID einer Variablen anzeigen lassen

```
id(name)
```

Eine Eingabe machen:

```
var = int(input(„Eingabe machen:“))
```

# Strings

Umwandeln:

Int in einen String umwandeln

```
age = 22


print("Ich bin" + str(age))
```

## Slicing von Strings

```
String = "0123456789"
print("gesamter String mit String [:] =", string[:])
print("ab dem 5. Zeichen mit string [4:] =", string[4:])
print("5. bis 7. Zeichen mit string [4:7] =", string [4:7])
```

// in Integer umwandeln

```
int(a)
```

// in Gleitkommazahl umwandeln

```
float(b)
```

## Runden

zum Runden einfach folgende Funktion verwenden:

```
gleitkommazahl = 2.3456677


gleitkomma_gerundet = round(gleitkommazahl, 2) # Rundet die Zahl auf zwei stellen nach dem Komma
```

# String → in eine Liste zerteilen

Mit `Split` werden die einzelnen Strings zwischen dem „Trenner“ als Liste gespeichert.

.split(" ")

```
i = "Monika, Max, Erik"
print (i.split(", ")
```

# Replace

Zeichen in einem String austauschen

```
float(wert.replace(',','.') # Ersetzt im Wert das Komma durch einen Punkt
```

# Strings formatieren

Mit Hilfe von .format können Strings formatiert werden:

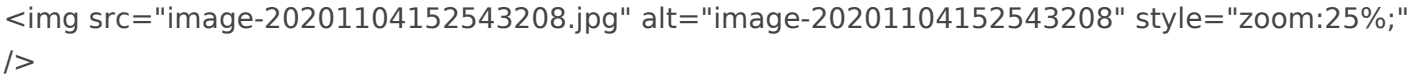
```
ausgabe = "hallo {0}, du hast noch {1} Euro auf dem Konto".format("Hermann", 123.13)


print(ausgabe)
```

Hallo Hermann, du hast noch 123.13 Euro auf dem Konto

... mit Formatierung für den Kontostand: auf 2 Stellen, Breite 10 Zeichen, führende Nullen

```
ausgabe = 'Hallo {0}, du hast noch {1:010.2f} EURO auf dem Konto!'.format('Anna', 123.4567)
```

Hallo Anna, du hast noch 0000123.46 EURO auf dem Konto!



Formatierung:

f = Kommazahl

d = Ganzzahl

links-, rechts-bündig, zentriert

...Name auf 10 Zeichen linksbündig

```
ausgabe = 'Hallo {0:<10s}, du hast noch {1} EURO auf dem Konto!'.format('Anna', 123.4567)


print(ausgabe)
```

...Name auf 10 Zeichen rechtsbündig

```
ausgabe = 'Hallo {0:>10s}, du hast noch {1} EURO auf dem Konto!'.format('Anna', 123.4567)
print(ausgabe)
```

...Name auf 10 Zeichen zentriert

```
ausgabe = 'Hallo {0:^10s}, du hast noch {1} EURO auf dem Konto!'.format('Anna', 123.4567)
print(ausgabe)
```

Hallo Anna , du hast noch 123.4567 EURO auf dem Konto!

Hallo Anna, du hast noch 123.4567 EURO auf dem Konto!

Hallo Anna , du hast noch 123.4567 EURO auf dem Konto!

# Listen

## Unterschiedliche Arten

Liste: `liste = [ ]`
Dictionary: `dictionary = { }`
Tupel: `tupel = ( )`

Wie ein Array mit den Eckigen Klammern eine Variable definieren

```
students = ["Max", "Monika", "Erika", "Franziska"]
```

Man kan auch eine Liste mit dem List Construktor generieren um Beispielsweise alle Zahlen von bis hinzuzufügen:

```
liste = list(range(1,50) # Fügt Liste eine Liste mit Zahlen zwischen 1 und 49 hinzu
```

Hinzufügen:

```
students.append("Moritz“)
```

Einfügen:

```
students.insert(0, "Hermann")
```

An der 0. Stelle wird Hermann hinzugefügt

**Elemente zählen**: Wieviele Inhalte hat eine Liste:

```
len(liste)
```

Letztes Element einer Liste **löschen**

```
liste.remove("Hermann") # Löscht Hermann aus einer liste
del liste[2] # Lösche an Index 2
liste.pop(2) # Löscht den Wert an Position 3
liste.clear() # leert die Liste
gelöscht = liste.pop(2) #Man kann auch den gelöschten wert in einer variabelen speichern
liste.sort() #Liste sortieren

# Listen zusamenfügen
students = ["Max", "Monika"] + ["Lisa", "Franziska"]
```

# In Liste suchen

Suchen mit Index

```
variableVon2 = variable.index(2)
```

Liste ausgeben: liste → String

```
.join(liste)
```

// benötigt einen Seperator (Was zwischen den Werten liegen soll)

```
print(", ".join(liste))
```

# List Slicing

```
print(students[-1]) # Gibt das letzte Element aus
print(students[1:]) # Dann fängt die liste bei Index 1 an
print(students[2:5]) # Gibt die Werte 2 bis 4 aus
print(students[1:-1]) # Enfernt das erste und das letzte Element der Liste
```

# Liste Sortieren

```
print(sorted(liste)
print(sorted(liste, reverse=True) # Umgekehrt sortieren
```

Sortieren eines Tupel nach einem Namen

```
def sort_helper(tupel):

    return tupel[1] #Wird nach der Position 1 Sortiert

l = [(1, 'Paul', 19), (2, 'Anna', 31), (3, 'Anna', 20)
print(sorted(l, key=sort_helper))
```

# List Comprehensions

```
xs = [1, 2, 3, 4, 5, 6, 7]

ys = [x * x for x in xs]

# Ist das Gleich wie:

ys = []
for x in xs:
  ys.append(x * x)
```

Kann aber noch mehr:

```
students = ["Max", "Monika", "Erik", "Franziska"]

lengths = []
for student in students:
  lengths.append(len(student))

# Als Comprehension
lengths = [len(student) for student in students]
```

# Tupel

```
t = (1, 2, 3) # Runde Klammern erstellen ein Tupel
```

Unterschied zu einer Liste:
Listen können mit `liste.append(5)` verändert werden, das geht bei einem Tupel nicht. Auch können keine Daten überschrieben werden.

**Warum brauchen wir das?**

Dadurch kann nicht aus versehen was verändert werden.
`person = ("Max Müller", 55)`
Die Person kann dann nicht verändert werden

# Arbeiten mit Tupel

```
student = ("Max Müller", 22, "Informatik")
name, age, subject = student # die Variablenanzahl links vom = muss mit dem Inhalt des Tupel übereinstimmen


print(name)
print(age)
print(subject)
```

Wenn eine Funktion mehrere Werte zurück geben soll, dann packt man diese Werte in ein Tupel und kann so mehr als nur eine Varialbe zurückgeben.

Beispiel:

```
def get_student():
  return ("Max Muster", 22, "Informatik")


name, age, subject = get_student()
```

# Liste mit Tupel

```
students = [
  ("Max Müller", 22)
  ("Monika Mustermann", 23)
]

for name, age in students:
  print(name)
  print(age)
```

*Max Müller*
*22*
*Monika Mustermann*
*23*

# Dictionary

**Syntax:**

Geschweifte Klammer: {}

Key und Value mit : getrennt

"IN":"Ingosltadt"

```
staedte = {"IN":"Ingosltadt"}
□□□□□Key □□Value
```

# Ohne Werte:

```
staedte = {}
```

**Wert hinzufügen**

```
staedte[RO] = "Rosenheim"
```

**Wert auslesen:**

```
print(staedte[IN])
```

*Ingolstadt*

**Wert löschen**

```
del staedte[RO]
```

**Key enthalten in Dictionary**

```
k in dict # liefert True, falls der Schlüssel k in dict enthalten ist
□□□□# k muss vorher deklariert werden
```

**Maximal Wert in einem Dictionary mit Funktion:**

```
nameHaeufigkeit = {} # Dictionary
maxName= max(nameHaeufigkeit, key=nameHaeufigkeit.get) # Gibt den Key aus, der den größten Value hat
nameHaeufigkeit[macName] # gibt den Value dazu aus
```

## Dictionaries und Schleifen

```
d = {"München": "MUC", "Budapest":"BUD", "Helsinki": "HEL"}

for key in d:
  value = d[key]
  print(key)
  print(value)



for key, value in d.items():
  print(key + ": " + value)
```

## Verschachteln von Liste in Dictionary

```
students = {
    "Informatik": ["Max", "Monica"],
    "BWL": ["Erik", "Franziska"]
}

print(students["Informatik"])
```

*["Max", "Monica"]*

Der Key Informatik enthält die Liste mit Max und Monika

# Set

Ein Set ist eine Liste in der Jedes Element nur einmal vorkommen kann.

Erzeugung:

```
einSet = set()
```

entweder:

```
begruessung = set('Hallo Welt!')
```

oder

```
staedte1 = {'Ingolstadt', 'Augsburg', 'München'}
```

**Wichtige Operationen auf sets:**

len(s) liefert die Anzahl Elemente in Set s □□
s.add(e) fügt e in Set s ein
s.clear() s. □□□□löscht alle Elemente in s
discard(e) löscht e in s, falls e existiert□□
s.remove(e) löscht e in s, falls e existiert. Anderfalls wird eine KeyError-Exception geworfen □□liefert die
s.union(s2) □□Vereinigungsmenge von s und s2
s.intersect(s2) liefert die Schnittmenge von s und s2□□
s.difference(s2) liefert das Komplement s.intersect(s2)

# Queue

Noch eine Datenstrucktur: Queue (Warteschlange)

Dabei werden die Daten hinein gepackt und danach wieder heraus genommen

```
import queue


q = queue.Queue()
q.put(„hallo“)
q.get()
```

# Kontrollstrukturen

**if-Anweisung**

Mit if und else können Kontrollen gemacht werden.

Innerhalb der If-Anweisung muss eingerückt werden.

```
n = 42
if n < 42:
  print("Die Zahl ist kleiner als 42")
```

```
else:
  print("Die Zahl ist größer oder gleich 42")
print("Hermann ist der beste")
```

**Vergleichen**

```
if a > b:


   print("a ist größer als b")
```

# Schleifen

## While

```
counter = 0
while counter < 10:
    print(counter)
    counter = counter + 1
print("Hallo Welt")
```

## For

For i in liste:

Es wird eine Liste abgearbeitet.

```
liste = [5, 8, 10]
for i in liste:
  print(i)
for i in range(0, 4):
print(i)
```

*0*
*1*
*2*
*3*

**Schlüsselwörter in Schleifen**

continue

Bricht den aktuellen durchlauf ab und springe direkt zum nächsten durchlauf

break

Bricht die komplette Schleife ab

**Range**

```
range(0, 10)
```

Gibt die Werte von 0 bis 9 wieder in 1er Schritten.

Will ich andere Schrittweite:

```
range(0, 10, 2)
```

Dann wird mit der 2 angegeben das es Zweierschritte sein sollen.

Beispiel

```
for counter in range(0, 10):
  print(counter)
```

# Fehler abfangen und Fehlermeldung erstellen

Mit Try und except

except ValueError

Fehlerhafte Eingaben

```
try:
    zahl1 = input("Ganze Zahl (Mit Fehlerbehandlung): ")
    zahl1AlsInt = int(zahl1)
    zahl2 = input("Ganze Zahl (Mit Fehlerbehandlung): ")
    zahl2AlsInt = int(zahl2)

    ergebnis = zahl1AlsInt * zahl2AlsInt
    q = zahl1AlsInt / zahl2AlsInt

    print("Ergebnis =", ergebnis)

except ValueError:
```

```
    print("Du hast keine Zhal eingegeben:", zahl)
    exit()

except Exception:
    print("Unbekannter Fehler")
    exit()
```

# Input

Input selber liefert immer einen String

## Programmierfehler finden und lösen

Wenn ein Fehler Programmiert wurde, dann gibt es die einfache Möglichkeit, denn TypeError Fehler im ganzen zu kopieren und dann in Google zu suchen.

Gute Seite für das Debuggen ist die Seite Stackoverflow

# Module laden

Um Python mit verschieden Sachen zu erweitern kann man Module laden.

## Random

Mit Random können Zufallszahlen generiert werden.

import random für den import des Moduls

```
random.randint(0, 4) #gibt Zufallszahlen zwischen 0 und 4 wieder
```

Zufälligen String aus einer Liste auswählen

```
import random

strings = ['string1', 'string2', 'string3', 'string4']

random_string = random.choice(strings)

print(random_string)
```

Gültige Eingabe erzwingen mit Schleife

Die Schleife ist immer aktiv. Wird eine richtig Zahl eingegeben bricht die Schleife ab.

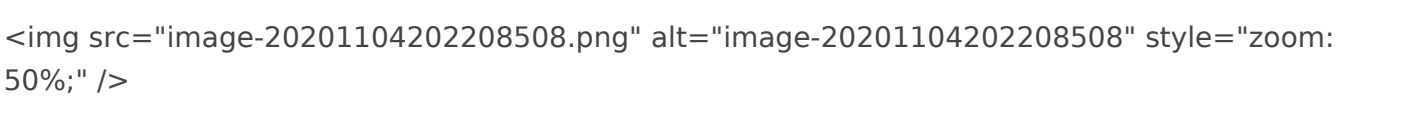
```
while True:
    try:
        zahl = int(input("Zahl: "))
        break

    except Exception:
        print("Bitte gültige ganze Zahl eingeben!")
```

# Verschiedene Module

## Grafiken zeichnen

Damit kann man Plotten und eine Grafik zeichnen

```
import matplotlib.pyplot as plt
xs = [1, 2, 3]
ys = [4, 7, 4]

plt.plot(xs, ys)
plt.show()
```



## Sys

Mit Sys können Parameter übergeben werden

```
import sys


if sys.arg[1] == '0'
```

In pythonista werden die mitgelieferten Parameter auf Pos [1] übergeben.

# Objekte und Klassen

image-20201104202244657

Title

# Self

Wie das mit dem Self in Klassen funktioniert kann der Udemy-Kurs ganz gut erklären

`__init__`

Damit werden beim initialisieren einer Klasse paramter abgefragt die unbedingt erstellt werden müssen

```
class Student():
  def __init__(self, firstname, lastname):
    self.firstname = firstname
    self.lastname = lastname
```

__ beide Unterstriche definert eine Provate Eigenschaft

# Private Eigenschaften

Verhindern von zugriff auf Eigenschaften einer Klasse

```
erik._term
# mit dem Unterstrich geben wir an, dass auf diese Eigenschaft nicht zugegriffen werden soll


erik.__term
# Durch die __ zwei Unterstriche kann nicht mehr auf die Eigenschaft zugegriffen werden
```

`__slots__`

Verbietet das weiter Attribute erstellt werden

`__str__`

ist die Funktion die Aufgerfuen wird wenn print ausgeben wird

# Vererbung

```
class WorkingStudent(Student):
  def __init__(self, firstname, lastname, company):
    super().__init__(firstname, surname) # Holt die Init-Methode aus der Elternklasse
```

```
 def name(self):
  return "WorkingStudent: " + self.firstname + self.lastname # überschreibt die Name-Funktion aus der Elternklasse.

 #kann auch so gemacht werden:
 def name(self):
  return super().name() + self.company # Dabei wird das an die Elternfunkton angehängt.
```

## Besondere Methoden

`__str__`

damit können bestimmt sachen ausgegben werden wenn auf ein objekt zugegriffen wird

`__len__`

damit kann man die Länge einer bestimmten Sache zurück geben

## Getter und Setter

In einer Klasse dürfen nicht auf die Attribute direkt zugreifen. Deshalb werden Setter und Getter als Funktion erstellt.

Um einen Wert aus einer Klasse zu holen den Getter benutzen:

```
def getSomething():
    return self.something
```

Um ein Attribut in einer Klasse zu verändern benutzt man einen Setter:

```
def setSomething(input):
    self.attribute = input
```

# Import Module

Die Liste mit den Modulen sind hier zu finden

Selbstgeschriebene aber auch integrierte Module können wie folgt importiert werden:

```
import xyz
```

Dabei wird das gesamte Modul geladen. Und beim Aufruf einer Funktion aus diesem Modul, muss das Modul separat angegeben werden:

```
xyz.methode()
```

Werden aber die Methoden direkt importiert:

```
from xyz import methode
```

dann werden die Methoden auch direkt eingeben:

```
methode()
```

## Ordner in ein Modul verwandeln

Damit python einen Ornder als Modul ansieht muss in diesem Ordner eine Datei

```
__init__.py
```

Liegen.

Um alle Module mit dem * laden zu können muss in die *_init_*.py Datei folgendes hinzugefügt werden

```
# in __init__.py
__all__ = ["datei"]
```

Um das ganze Modul zu laden (import modul - ohne from und import) muss in die *_init_*.py Datei folgendes hinzugefügt werden:

```
from . import datei
```

# Sonstiges:

Printausgabe ohne Zeilenvorschub

Durch den Parameter end wird der Standardwert von \n überschrieben

```
print(club, end='::')
```

Umwandeln von Buchstaben in Kleinbuchstaben

```
.lower()
```

String-Repräsentation

```
__str__()
```

## ID generieren

import uuid

uuid.uuid1

# Module verfügbar machen

Wenn eigene Module geladen werden sollen, muss in dem Ordner, in dem die Module liegen auch immer die `__init__.py` -Datei koniguriert werden.

mit

```
__all__ = ["CrawledArticle", "ArticleFetcher"] # Die Module werden verfügbar gemacht


from .CrawledArticle import CrawledArticle # Die Funktionen sind verfügbar
from .ArticleFetcher import ArticleFetcher
# Der . vor dem Modul gibt an, dass das Modul im selben Ordner liegt
```

# Dateioperationen

Infos im Internet

# Um eine Datei zu öffnen:

```
txt = open('dateipfad')
```

Txt ist eine Variable in die der Inhalt der Text-Datei gespeichert wird.

```
print(txt.read())
```

Gibt den Text der Variablen in der Konsole aus.

```
txt.write("\nNeue Zeile")
```

Schreibt eine neue Zeile in die variable txt

Eine weitere Möglichkeit um eine Textdatei auszugeben ist line

```
print(line)
```

Wenn eine Textdatei ausgelesen wird, und Zeile für Zeile Ausgeben dann werden auch Steuerzeichen ausgegeben (z. B. \n). Um das zu verhindern benutzt man .strip()

Beispiel:

```
file = open("lesen.txt", "r")
for line in file:
  print(line.strip())
```

# Um eine Datei zu schreiben:

```
File = open("schreiben.txt", „w“) # das w benötigt man für den Schreibzugriff

File.write("hermann\n")
File.wirte("ist der beste")

File.close() # Datei wieder schließen

Optionen für Open
'r'= öffnen
'w' = schreiben
'a'= append (anhängen)
```

Sicheres Schließen von Dateien, wenn vorher ein Fehler passieren sollte:

```
  open("lesen.txt", "r") as file:
for line in file:
Print(line)
```

Dann kümmert sich Python ums schließen. Ein Close ist nicht mehr nötig. Allerdings wird dann nach dem 'with'-Block die Datei wieder geschlossen.

Um jetzt den Pfad der Datei auszuwerten könnte man mit

```
my_label = Label(root, text=root.filename).pack()
```

Auf dem Fenster ausgeben.

# Dateien laden und speichern

Zum laden von pickle-Dateien wird das Modul pickle benötigt

`import pickle`

Um zu überprüfen ob Dateien vorhanden sind, die man laden will, muss das Modul os importiert werden

`import os`

Überprüfen ob Datei vorhanden:
if os.path.exists(dateipfad):

Laden einer Datei

```
objektvariable = open('dateipfad', 'r')
```

Inhalt der Datei in eine Variable speichern

```
dumpfile = open(dateipfad, 'rb‘)
```

Wobei rb einen binären Lesezugriff gibt.

```
liste = pickle.load(dumpfile)
```

Speichert den Inhalt der Dumpfile in eine Liste danach kann es wieder verwendet werden.

**Achtung** Wenn eine Class noch nicht angelegt ist kann es zu Fehlern kommen.

# Speichern einer Datei

```
objektvariable = open('dateipfad', 'w')


dumpfile = open(dateipfad, 'wb')
```

Binärer Schreibzugriff auf die Datei

```
pickle.dump(liste, dumpfile)
```

Die Pickle Funktion nimmt dabei den Inhalt der liste und packt sie in die dumpfile.

```
dumpfile.close()
```

Wieder schließen.

# Generatoren

`yield`

In einer Funktion wird mit yield das angefragte zurück gegeben. Wenn in einer Schleife die Funktion immer wieder aufgerufen wird. Siehe Udemy Kurs: Crawler

## Umlaute aus Datei auslesen

Zunächst wird die Textdatei die Umlaute enthalten kann in Binär umgewandelt und danach in UTF-8 zurück geschrieben.

```
liste = []


with open ("zitate.txt", "r") as file:
    for line in file:
        binari = line.encode('iso-8859-1').strip()
        liste.append(binari.decode('utf-8'))


print(liste)
```

---

# Tkinter - Einfache GUI

```
# Importieren der Module


from tkinter import Tk, messagebox, Label, Button, Frame, Entry, Checkbutton, Radiobutton
```

*Programmierbeispiel > Private Projekte > Backup S13*

## Grunsätzliche Komponenten

**Tk** Braucht man grundsätzlich

**Label** - Statischer Text im Fenster

**Button** - Durch Klick Command ausführen

**Textfeld** (Entry) - Feld für Texteingaben

**Combobox** - Auswahl Klappliste mit Einfachauswahl

**Checkbox** - Klick

**Radiobutton** Einfachauswahl mit mehreren Optionen

Mit Frame kann eine Zelle mit mehrerem Inhalt belegt werden.

```
teilfenster = Frame(Hauptfenster)
Teilfenster.grid(row = 1) # Somit ist teilfenster in Reihe 1 in Hauptfenster.
```

columnspan

```
e.grid(row=0, column=0, columnspan=3, padx=10, pady=10)
```

Bewirkt, dass unter dem Grid 3 Spalten entstehen.

```
insert
```

Um Text in einen Entry als Vorauswahl zu platzieren:

```
entry.insert(0, 'text')
```

Wobei 0 die Position definiert.

Um einen Button einen Parameter im Kommando zu übergeben benötigt man lambda

```
command=Lambda: button_click()
```

Icon

```
window.iconbitmap('pfad')
```

Bilder verwenden

Modul: from PIL import ImageTk, Image

# Erstellen der Komponenten

*Hinweis: formular ist der Frame in dem die Komponenten platziert werden*

image-20201105090917500

**Label** Text der Angezeigt wird

```
Label(formular, text="Vorname: ").grid(sticky="W", row=0)
```

**Entry** Textfeld zur Eingabe

```
vorname_feld = Entry(formular) # Variable mit Entry erstellen
vorname_feld.grid(sticky="W", row=1, column=1) # Positioneieren des Entry
```

```
# Um einen String in ein Entry zu übergeben:
vorname_feld.insert(0, variable) # 0 setzt den Corser auf eine Position
# Textfeldinhalt löschen
vorname_feld.delete(0, "end")
```

**Radiobuttons** Auswahl eines von mehreren

```
geschlecht_frame = Frame(formular) # Erzeugt Frame im Frame formular
geschlecht_frame.grid(sticky="W", row=2, column=1) # Positionieren
geschlecht = StringVar()
geschlecht.set("w") # Vorauswahl
geschlecht_raido1 = ttk.Radiobutton(geschelcht_frame, text="weiblich", variable=geschlecht, value="w")
geschlecht_raido1.grid(sticky="W", row=0, column=0)
... mit 2 und 3 weitermachen
```

**Checkbox** Kontrollkästchen

```
glaeubig = StringVar()
glaeubig.set("ja") # Vorauswahl
glaubig_checkbox = Checkbutton(formular, text="", command=glaeubig_change, variable=glaeubig,
onvalue="ja", offvalue="nein") # glaeubig_change ist eine funktion, die die Änderung auf der Konsole ausgibt
(wurde so festgelegt von mir)
```

**Combobox** Dropdown

```
glaube = StringVar()
glaube.set("keine Angabe")
glaube_combobox = ttk.Combobox(formular, textvariable=glaube)
```

```
glaube_combobox["values"] = ("keine Angabe", "katolisch", "evangelisch")
glaube_combobox.bind("<<ComboboxSelected>>", glaub_change) # Event-Handler um zu sehen, welcher Glaube ausgewählt wurde
```

**Button** Schalter

```
ok_button = Button(formular, text="Ausgabe", width=10, command=behandle_ausgabe)
```

> **Dieses Programm:**
>
> Befindet sich unter 01 Grundlagen der Programmierung > Python Beispiele > werbistdu_dialog.py

# Ordner öffnen

Mit tkinter kann man Ordner öffnen.

```
from tkinter import *
from tkinter import filedialog

root = Tk() #Fenster erzeugen

root.title('Codemy.com Image Viewer') #Titel

root.filename = filedialog.askopenfilename(initialdir="/Users/hermannp/Library/Mobile Documents/com~apple~CloudDocs/Lernen und Studieren/Programmieren Python",title="Datei auswählen")

root.mainloop()
```

**Askopenfilename Optionen:**

- Initialdir: Standardort
- Title: Fenstertitel

**Weitere Optionen:**

- File - Typ auswählen

```
# PNG Dateien oder Alle Dateien auswählen
filetyps=(("png files","*.png"), ("all files","*.*"))
```

# TKinter formatieren

```
# Hintergrund-Farbe ändern
bg = "yellow"
bg = "black"
# Textfarbe ändern
fg = "white"
```

# Optionen für Widgets in tkinter

| Option | Erklärung |
|---|---|
| bd, border-width | Ganze Zahl, die die Breite des Rahmens in Pixel angibt |
| bg, background | Hintergrundfarbe |
| fg, foreground | Vordergrundfarbe (Textfarbe) |
| font | Font-Deskriptor für den verwendeten Schrifttyp |
| height | Höhe des Widgets in Pixel |
| image | Name eines Bildes (Image-Objekt), dasd auf dem Widget zu sehen ist |
| justify | Ausrichtung von Textzeilen auf dem Widget: CENTER: zentriert, LEFT, RIGHT |
| padx | Leerer Raum in Pixel rechts und links vom Widget oder Text |
| pady | Leerer Raum in Pixel über und unter dem Widget oder Text |
| relief | Form des Rahmens: SUNKEN, RAISED, GROOVE, RIDGE, FLAT |
| text | Beschriftung des Widgets (z. B. Button oder Label) |
| textvariable | Ein Objed der Klasse StringVar, das den (variablen) Text enthält, der auf dem Widget (z.B. Button oder Label ) erscheint |
| underline | Default ist -1. Wenn die Zahl nicht negativ ist, gibt sie die Nummer des Zeichens an, das unterschrichen sein soll |
| width | Breite des Widgets (horizontal) in Pixel, z. B. 100 |

# Widget nachträglich konfigurieren

```
widget.config(fg='red') # Textfarbe wird rot gesetzt
```

# Schriftarten

Schriftarten werden mit einem Tupel formatiert (famile, größe, [stil])

```
label = Label(fenster, text='Hallo Welt', font=('Verdana', 40, 'bold'))
```

| Schriftarten | Stile | |
|---|---|---|
| Verdana | bold | fett |
| Times | italic | kursiv |
| Comic Sans MS | overstrike | durchgeschtrichen |

# Farben

Farben für Hintergrund (bg) oder den Text (fg) können mit wort oder #rgb #rrggbb angeben werden

```
labelnacht = Label(fenster, text='Nacht', font=('Arial', 20), fg='white', bg='#000000')
```

| Farben | Hex |
|---|---|
| white | ffffff |
| black | 000000 |
| red | ff0000 |
| green | 00ff00 |
| blue | 0000ff |
| cyan | 00ffff |
| yellow | ffff00 |

# Rahmen

- Attribut bd (borderwidth) gibt die Breite des Rahmens an
- Attribut relief beschreibt die Form
  - SUNKEN, RAISED, GROOVE, RIDGE, FLAT

# Größe

- Breite mit width
- Höhe mit height

# Leerraum um Text

- padx und pady

## Gemeinsame Methoden der Widgets

| Methode | Erklärung |
|---|---|
| after (ms , func[,arg1[,...]]) | Aufruf einer Funktion oder Methode nach ms Millisekunden |
| bell() | Erzeugt Glockenklang |
| bind(sequence=event, func=f[,add='+']) | Bindet die Funktion f (Eventhandler) an einen Event |
| config(option1=wert1, ....) | Das Widget wird neu konfiguriert, die angegebenen Optionen erhalten neue Werte |
| destroy() | Das Widget und alle Nachkommen in der Parent-Child-Hierachie werden gelöscht. |

# Passwörter verborgen eingeben

```
import getpass
```

Getpass wird danach verwendet wie Input. Auch mit prompt. Nur dass dann während der Eingabe keine Zeichen zu sehen sind.

```
Passwort = getpass.getpass(,Passwort eingeben: ,)
```

# Fehlerbehandlung in Funktionen

Prüfen ob eine Parameter den richtigen Typ enthält

```
def equals(self, anderePosition):
    if type(anderePosition) != '<class __main__.Position>'
        raise Excepetion('Invalid parameter type ' + type(anderePosition))
```

Mit **raise** mache ich eine eigene FehlerAusgaben

```
class InvalidEmailError(Exception):
    pass


def send_mail(email, subject, content):
    if not "@" in email:
        raise InvalidEmailError("Email hat kein @")
try:
    send_mail("hallo", "Betreff", "Inhalt")
```

```
except InvalidEmailError:
    print("Bitte gebe eine gültige E-Mail Adresse ein")
```

hier würde `raise` den InvalidEmailError mit der Meldung "Email hat kein @-Zeichen werfen. Da wir aber mit Try und Except den Fehler InvalidEmailError abfangen, wird der Print-Befehl ausgegeben.

# Finally

Finally wird bei einem

```
try:
...
finally:
...
```

immer ausgeführt und kann dann zum beispiel eine Datei sicher wieder schließen. Das Finally wird immer ausgeführt, egal welcher Fehler auftritt.

# With

`with open ("datei.xyz", "r") as file:` Damit wird eine Datei geöffnet. Wird `with` verlassen, schließt python selbstständig die Datei wieder, damit sie wieder geschlossen ist und von anderen Benutzern verwendet werden kann.

```
with open ("./pfad/datei.xyz", "wb") as file: □□# Die Datei wird schreibend geöffnet und zwar in Binär
□file.wirte(inhalt) □□□□□□□□# Schreibt in die Datei den Inhalt rein
```

# Notizen

Heraussprungen aus zwei Schleifen

```
i = 0


ende = False


while True:
□while True:
□□print(i)
```

```
    if i > 10:
        ende = True
        break
    i += 1

if ende = True:
    break
```

# Tools

| Name | Funktion |
| --- | --- |
| requests | Damit kann man Seiten mit Python herunterladen |
| beautifulsoup | Zerlegt ein HTML in seine Bestandteile |
| UrlJoin | Damit kann man URLs zusammensetzen aus verschiedenen Teilen |
| CSV | Erstellen einer CSV-Datei |
| EXIFread | Metadaten aus Bildern auslesen |

# Beispiele

Jupiternotebook: Bilder Downloaden

Jupiternotebook: Exif Daten auslesen

# requests

Mehr infos zu requests

```
r = requests.get("http://www.google.com") # Lädt die HTML von google.com in die Variable r
r.status_code # Damit kann abgefragt werden, ob die Seite erreichbar ist if (status == 200)
r.headers # Kopfdatei anzeigen
```

# beautifulsoup

Mehr infos zu beautifulsoup

```
doc = BeautifulSoup(r.text, "html.parser") # r ist von requests
□□□□□□□□□□# damit wird in doc der text von r gespeichert und aufbereitet
doc.find_all("img") # kann mann alle Bilder der Seite finden
image.attrs("src") # gibt das Attribut Quelle des Bildes wieder
```

# UrlJoin

```
from urllib.parse import urljoin #Damit können Urls erstellt werden
urljoin("http://irgendwas", "./img/1.jpg")
#Ergibt dann folgende Ausgabe:
#http://irgendwas/img/1.jpg
```

# CSV

Mehr Infos zu CSV

## CSV Lesen:

```
import csv

with open('datei.csv', newline='') as csvfile:
  reader = csv.reader(csvfile, delimiter=",", quotechar='"')
  for row in reader:
    print(row)
```

> Hier wird die Datei.csv geöffnet. In die Variable reader wird die CSV-Datei abgelegt und an den ',' getrennt.

**Weitere Möglichkeit eine CSV in einer Liste zu speichern:**

```
with open(csv_datei, 'r', newline='', encoding='utf-8') as file:
    eintraege = list(csv.reader(file))
```

## Erste Zeile überspringen

```
with open('empfaenger.csv', newline='') as csvfile:
  reader = csv.reader(csvfile, delimiter=";", quotechar='"')
  next(reader, None) # Überspringt die erste Zeile
  for row in reader:
    print(row)
```

## CSV schreiben

Um eine CSV Datei zu schreiben geht man folgender Maßen vor:

```
import csv
with open('datei.csv', 'w', newline='') as csvfile:
  line = csv.writer(csvfile, quotechar='"', delimiter=";")
  line.writerow([ersteSpalte, zweiteSpalte])
```

> Erstellt eine csv-Datei mit dem Namen: datei.csv
> Schreibt jedes Zeile in eine Neue Zeile
> und trennt die Teile mit einem ";" (Das gibt der delimiter an. Wenn dieser nicht gesetzt wird, ist das Trennzeichen ein ",")

## CSV Reihe anhängen

```
with open('document.csv','a', newline='') as csvfile:
    writer = csv.writer(csvfile, delimiter=";")
    writer.writerow([wert1, wert2])
```

Wenn Sie eine Datei mit dem `'a'` Parameter öffnen, können Sie sie an das Ende der Datei anhängen, anstatt einfach den vorhandenen Inhalt zu überschreiben. Versuch das.

# Exifread

```
import exifread


tags = exifread.process_file(Bilddatei)
```

# Sleep, Zeitstempel

Um eine gewisse Zeit zu warten benutzt man sleep

```
from time import sleep, strftime


...
sleep(1) # Wartet 1 Sekunde
...
zeitstempel = strftime("%d.%m.%Y %H:%M:%S") # Gibt den aktuellen Tag und Zeit wieder
```

# Datetime

Aktuelle Zeit abfragen

```
import datetime
```

## Heutiges Datum einfügen

```
import datetime

jetzt = datetime.datetime.now()

formatiert = jetzt.strftime("%d.%m.%Y %H:%M:%S")
```

## Dauer eines Programm berechnen

um die Laufzeit zu messen:

```
import time
start = time.time()
...
ende = time.time()

dauer = ende - start
```

# Shell-Befehle verwenden

os.subprocess

```
# Beispiel


import subprocess


the_command = ["ipconfig"]


with subprocess.Popen(the_command, stdout=subprocess.PIPE, stderr=subprocess.PIPE) as proc:
  stdout=(proc.stdout.read())
  stderr=(proc.stderr.read())


# print("stdout: %s" % (stdout))
stdout2 = stdout.decode('cp1252')
print(stdout2)
print("stderr:%s" % (stderr))


# print("stderr: %s" % (stderr))
```

# Modul 2

# Rechner und Betriebssysteme

## Binäre Zahlen

Umrechnung von Dezimal in Binär und umgekehrt

## Addieren von Binärzahlen

Kommazahlen: Umrechnung von Dezimal in Dual:

0,3 * 2 = 0,6 => 0
0,6 * 2 = 1,2 => 1
0,2 * 2 = 0,4 => 0 ...

Binär: 0,010...

Umrechnung von 0,10011 binär in Dezimal

0,1 0 0 1 1 =
1 1 1 1 1
2 4 8 16 32

# Negative Binäre Zahlen:

## Zweierkomplement

Alle Bits invertieren und 1 dazu zählen

# Zahlenbereiche:

| Anzahl Bits. | Kleinste Zahl | Höchste Zahl |
|---|---|---|
| 8 | -128 | +127 |
| 16 | -32768 | +32767 |
| 32 | -2.147.483.648 | +2.147.483.647 |

## Subtrahieren

Zweikomplement der Zahl erstellen, die abgezogen werden soll und addieren

## Hexadezimale Zahlen

Schreibweise in Python:
0x96D
0x wird vorangestellt

**Text darstellen**
ASCII (American Standard Code für Information Interchange) Seit 1963
mit allen Sonderzeichen wird der Unicode Standard verwendet

## TCP/IP Modell

Die 5 Schichten des Internets

# HTML Schriftarten und Bilder

# Schriftarten HTML CSS

**Arial**

```
<p ID="code"><font face="arial">Konstanze will sich gut benehmen ist aber leider doof</font></p>
```

**Times New Roman**

```
<p ID="code"><font face="times new roman">Konstanze will sich gut benehmen ist aber leider doof</font></p>
```

**Palatino**

```
<p ID="code"><font face="Palatino">Konstanze will sich gut benehmen ist aber leider doof</font></p>
```

**Verdana**

```
<p ID="code"><font face="Verdana">Konstanze will sich gut benehmen ist aber leider doof</font></p>
```

**Calibri**

```
<p ID="code"><font face="Calibri">Konstanze will sich gut benehmen ist aber leider doof</font></p>
```

**Tahoma**

```
<p ID="code"><font face="Tahoma">Konstanze will sich gut benehmen ist aber leider doof</font></p>
```

**Monaco**

```
<p ID="code"><font face="Monaco">Konstanze will sich gut benehmen ist aber leider doof</font></p>
```

**Courier New**

```
<p ID="code"><font face="Courier New">Konstanze will sich gut benehmen ist aber leider doof</font></p>
```

# Bilder mit Unterschrift

```
<dl class="bildunterschrift">
 <dt><img src="bildname.jpg" alt="Bildbeschreibung" title="Bildbeschreibung"></dt>
 <dd>Bildunterschrift</dd>
</dl>
```

```
<dl class="bildunterschrift">
<dt><img src="https://basic-tutorials.de/wp-content/uploads/2013/05/Definitonsliste-für-Bildunterschrift-nutzen.jpg" alt="Bildbeschreibung" title="Bildbeschreibung"></dt>
<dd>Bildunterschrift</dd>
</dl>
```

# Powershell Befehle

# Befehlsübericht

| Comandlet | Beschreibung |
|---|---|
| `Get-command` | Zeigt alle Befehle an |
| `get-host` | Zeigt die Version von Powershell an |
| `Get-Computerinfo` | Gibt die Computerinfo wieder |
| `Get-Content` | Damit kann eine Datei ausgelesen werden Beispiel: Get-Content C:\Users\XY\text.txt |
| `Get-ChildItem` | Ordner Anzeigen |
| `clear-host` | Bildschirm leeren |
| `Write-host` | Print Befehl |
| `Read-host` | Benutzereingabe in der Konsole |
| `Set-Location` | Ändert das Verzeichnis von Powershell |
| `Start-Transcript` | Aufzeichnen der Powershell |
| `Start-Sleep` | Zeit warten |
| `Get-PSDrive` | Zeigt alle zur verfügungstehenden Laufwerke an |
| `Get-NetIPConfiguration` | Zeigt Netzwerkkarte an |
| `Get-Help` | Zeigt die Hilfe zu einem Cmdlet an Beispiel: Get-Help Test-Connection auführlich: -detailed |
| `Get-Help x  -Online` | Zeigt die Umfangreichere Onlinehilfe an Beispiel: Get-Help Test-Connection -Online |
| `Test-Connection` | Ping ausführen, kann mehrere Pings gleichzeitig ausführen<br>Beispiel: Test-Connection orf.at,8.8.8.8 -Count 1 -Quiet |
| `Test-NetConnection` | Beipiel: Test-NetConnection 25.25.15.3 -TraceRoute Macht einen Ping auf ein Gerät. - TraceRoute gibt den Weg zurück |
| `Tracert` | Genauere TraceRoute |
| `Req query` | Auslesen von Registry Eintragen |
| `Remove-Item` | Datei löschen |

| Comandlet | Beschreibung |
| --- | --- |
| `Psexec` | Remotezugriff Ausführen eines Befehls auf einem Remoterechner |
| `Copy-item` | Beispiel: Copy-Item -Path Pfad+Datei -Destination Zielpfad |
| `CSV` | https://techexpert.tips/powershell/powershell-read-lines-from-csv-file/ |
| `Get-Hotfix` | Zeigt die letzten Windowsupdates an |

# Ausführungsrichtlinien

Um zu verhindern, dass ps1 Dateien unberechtigter weise ausgeführt werden können gibt es die ExecutionPolicy. Diese ist normalerweise undifined und damit können keine PS Dateien ausgeführt werden.

```
# Anzeigen
Get-ExecutionPolicy -List

# Ändern
Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy Bypass
```

# Vergleichsoperatoren

| Befehl | Beschreibung |
| --- | --- |
| -eq | gleich |
| -ne | ungleich |
| -lt | kleiner |
| -le | kleiner oder gleich |
| -gt | größer |
| -ge | größer oder gleich |

# Parameterbeschreibung

| -WhatIf | Zeigt was geschehen würde wenn… |
| --- | --- |
| -Confirm | Fordert zur Bestätigung auf |
| -Verbose | Zeigt an was gerade geschieht |

# Weitere Befehle

Systeminformationen anzeigen

```
msinfo32
```

Computername anzeigen:

```
nslookup +ip-Adresse
```

Systeminformationen anzeigen:

```
Systeminfo
```

Gruppenrichtlinien als HTML ausgeben

```
gpresult /H datei.html
```

Gruppenrichtlinien manuell updaten

```
gpupdate /?
```

Anpingen...

```
ping -a
```

Weg verfolgen zu einem Rechner

```
tracert
```

# Code-Beispiele

## Liste von ip-Adressen anpingen

```
1..255 | ForEach { Write-Host 169.254.66.$_, "-",
([System.Net.NetworkInformation.Ping]::new().Send("169.254.66.$($_)")).Status}
```

# Remote einschalten

```
Set-Service -ComputerName haeb2pc4 -Name WinRM -StartupType Automatic -Status Running
Invoke-Command -ComputerName haeb2pc4 -Scriptblock {
    Get-ComputerInfo # In dem Scriptblock kann alles eingegeben werden, was auch auf einem Lokalen Rechner
verwendet werden kann
}
Set-fService -ComputerName haeb2pc4 -Name WinRM -StartupType Manual
```

# Anmeldeinformationen speichern

```
$credentials = Get-Credential
$credentials | Export-Clixml -Path "C:\Pfad\Zur\Datei\credentials.xml"
```

# Anmeldeinformationen abrufen

```
$credentials = Import-Clixml -Path "C:\Pfad\Zur\Datei\credentials.xml"
```

# Netzlaufwerk verbinden

```
$driveLetter = "Z"
$networkPath = "\\ServerName\SharedFolder"

net use $driveLetter: $networkPath /user:$($credentials.UserName)
$($credentials.GetNetworkCredential().Password)
```

**Trennen**

```
$driveLetter = "Z:"
net use $driveLetter /delete
```

# CMDledts

- Get- Verben nehmen niemals Änderungen vor

# CMDlets erkunden

```
Start-Transcript #Logfile erzeugen
Set-Location HKLM:  #Wechselt in die Registry

## Parameter

- Whatif #zeigt was geschen würde wenn
- Confirm #Fordert zur Bestätigung auf
- Verbose Zeigt an was gerade geschiet
```

# Hilfen anzeigen

Um mit den Comandos besser zurecht zu kommen kann man sich die Hilfen dazu wie folgt anzeigen lassen:

`Get-Help Get-Date -ShowWindow`

Bei dem Befehl wird die Hilfe von Get-Date in einem neuen Fenster angezei

# Pipe

Die Pipe "|" lenkt das Ergebnis von der linken Seite an die rechte Seite weiter

# Get-Hotfix angepasst

```
Get-Hotfix
```

# Foreach

```
$testrechner = "10.198.48.102","10.198.48.103"  # Erstellt ein Array mit zwei IP-Adressen

Foreach ($i in $testrechner)                    # Foreach Schleife die jedes Element in Testrechner durchgeht
{ Test-Connection $i -Count 1 -Quiet }          # Führt einen Ping auf die Elemente aus dem Array aus
```

# Formatierung und Ausgabe

## Format Table

```
# Ausgabe
Get-Process -Name VSSVC

# Umleitung an Format-Table
Get-Process -Name VSSVC | Format-Table Id,ProcessName

# Herausfinden wie Atribite heißen
Get-Process -Name VSSVC | Get-Member
```

## Format List

```
Get-Process -Name VSSVC | Format-List ID,ProcessName
```

## Out-CMDlets

```
Get-Command | Out-File C:\Temp\cmdlets.txt # Speichert die Commandos in eine Datei aus

# OUt-Gridview erstellt ein Grafisches Raster

Get-Process | Out-GridView -PassThru | Stop-Process # Damit kann ich ein Fenster erstellen und danach den Prozess beenden
```

# Variable $_

($_ und $PSItem ist das selbe)

Beinhaltet das aktuell von der Pipeline verarbeitet Objekt

```
# Beispiel
Get-Process | Where-Object { $_.CPU -GT 10 } # Filtert nach Objekten die > 10 sind Bei jedem Durchlauf wird geschaut ob das Objekt was durch $_ zurückgegeben wird größer ist
```

# Filtern

```
Get-ADComputer -Filter 'enable -eq $false' | irgendwas
# Hier wird vor der Pipline schon gefilter. Dadurch wird das Programm viel schneller, da es nicht alle Computer in die Pipeline weiter gibt.
```

# Sort-Object

```
Get-Process | Sort-Object
```

# Select-Object

```
Get-Process | Sort-Object -Proberty XXX | Select-Object -Last 5
# Mit Select-Object * kann man sich alles anzeigen lassen.

# Spezilles Parameter aus einem Objekt am Beispiel Get-ComputerInfo
Get-ComputerInfo | Select-Object OsName,
```

# Ping mit ExpandProberty

```
$b = Get-ADComputer -Filter * | Select-Object -ExpandProperty Name
# Dadurch wird der Wert abgerufen. Damit kann man weiter arbeiten, da es kein Objekt ist.

Test-Connection $b
```

# Objekte und Klassen

Wie bei Python kann auch in Powershell eine Klasse erstellt werden. Hier ein Beispiel

```
# Klassen

class Person {
    [string]$Name
    [int]$Age

    Person([string]$name, [int]$age) {
        $this.Name = $name
        $this.Age = $age
    }



    [void] SayHello() {
        Write-Host "Hello, my name is $($this.Name) and I am $($this.Age) Years old"

    }
}

$person = [Person]::new("John", 30)
$person.SayHello()
```

# Powershell Kurs für Administratoren und Udemy

# Schulungsunterlagen

> Willkommen bei Skillpipe!
> Hallo Hermann Pelzer,
> Mit Skillpipe kannst du deine Trainingsmaterialien jederzeit und überall auf deinem Lieblingsgerät aufrufen. Dein Lernfortschritt, die Markierungen, Notizen und Lesezeichen werden automatisch auf all deinen Geräten synchronisiert.
>
> Zugriff auf Skillpipe

# Dateien

202205-PowerShell-Wiederholungsfragen.pdf

Alle Dateien

---

Als Administrator starten

Verknüpfung bearbeiten und dort auswählen die Powershell als Administrator zu starten

# Welche Powershell version

`$PSVersionTable`

Commandlet

# Befehle

| | | Beispiel |
|---|---|---|
| Get-Process | Taskmanger | |
| Get-Command | | Get-Command -Verb Install |
| Get-Help Get-ChildItem -Online | Zeigt die Onlinehilfe an. Was aber bei uns meistens nicht funktioniert | |
| Get-ChildItem | Inhalt eines Ordner anzeigen | |
| where-Object | Zum Filtern von Ergebnissen | `-Filterscript {$_.handles -gt 1000}` |
| Select-Object | Hier kann man sich bestimmte Parameter anzeigen lassen | `Select-Object CsName, OsNmae, OsInstallDate` |
| Format-Table | | |

| | | Beispiel |
|---|---|---|
| Get-Member | Gibt eigenschaft wieder | |
| Get-NetIPConfiguration | Hier kann auf die ergebnisse wie auf ein Array zugegriffen werden | |
| Format-List | Damit kann man ausgegebene Listen formatieren | |
| | `>>` mit den beiden Pfeilen kann man an eine Datei anhängen | |
| | `Get-Content .\myfile.txt -Wait` | -Wait Wenn man live zusehen will, wie zum Beispiel eine Textdatei angepasst wird |
| `Export-Csv` | Als CSV ausgeben | |
| `Import-Csv` | | |
| `Export-Clixml` | Als XML exportieren | Dadurch werden nicht alle daten in einen String umgewandelt wie bei CSV |
| `Add-Computer` | Computer zu einer Domain hinzufügen | Für Remote muss man Credential angeben weil man sich sonst nicht anmelden kann. |
| `$cred = Get-Credential` | Damit kann man Benutzername und Passwort eingeben | |
| `$cred = New-Object -TypeName pscredential -ArgumentList 'Administrator',(ConvertTo-SecureString -String 'Pa$$w0rd' -AsPlainText -Force)` | Damit kann man die Credentials direkt einspeichern, damit das nicht extra eingegeben werden muss. | |
| `Test-ComputerSecureChannel -Repair` | Kann Computer-Clients in der Domain reparieren. | Mal ausprobieren, wenn es hier probmele gibt. <br />Test-ComputerSecureChannel kann annzeigen ob der Computer einen Fehler hat. |
| `1..3` | 1 bis 3 | eine Schleife die von 1 bis 3 zählt |
| `(Get-Date).AddDays(4)` | Gibt das heutige Datum + 4 Tage aus | |
| `"{} {}" -f $Temperatur,$Bedeckung` | -f arbeitet wie `.format()` aus python | |
| `New-Item -Path \\rechner\C$\ -Name ScriptShare -ItemType Directory` | Erstellt auf einem Rechner einen Ordner in C | |

# Notizen

Cmdlets bestehen immer aus Verb-Noun

Get bekomme etwas

Set etwas verändern

new um was neues zu erstellen

> Tipp: Mit Strg + Leertaste kann man sich alle Befehle anzeigen lassen

aliase: Andere Schreibweise für befehle. Machen dann aber das gleich wie das original

Wildcards * oder ? wobei Fragezeichen nur ein Zeichen ersetzt

Hilfe Updaten: Update-Help

Hilfe ausfühlricher:

```
Get-Help name -Detailed
```

Weiteres zu Hilfen:

```
get-help about_if -ShowWindow #zeigt wie man if verwenden kann.
```

```
Show-Command # Damit kann man Cmdlets erstellen
```

# Umwandeln von Bite

`Wert / 1GB` # Gibt GB wieder

---

# Variablen

```
$a = 100 # Variablen deklarieren


#Variable auslesen
$a


Get-Variable # Zeigt alle bestehenden Variablen an
```

```
# Arrays
# Leeres Array erstellen:
$myprocesses = @()

# Hashtables (=Dictionary)
$myHT = @{ Marke='Audi'; kw=140; Coupe=$false }
# Abfrage der Werte:
$myTH.Marke
```

Man kann auch Variablen innerhalb einer Zeichenkette mit Anführungsstriche direkt verwenden

```
"das ist ein Text mit $variable"

# Hashtablewerte in einen String übergeben
"Das ist ein Text mit einem Value $($wetter.temperatur)"
```

Skript muss immer in geschweiften Klammern sein

```
Get-Service | Where-Object -FilterScript { $_.Name -like 'x*' }
```

Filtert das Service auf Objekte die mit X beginnen

# Aus einem Computer einen DomainController machen

Man muss erst mal die Tools installiert werden, damit ein Rechner als DomainController arbeiten kann.

```
Get-WindowsFeature # Listet auf welche Features der PC hat

# Das ist nur auf einem Server möglich
Install-WindowsFeature -Name AD-Domain-Services -IncludeAllSubfeatures -IncludeManagementTools
```

# Datum verwenden

```
$myDate = Get-Date -Year 2021 -Month 5 -Day 25
$myDate.Date # Uhrzeit wird nicht betrachtet
| where-Object { $_.lastwritetime.D -eq $myDate } # Filtern zum 25.05.
```

# Provider

Sind Programme, die auf andere Programme zugreifen

`Get-PSProvider`

| Provider | |
|---|---|
| Get-PSDrive | Zeigt alle Laufwerke an |
| `New-PSDrive - Name Scriptshare -Root \\Lon-Srv\c$\ScriptShare -PSProvider FileSystem` | Bindet das Laufwerk vom Server in die Powershell auf dem Client ein und man kann damit arbeiten. |
| | |

# Spezielle Variable:

$env: - Damit kann man auf Variablen und auf Computereigenschaften zugreifen

# WMI

> Windows Management Instrumentation:

Cimclass

WQL ist die Sprache für WMI

# Registry Editor

HKEY_Current_User: Einstellungen die der User macht
HKEY_Local_Machine: Einstellungen der Machine für alle User

```
cd hkcu #Wechsel ins Regestry Laufwerk


Get-Childitem #listet alle Schlüssel auf


#Neues Element in Software erstellen
cd .\Software\
New-Item -Name Az040 -Path . -ItemType Key
cd .\Az040\
```

```
New-ItemProberty -Path . -Name TlnAnzahl -Value 6 -ProbertyType dword
Set-ItemProberty -Path . -Name TlnAnzahl -Value 7 #Erstellt keinen neuen Wert kann aber einen vorhandenen bearbeiten
(Get-ItemProberty -Path .).Level # gibt den value von level wieder
```

mit sapien kann man aus ps1 auch exe machen kostet aber geld

Sapien Editor

isesteroids kann die ise erweitern

# Skripte

> Skripte Starten
>
> - Pfad angeben um das Skript zu starten
> - Ausführungsrichtlinie kann verhindern, dass ein Skript gestartet wird
>   - Get-ExecutionPolicy -List
>   - Get-ExecutionPolicy > Restricted
>   - mit Bypass wird die Policy nicht überprüft

## Beispiele

```
<# Mehrzeilige Kommentare
geht so.
ziemlich einfach#>
hostname

#region Region
mit dem region kann man bereiche ausklappbar machen fuer bessere uebersicht
#endregion

# Um einen Parameter bei starten eines Skripts abzufragen
param (
   $status
)
```

## Parameter in Skript übergeben

```
# Verpflichtende Parameter
param (
    [parameter(Manatory=$true, Helpmessage='Bitte Parameter eingeben')]
    [validateSet('Running','Stopped')] #Welche werte sind erlaubt
    $status
)
```

# Bedingungen

## if

```
# if

if ( $a -gt 50 ) {

    mach das
}
else {

    mach jenes
}
```

## Switch

```
# Switch

Switch ($color) {
    'Yellow'{
        'Die Farbe ist gelb'
    }
    default {
        "Die Farbe ist unbekannt"
    }
}

# Schleife abbrechen
break
return
```

# Schleifen

## For

```
# for ( $zähler; Endbedingung; Schrittweite)

for ( $i = 1; $i -le 5; $i = $i +1 ) {
    "Der Zähler hat den Wert $i"
}
```

## foreach

> Hilfreich bei Arrays zu bearbeiten

```
$colors = 'White', 'Yellow', 'Magenta', 'Cyan', 'Darkgray'

foreach ($temp in $colors) {
    write-Host "Rainbow" - ForgroundColor $temp
}
```

## Do while

```
$i = 1

do {
    $i
    $i++
} while ( $i -le 10)
```

```
$Kapital = 1000
$Zinssatz = 0.01
$Jahre = 0

$Endkapital = 1500

do {
    $Kapital = $Kapital * $Zinssatz + $Kapital
    $Jahre++
} while ($Kapital -le $Endkapital)
```

```
"Das Kapital muss $Jahre Jahre angelegt werden"
```

# Fehlerbehandlung

`Set-StrictMode -Version latest` dann werden mehr Fehler angezeigt

```
try {
$b = Read-Host "Bitte einen Wert eingeben: "
}
catch{
   "Ui, ein Fehler ist aufgetreten"
   return # Mit return kann man das Scipt direkt abbrechen. Gibt man noch eine Zahl mit, dann kann man einen Fehlercode mitgeben
}
```

Bei Cmdlets muss man die -ErrorAction auf Stop setzen um bei einem Try eine Fehlermeldung zu bekommen

modul 4,5,7

# Remoting

```
Enable-PSRemoting

Get-PSSessionConfiguration

#Verbinden mit Remote

Get-Process w* -ComputerName xxx #Zeigt Prozesse eines anderen Rechner an

# Methode um auf anderem Rechner Cmdlets auszuführen
Invoke-Command -Computername xxx -ScriptBlock { Get-Process w* }

# Skript auf einem anderem Rechner ausführen
Invoke-Command -Computername xxx -FilePath C:\Script01.ps1

# Lokale Variablen werden über den Skriptblock nicht übergeben außer mit using
-ScriptBlock { Get-Service | Where-Object { $_.status -eq $using:status} | Select-Object -First 5 }
```

```
# Dauerhafte verbindung zu einem Rechner
$dc1Session = New-PSSession -Computername xxx

Invoke-Command -Session $dc1Session -Skriptblock { .. }
```

# Invoke-Command Info

Kann nur ausgeführt werden wenn WinRM läuft

```
Get-Service -ComputerName xxx -Name WinRM # Nachschauen ob WinRm in Running ist

# WinRM starten und Automatisch beim h
Set-Service -Name WinRM -CumputerName xxx -StartupType Automatic -Status Running
```

# Funktionen

Speichern unter als ps1 Datei oder besser .psm1

```
# Modul erstellen
function Write-Hello {
    $Get-Corp
    return
}

# Modul öffnen
Write-Hello

# Module Anzeigen
Get-Module -Listavailable -Name XXX -Refresh
```

Pfad zu Dokumenten:

`$env:PSModulpath`

# Hintergrund Aktionen

```
Start-Job -name MyFirstJob -ScriptBlock { Get-ChildItem 'C:\Program Filse\' -Recurse}

# Aufrufen des Jobs:
Get-Job

# Ausgabe des Jobs:
Receive-Job -Job (Get-Job -Id 2) #Wird neu ein einziges mal angezeigt. Besser speichern

# Man kann auch den Parameter -asjob verwenden
-asjob

# Warten bis alle Jobs erledigt sind:
Get-Job | Wait-Job
```

# Automatische Jobs

```
$jobtrigger = New-JobTrigger -At (get-ate).AddMinutes(1) -Once

$joboptions = New-ScheduledJObOpti
```

# Udemy

# Wiederholung Teil 1/3

## Select-Object

```
Get-ComputerInfo |
Select-Object `
CsName, OsName, OsInstallDate, OsLastbootUptime |
Format-Table -AutoSize -Wrap # Macht ggf. Zeilenumbruch
```

# Attribute

Alle Attribute auflisten

`Get-Disk | Get-Member  # Zeigt alle Attribute von Get-Disk an`

Mit Select-Object kann ich die Attribute greifen

```
Get-Disk | Select-Object  -Property DiskNumber,BusType,FriendlyName
```

## ().Count

```
(Get-Content irgendwas.txt).count # Zählt alle Objekte
```

## Wert einer Property abrufen

```
Get-ADComputer -Filter * | Select-Object -ExpandProperty Name # Gibt den Wert von Property Name zurück

# Mann kann aber auch über die Methode aufrufen
(Get-ADComputer -Filter*).Name
```

# Methoden

> Info: Erkunde mit Get-Member

Eine Methode auf aufrufen

```
(Get-Date).AddDays(8) # Fügt dem Aktuellen Datum 8 Tage hinzu
```

Dem Unterschied zwischen Attribut und Methode kann man schon während der Eingabe sehen

# Beispiele

```
Get-Disk | Get-Member

# Alles Anzeigen
Get-Disk | Select-Object *

# Gezielt Attribute Abrufen
Get-Disk |
Select-Object -Property `
```

```
DiskNumber,BusType,FriendlyName,NomberOfPartisions,Firm
```

# Array

Ein Array erzeugen (Liste)

```
$array = [array]('Peter', 'Margit')
$array[1] # Gibt
```

# CMD Windows

## Komandozeile

#lernen #programmieren #cmd #shell

## Batchprogrammierung

Eine genau Beschreibung findet sich im Internet

Batch-Dateien erstellen: Befehle für Anfänger und nützliche Expertentipps

## Befehle

| Befehl | Funktion |
| --- | --- |
| `nbtstat -A ip-adresse` | Damit wird der Computername der IP-Adresse angezeigt. (Funktioniert wohl nur im gleichen Subnetz) |
| `ping -4 computername` | Mit diesem Befehl kann der Computername angepingt werden, danach wird die IP-Adresse angezeigt. |
| | |

## Programmbeispiele

### Befehlsausgabe Unterdrücken

```
@echo off
timeout /t 60 > NUL
"C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe"
```

`@echo off` schaltet die Rückgabe in der Shell aus. Timeout wird in diesem Beispiel trotzdem angezeigt. Durch > NUL wird auch das verhindert.

Desweiteren startet diese Batch nach 60 Sekunden den Edge Browser

# Powershell Masterclass

Powershell-Version herausfinden

```
Get-Host
```

Hilfe erhalten

```
Get-Help Get-Command
```

Beschreibungen zu Hilfen

Parameter in [ ] verpflichtend `[-Logname] <String>`
Paramenter komplett in [ ] optional `[-ComputerName <String[]>]`

# Wichtige Befehle (Cmd-lets)

Aufbau: Verb-Nomen Parameter

| **CMD-Let** | **Beschreibung** |
|---|---|
| `Get-Uptime` | Zeigt an, wie lange der Letzte Boot her ist |
| `Get-ComputerInfo` | Zeigt die Computerinfo an |
| `Get-ChildItem` | Verzeichnisinhalt anzeigen |
| `Set-Location` | Aktuellen Ordner festlegen<br>`Set-Location HKLM` setzt die Location auf die Registry |
| `Start-Transcript`<br>`Stop-Transcript` | Zum Loggen verwenden |
| `Read-Host`<br>`Write-Host` | Benutzereingabe abfragen<br>Gibt auf die Konsole aus |
| `Start-Process` | Programm öffnen `Start-Process notepad` |
| `New-Item` | Erstell eine neue Datei<br>`New-Item -ItemType File -Path C: -Force` |
| `Add-Content` | Fügt einer Datei Inhalt hinzu |
| `Get-Content` | Auslesen einer Datei |

| | |
|---|---|
| `Measure-Object` | Zählen<br>gleich wie `.Count` Vorher in Klammern setzen |
| `Measure-Command` | Messen von Befehlen. Wie lange dauert ein Befehl. Damit kann die Zeit gemessen werden.<br>`Measure-Command { Get-ADComputer -Filter 'enabled -eq $false' \| Set-ADComputer -Enabled $true}` |
| `Foreach-Object` | Nimmt jedes Objekt entgegen |
| `Where-Object` | Filtert auf bestimmte Eigenschaften<br>`Get-Process \| Where-Object CPU -GT 10` |
| | |

# Parameter

| **Option** | **Beschreibung** |
|---|---|
| `-Recurse` | Auch für Unterordner |
| `-Confirm` | Fordert zur Bestätigung aus<br>`-Confirm:$false` Damit wird das Bestätigen umgangen |
| `-AsSecureString` | Verschlüsselt abspeichern |
| `-Whatif` | Was wäre wenn. Der Befehl wird nicht ausgeführt |
| `-Verbose` | Was wird passieren |
| `-PassThru` | Gibt nur die Werte wieder, keine Tabellen |
| `-Wrap` | Zeilenumbruch |
| `-AutoSize` | Passt die Spaltenansicht an |

# Alias

`Get-Alias` zeigt alle Aliase an

# Variablen

erstellen `$a`

zu Variablen hinzufügen `$a += 2` Es wird die Variable um 2 erhöht

# Execution Policy

| | |
|---|---|
| Get-ExecutioanPolicy -List | |
| Restricted | Nichts erlaubt |
| Unrestricted | Alles erlaubt |
| RemoteSigned | Aus dem Internet geladene müssen signiert sein |
| AllSigned | Müssen auch die eigenen signiert sein |
| Bypass | |

## Setzen

`Set-ExecutionPolicy Bypass`

## Dateien freigeben zum ausführen

```
Unblock-File
```

# Programmbeispiele

hier zu finden

# Profile erstellen

Darin können verschiedene Einstellungen gespeichert werden, damit Powershell immer mit den gleichen Optionen startet.

```
New-Item $PROFILE -ItemType File -Force
```

# Hilfe lesen können

- Aufrufen mit Get-Help und dem Befehl
- Beispiele sind besser um etwas zu verstehen
  - Das gibt es Online

# Pipeline

Was ist die Pipeline


# Format Befehle

> Format Befehle sollten immer am Ende eines Befehls stehen und diehnen zur Ausgabe auf der Konsole

```
Get-Process | Format-Table Id,ProcessName
```

```
Get-Process | Format-List
```

# Out Befehle

```
Out-File #Ausgabe als Datei
Out-Printer #Ausgabe auf dem Standard-Drucker
Out-Null #Keine Ausgabe
Out-Gridview #Zeigt die Ausgabe als Tabelle an
```

```
Get-Process | Out-File $home\process.txt
```

## Out-Gridview

Mit dem folgenden Skript werden alle AD-User an GridView übergeben. Der Parameter PassThru ermöglicht es eine Auswahl zu treffen. Diese Auswahl wird dann an Disable ADAccount weiter.

```
Get-ADUser -Filter * | Out-GridView -PassThru | Disable-ADAccount -Verbose
```

## Tee-Object

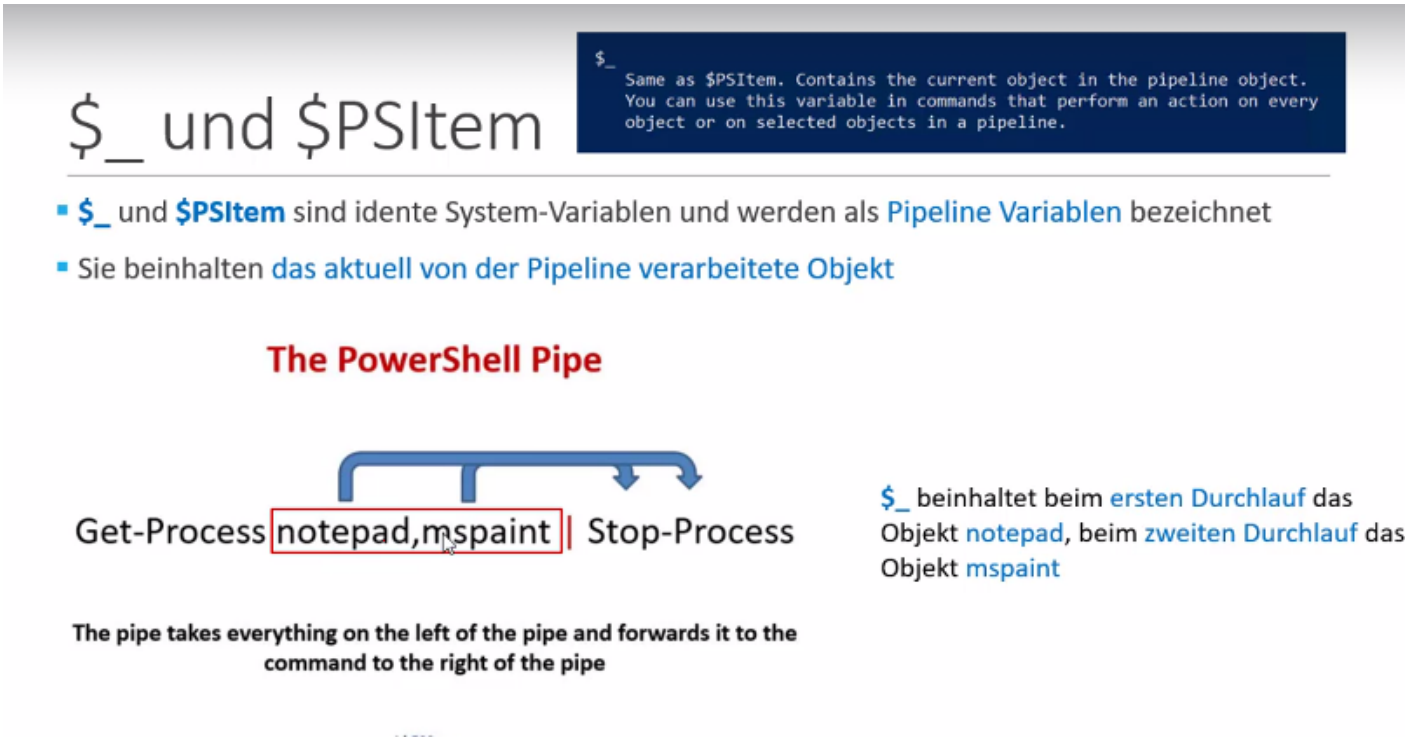
Mit dem CMDlet wird das Ergebnis in der Konsole ausgegeben und als Datei gespeichert.

```
Get-ADUser -Filter * | Tee-Object -FilePath $home\adusers.txt
```

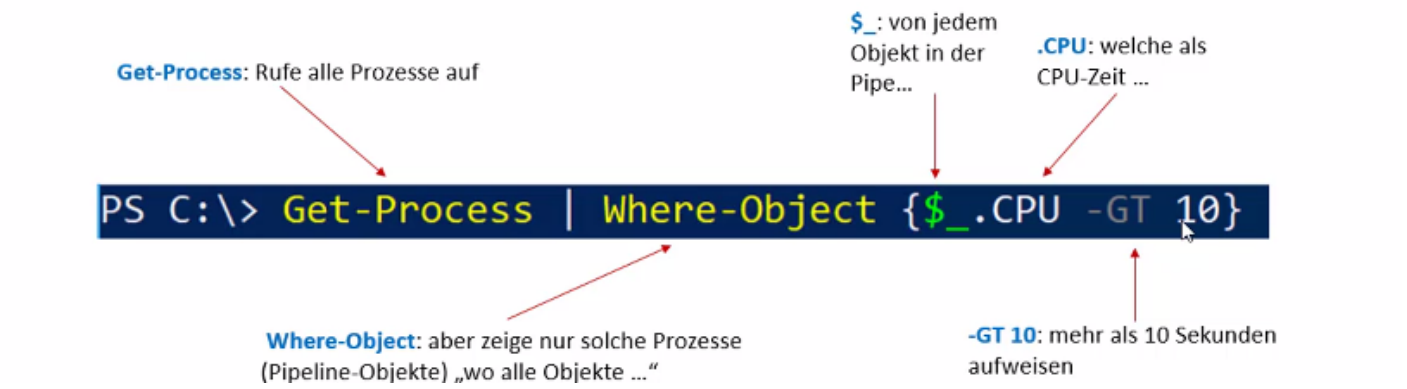
## CSV

```
Get-ADUser -Filter * | Export-Csv $home\aduserscsv.csv


Import-Csv -Path $home\aduserscsv.csv
```

## $_ und $PSItem



Damit wird in der Pipeline jede Variable in den jeweiligen Durchlauf übergeben.

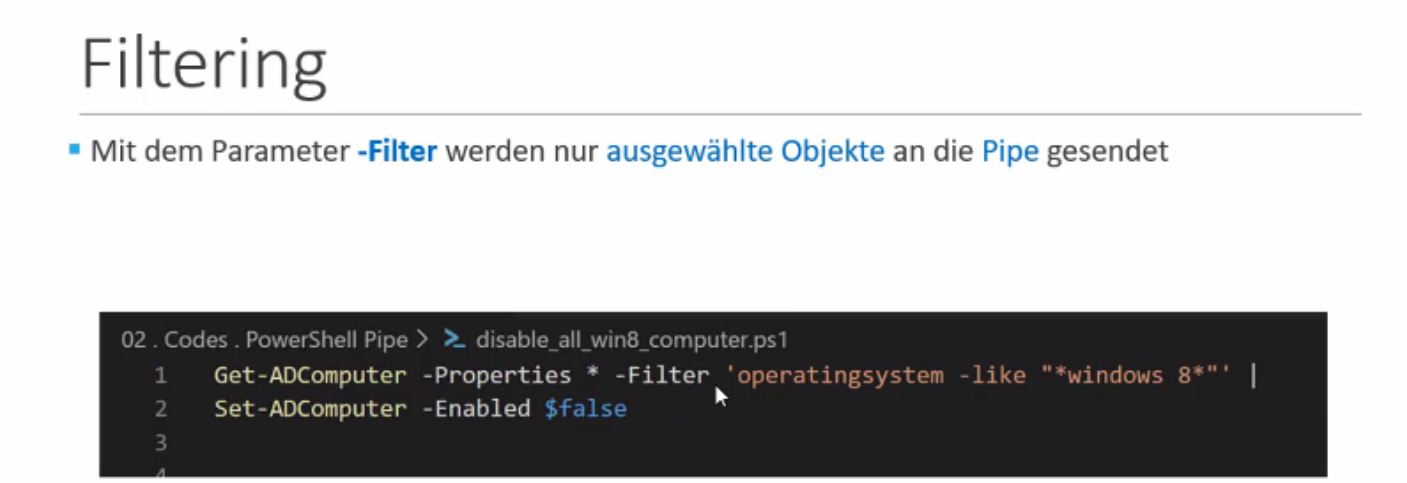



Ein weiteres Beispiel in dem zwei Filter kombiniert werden

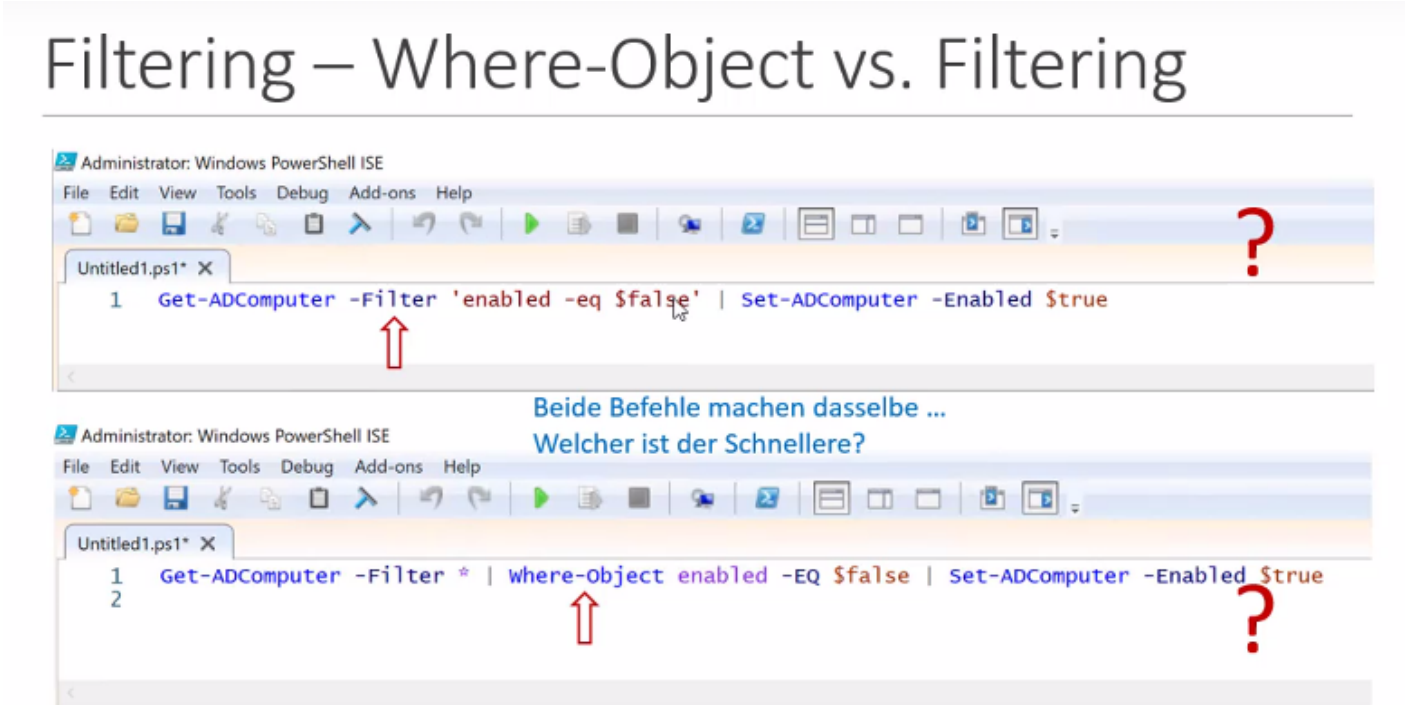
```
Get-Prozess | Where {$_.CPU -gt 50 -AND $_.Handles -gt 1000}
```

# Filtern

Weitere Infos auf der Website




Der Unterschied zwischen den beiden Filtern ist, dass der obere schneller ist, weil der Filter schon vor der Pipeline ausgeführt wird.




# Sort-Object und Select-Object

```
# Sort-Object
Get-Process | Sort-Object CPU -Descending


# Select-Object
Get-Process | Select-Object CPU,Id,ProcessName
```

Select-Object macht etwas ähnliches wie Format-Table nur das die Daten nicht verändert werden. Dadurch kann man mit allen Daten weiter arbeiten.

```
# Gemischt
Get-Process | Sort-Object CPU -Descending | Select-Object ProcessName,CPU -First 3
```

> Schlüsselfunktion in Powershell!

Select-Object mit -ExpandProberty dann wird wirklich nur der Name ausgeben und nicht eine Tabelle die die Namen enthält.

```
$comp = Get-ADComputer -Filter * | Select-Object -ExpandProperty Name
Test-Connection -ComputerName $comp #Text-Connection kann Multible anfragen entgegennehmen und alles in $comp anpingen
```

# Get-Member und Select-Object *

```
# Gibt aus was es alles an Möglichkeiten zu einem CMD-Let gibt
Get-Process | Get-Member


# Hiermit kann man alles alles ausgeben
Get-Process | Select-Object -Property *
```

# Powershell Programmbeispiele

## Programme schließen

```
Get-Process notepad | Stop-Process -Verbose
```

## Edge Browser mit mehreren Seiten öffnen

```
Start-Process msedge -ArgumentList "nc.hhml.selfhost.co www.duckduckgo.com"
```

## Mit einer Verknüpfung ein Powershell-Skript ausführen

```
pwsh.exe -noExit -NoProfile -NoLogo -Command "Get-Eventlog -LogName Aplication -Newest 5"
```

## Vergleichen zweier Dateien

```
Compare-Object -ReferenceObject (Get-Content $home\usernames.txt) -DifferenceObject (Get-Content
$home\usernames_unique.txt)
# in den Klammern wird sofort ausgeführt, bevor die ganze Zeile ausgeführt wird
```

## Aktuellen Speicherpfad des Skripts abrufen

```
$Skriptpfad = Split-Path -Parent $MyInvocation.MyCommand.Path
$input_file = "$($Skriptpfad)\input.txt"
```

## Windows Updates anzeigen

die in den letzten 50 Tagen installiert wurden

```
Get-Hotfix | Where-Object InstalledOn -ge (Get-Date).AddDays(-50)
```

## Öffnen einer Textdatei und das erste Wort in Variable speichern

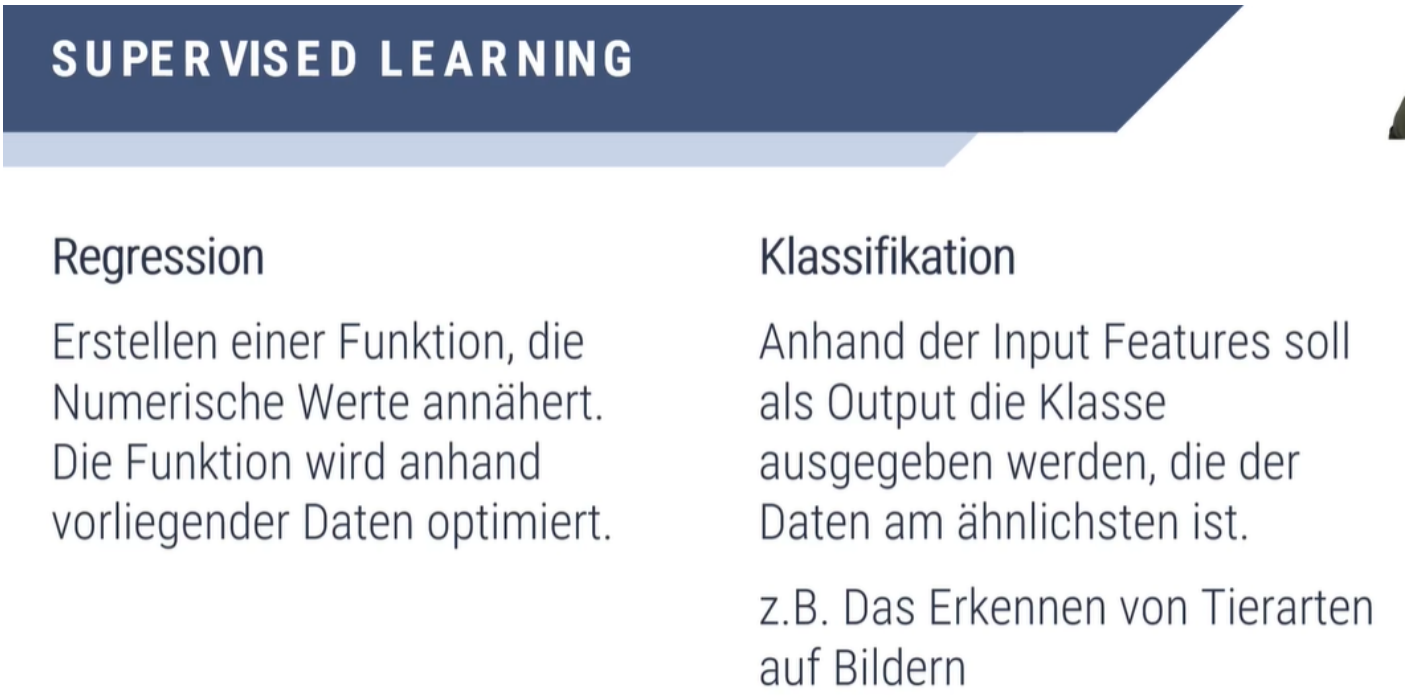
```
$variable = @() # Initialisiere eine leere Variable als Array

Get-Content "C:\Pfad\zur\Datei.txt" | ForEach-Object {
    $firstWord = $_.Split(" ")[0] # Splitte die aktuelle Zeile anhand des Leerzeichens und wähle das erste Wort aus
    $variable += $firstWord # Füge das erste Wort der Zeile dem Array hinzu
}
```

# Updates überprüfen

```
$ComputerName = "RemoteComputerName"
$Date = (Get-Date).AddDays(-50)
Invoke-Command -ComputerName $ComputerName -ScriptBlock {
    Get-CimInstance -ClassName Win32_QuickFixEngineering | Where-Object { $_.InstalledOn -ge $using:Date } | Select-Object HotFixID, InstalledOn
}
```

# Machine Learning (Udemy Kurs)

## Theorie

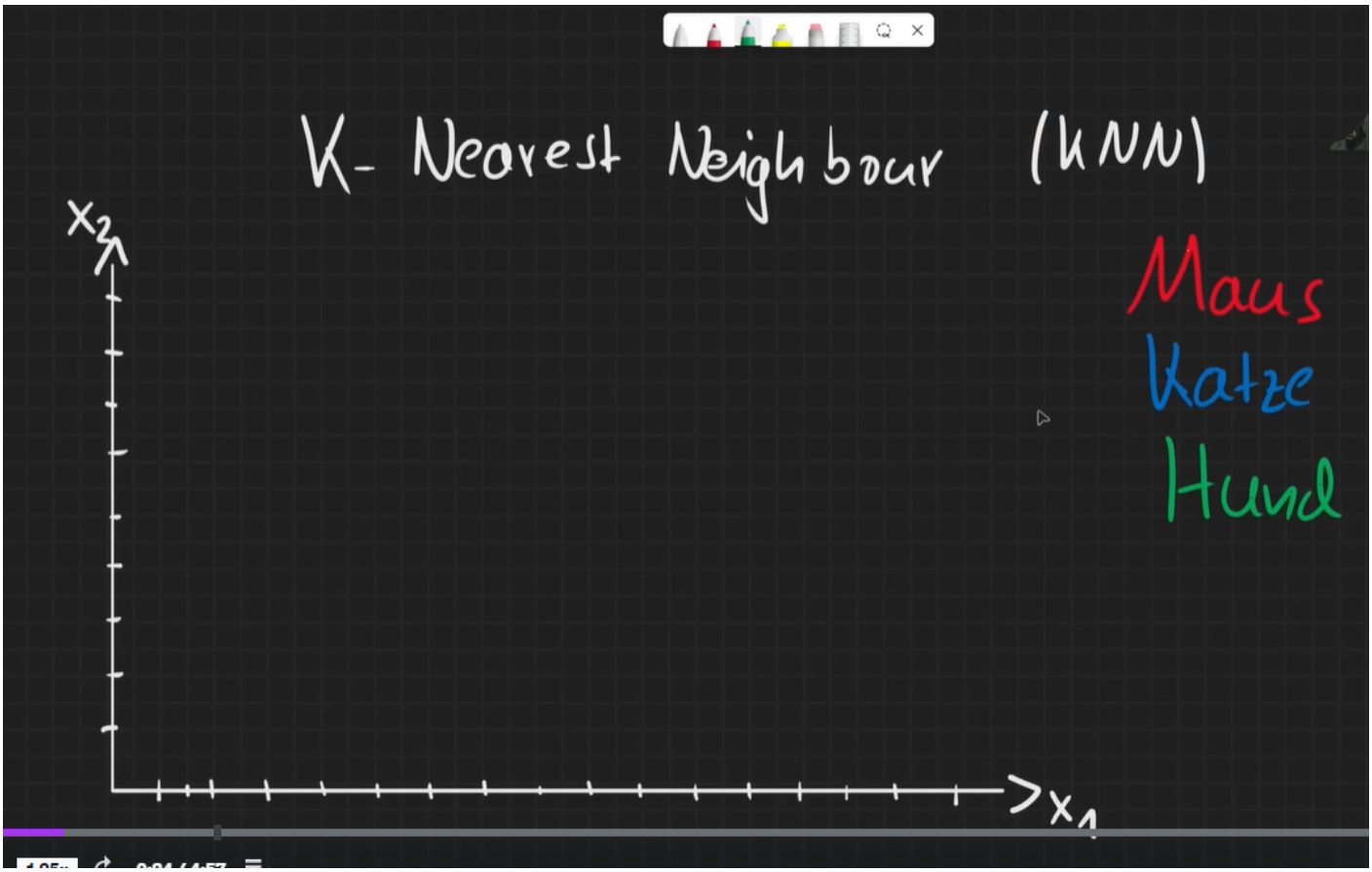


## EIGENSCHAFTEN DES MODELLS

- Das Modell soll anhand der Features die richtigen Aussagen treffen
- Wir „Überwachen“ das Programm beim Training
- Beim Training wissen wir welcher Output zu dem jeweiligen Input richtig ist
- Somit sind wir der Lehrer des Modells, der auf Fehler hinweist

## DER LERNPROZESS

- Wenn ein Fehler beim Training auftritt, muss erforscht werden warum er auftrat
- Das Programm soll dann nach Anpassung der Parameter ein besseres Ergebnis liefern
- Das Ziel ist dann anhand des Trainings ein allgemeingültiges System zu haben




# Praxis

## Module

### Numpy

```
import numpy as np
```

Ein Array mit Numpy erstellen

```
x = np.array([-2, 1, 3, -10, 22])
```

### Matplotlib

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

Einen Plot erstellen

```
x = np.array([1,2,3,4,5])
y = np.array([2,-3,5,10,12])


# Scatter Plot # Punkte ohne Verbindung


plt.scatter(x, y, color="red")
plt.legend(['f(x)'])
plt.title('This is a title')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.show()
```

Mit zeilen verbinden

```
# Plot


plt.scatter(x, y, color="red")
plt.plot(x,y,color="blue")
plt.legend(['f(x)'])
plt.title('This is a title')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.show()
```

# Listslicing und Arrays

```
import numpy as np


l1 = [i for i in range(20)] # Erstellt eine Liste mit Werten von 0 bis 19


l2 = l1[0:20:2] # Erstellt eine Liste aus der ersten Liste und zwar nur jeden 2. Wert


l3 = l1[:10] # Gibt die ersten 10 wieder


# Arrays mit Numpy erstellen
my_array = np.zeros(shape=(2,2)) # erstellt ein array mit 2 x 2 Zellen


my_rershaped_array = np.reshape(my_array, newshape=(4,))
```

```
# Zufallszahlen generieren
my_random_array = np.random.randint(low=0, high=11, size=20) # Ganzzahlen mit Randint

my_random_array2 = np.random.uniform(low=0.0, high=10.0, size=20) # Floatzahlen erstellen
```

# Type Annotations

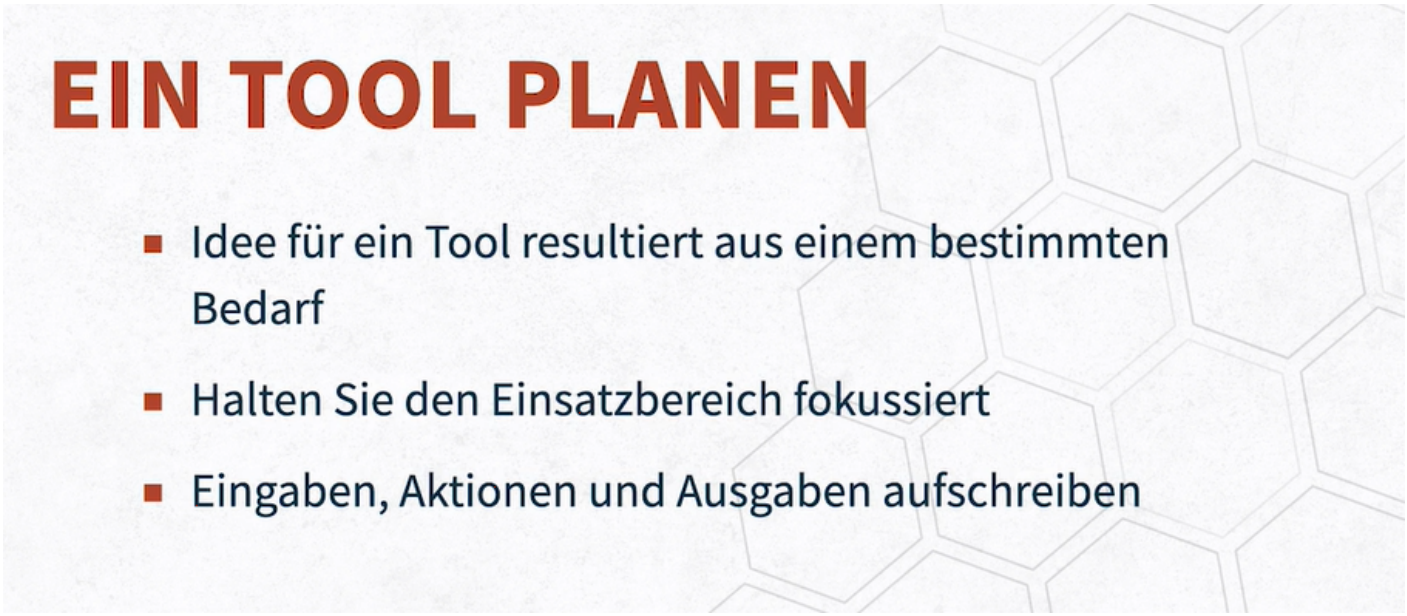
Festlegen welche Dateitypen für einen Parameter infrage kommen

```
class Car:
  def __init__(self, name: str, oem: str, hp: int, year: int) -> None: # -> None bedeutet es wird nichts zurück
gegeben
    self.name = name
    self.oem = oem
    self.hp = hp
    self.year = year

  def get_info(self) -> str: # -> str bedeutet es wird ein String zurück gegeben
    return "Name: " # self.name

def main():
  car1 = Car(10, "Audi", 400, 2022)
  info1 = car1.get_info()
  print(info1)



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

# f-Strings

```
# Mit f-string kann man die Variable direkt in den String schreiben

f"Name: {self.name} OEM: {self.oem} HP: {self.hp}"
```

# Tools mit Python entwickeln



# Checksumme vergleichen

# Checksum vergleichen

> mit Hilfe von Powershell

# Checksum / Prüfsumme berechnen / ermitteln

Anleitung für Windows und Linux

Hier erfährst du was Checksum bedeutet und wie du unter Windows und unter Linux (Ubuntu/Lubuntu) den Checksum einfach ermittelst. Mit einem Vergleich der Checksummen prüfst du eine heruntergeladene Datei auf Integrität, d.h. du stellst damit sicher, dass du die Originaldatei erhalten hast.

- Kurzinfo 1: Was ist Checksum / Prüfsumme?
- Kurzinfo 2: Wozu ist Checksum nützlich?
- Windows 10 Möglichkeit 1: Checksum mittels Powershell unter Windows 10 ermitteln
- Windows 10 Möglichkeit 2: Checksum mittels 7-Zip Kontextmenü CRC SHA unter Windows 10 ermitteln
- Ubuntu / Lubuntu Möglichkeit 1: Checksum mittels Terminal unter Linux Ubuntu-Lubuntu verifizieren

## Was ist Checksum / Prüfsumme?

Ein Checksum (Prüfsumme) ist ein digitaler Fingerabdruck einer Datei. Genau wie ein Fingerabdruck einer Person einzigartig ist, ist auch der digitale Fingerabdruck, also die Checksum einer Datei, einzigartig. Solange der Inhalt einer Datei nicht verändert wurde, bleibt der Fingerabdruck (die Checksum) jeder Datei auch unverändert. Wird jedoch der Inhalt (nicht der

Dateiname) einer Datei auch nur minimal verändert, so ändert sich auch der digitale Fingerabdruck diese Datei.

# **Wozu** ist das nützlich?

Wenn ich eine Datei erstelle und dir per E-Mail, Download etc. zur Verfügung stelle, dann kann es sein, dass diese Datei auf dem Weg zu dir manipuliert/verändert wurde. Und hier hilft die Checksum bzw. die Prüfsumme. Ich ermittle aus der Datei die ich dir zusenden möchte die Checksum und teile dir diese Checksum mit.

Und sobald du die Datei empfangen hast, ermittelst du auf deinem PC zu der Datei auch die Checksum und vergleichst die beiden Checksumen miteinander. Sind sie identisch, so wurde die Datei nicht verändert. Sind diese Checksumen nicht identisch, so wurde der Inhalt dieser Datei auf dem Weg zu dir verändert.

Natürlich ist die Sicherstellung der korrekten Checksum bei unwichtigen Dateien wie Bilder und mp3 Musik nicht wirklich notwendig. Solange das Bild der Erwartung entspricht und die Musik das gewünscchte Lied abspielt, ist es völlig egal, ob die Checksumen identisch sind oder nicht.

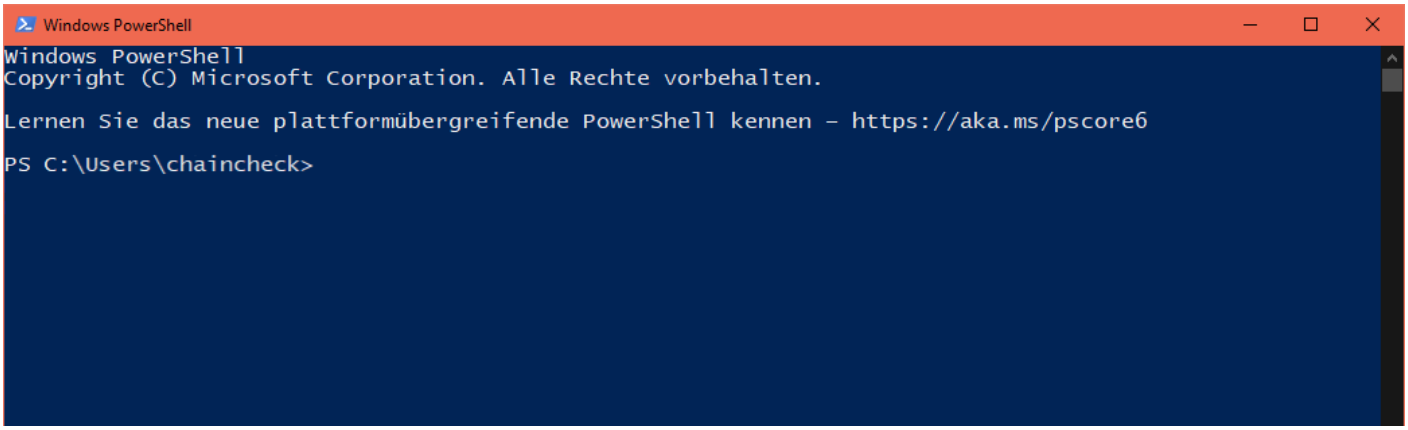
Das sieht jedoch bei wichtigen Programmen wie zB. bei Krypto-Brieftaschen (Wallets) ganz anders aus. Wenn jemanden z.B. 5,5 Bitcoin **(aktueller Wert: 103.450,71 EUR**), 50 Litecoin **(aktueller Wert: 2.618,08 EUR**) oder 100 Monero **(aktueller Wert: 14.038,05 EUR**) auf seiner Krypto-Adresse besitzt, der sollte genauer Prüfen ob das heruntergeladene Wallet-Programm tatsächlich auf dem Downloadweg nicht verändert / manipuliert wurde. Kurz gesagt, wichtige Programme/Dateien sollten immer auf Integrität geprüft werden.

Wie wichtig es ist eine Wallet-Datei vor der Installation über den Hashwert auf Integrität zu prüfen, zeigt der am 19.11.2019 bekannt gewordener Fall der Mondero cli Wallet. Wer zwischen dem 19.11. und dem 20.11.2019 die Monero cli Wallet heruntergeladen und ohne zu prüfen auch installiert hat, hat sich eine Wallet mit Malware installiert. Diese Wallet hat die Seed entwendet, was dazu führte, dass die Moneros für immer weg waren.

# Checksum unter Windows 10 ermitteln

Mittels Powershell / Eingabeaufforderung (cmd)

Als erstes öffne die Powershell. Drücke dazu gleichzeitig die Tastenkombination **[Windowstaste] + [R]**. Es öffnet sich ein kleines Fenster. Gebe hier "powershell" (ohne Anführungszeichen) ein und drücke Enter.
*In dieser Anleitung wird Powershell verwendet. Das selbe funktioniert auch mit der Eingabeaufforderung "cmd".*

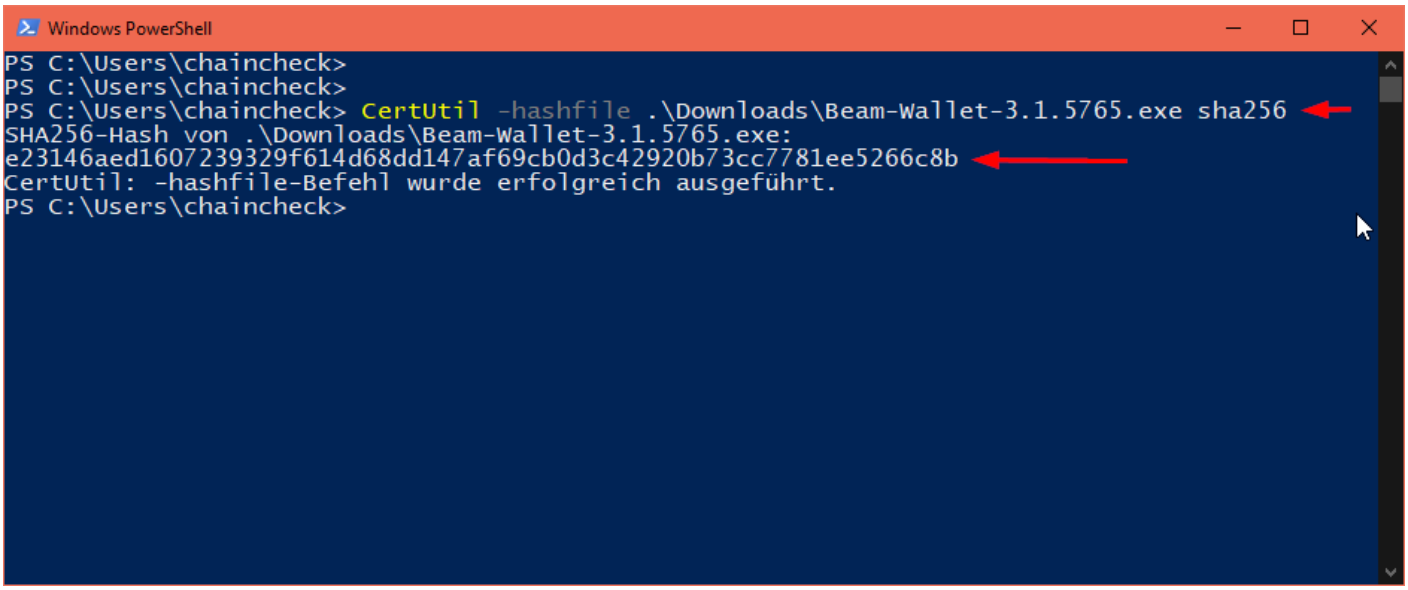



In diesem Beispiel wollen wir die Checksum der Datei "Beam-Wallet-3.1.5765.exe" ermittelt. Diese Datei befindet sich im Downloads-Ordner.

Hier nutzen wir den Hash-Algorithmus sha256, da die Checksum der Datei auf der Beam-Webseite auch mit diesen Hash-Algorithmus erstellt wurde. Entsprechend geben wir in der Powershell folgendes ein (ohne $ Zeichen) und drücken Enter:

```
$ CertUtil -hashfile .\Downloads\Beam-Wallet-3.1.5765.exe sha256
```

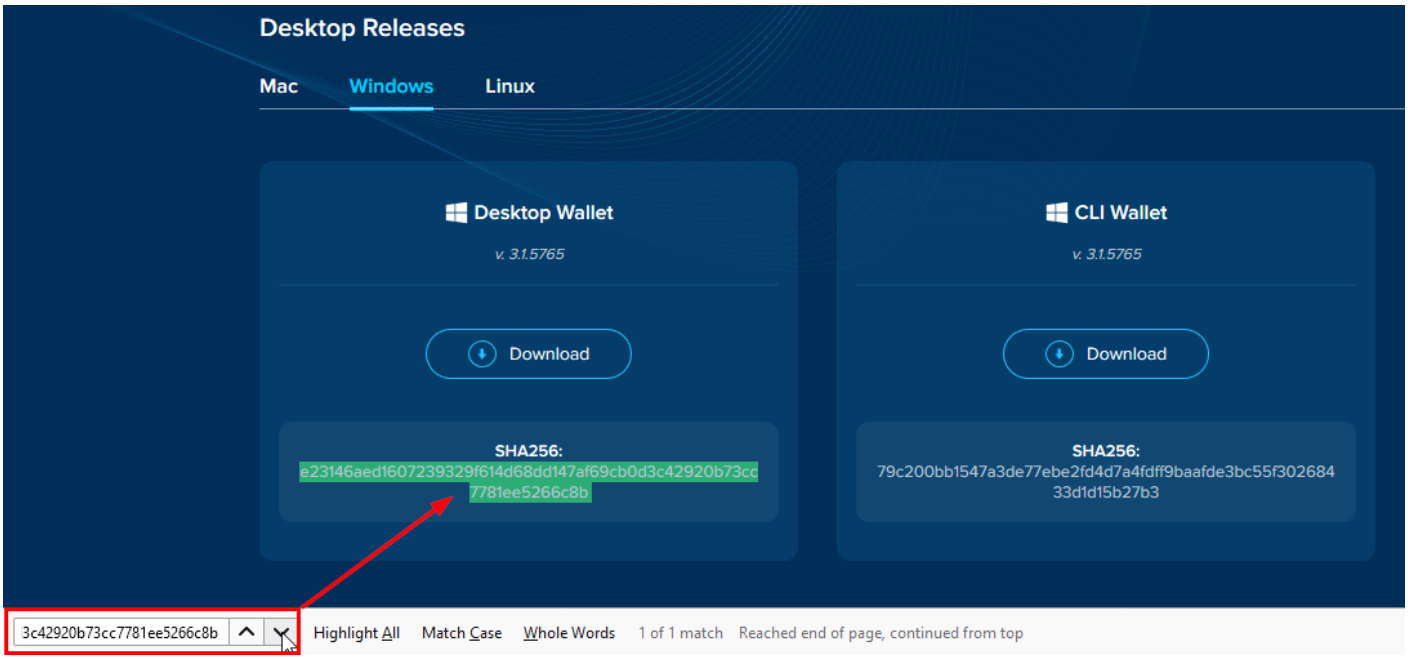
Es erscheint der sha256 Checksum der Datei Beam-Wallet-3.1.5765.exe. In diesem Fall "e23146aed1607239329f614d68dd147af69cb0d3c42920b73cc7781ee5266c8b". Mit der Maus kann diese Checksum markiert und durch drücken der Enter Taste in die Zwischenablage kopieren werden.




Jetzt muss nur noch geprüft werden, ob die auf dem eigenem Rechner ausgegebene Checksum die selbe ist, wie auf der Download-Seite. Kopiere den Checksum und gehe dazu auf die Download-

Seite. Drücke dort *STRG+F* und dann *STRG+V* um nach dem Zeichensatz in der Zwischenablage zu suchen.

Ist die Checksum identisch, dann kann davon ausgegangen werden, dass die heruntergeladene Datei die Originaldatei ist.




# Checksum unter Windows 10

Mittels 7-zip Kontextmenü CRC SHA ermitteln

Wer das Datenkomprimierungs-Programm 7-Zip unter Windows installiert hat, der kann auch über das Windows Kontextmenü schnell die Checksum einer Datei überprüfen.

Dazu einfach die Datei mit der rechten Maustaste anklicken. Es öffnen sich das Kontaxtmenü in diesem der Unterpunkt "CRC SHA" erscheint. Hier muss nur noch der gewünschte Checksum entsprechend ausgewählt werden:

Als Ergebnis erhalten wir den Checksum zu dieser Datei. Dieser Zeichensatz ist leider nicht kopierbar.

# Checksum unter Linux (Ubuntu/Lubuntu) ermitteln

Mittels Linux Terminal

Um den Checksum Unter Linux (Ubuntu/Lubuntu usw.) zu ermitteln, wird der gesuchte Hash-Algorithmus einfach vor der Datei eingegeben.

Möchten wir z.B. für die Datei "Beam-Wallet-3.1.5765.exe" den Hash-Algorithmus **sha256** ermitteln, so müssen wir im Ubuntu/Lubuntu Terminal folgendes eingeben:

```
$ sha256sum Beam-Wallet-3.1.5765.exe
```

Und für **MD5** geben wir folgendes ein:

```
$ md5sum Beam-Wallet-3.1.5765.exe
```

usw.

```
$ sha1sum Beam-Wallet-3.1.5765.exe
```

Bei crc32 dann wieder ohne sum anzuhängen:

```
$ crc32 Beam-Wallet-3.1.5765.exe
```

Das sind die Möglichkeiten die Checksum bzw. die Prüfsumme zu ermitteln.

Fehlt eine eventuell noch bessere oder einfachere Möglichkeit die Checksum zu ermitteln? Dann schreibe mir bitte, damit ich dise Anleitung anpassen kann.

# Bash Skripting

# Benutzereingabe

```
#!/bin/bash

# Root oder nicht root?
if [[ $UID -ne 0 ]]
then
  echo "Das Skript läuft nicht mit Root-Rechten"
fi

# Den Login-Namen abfragen
read -p 'Bitte den Login-Namen eingeben: ' USERNAME

# Den vollständigen Namen abfragen
read -p 'Bitte den vollständigen Namen eingeben: ' COMMENT

# Das Passwort abfragen
read -p 'Bitte Das Paswort eingeben: ' PASSWORD

# Den Benutzer anlegen
useradd -m -c "$COMMENT" -s /bin/zsh $USERNAME

# Das Passwort setzen
echo "$SUERNAME:$PASSWORD | chpasswd # chpasswd muss verwendet werden, da passwd nicht über ein skript funktioniert
```

# Usereingabe über Parameter

```
#!/bin/bash

# Root oder nicht root?
if [[ $UID -ne 0 ]]
```

```
then
  echo "Das Skript läuft nicht mit Root-Rechten"
  exit 1
fi

# Drei Parameter?
if [[ $# -ne 3 ]];then
  echo "Syntax: Adding-user-parm.sh USERNAME \"REAL NAME\" PASSWORD"
  exit 1
fi

USERNAME="$1"
COMMENT="$2"
PASSWORD="$3"

# Den Login-Namen abfragen
#read -p 'Bitte den Login-Namen eingeben: ' USERNAME

# Den vollständigen Namen abfragen
#read -p 'Bitte den vollständigen Namen eingeben: ' COMMENT

# Das Passwort abfragen
#read -p 'Bitte Das Paswort eingeben: ' PASSWORD

# Den Benutzer anlegen
useradd -m -c "$COMMENT" -s /bin/zsh $USERNAME

# Das Passwort setzen
echo "$SUERNAME:$PASSWORD | chpasswd # chpasswd muss verwendet werden, da passwd nicht über ein
skript funktioniert
```

# Parameter übergeben

```
#!/bin/bash

# Parameter prüfen
if [[ $# -lt 1 ]];then
  echo "Syntax: parameter.sh USERNAME [USERNAME2]"
fi

# Parameter auslesen
echo "Name des Sktips: $(basename $0)" # mit $(basename) nur den Namen des Skripts anzeigen lassen
$(dirname) würde den Pfad anzeigen
echo "Erster Parameter: $1"
echo "Zweiter Parameter: $2"
echo "Alle Parameter: $@"
echo "Du hast $# Parameter übergeben"
```

# Zufall Random

```
$RANDOM

#Datum in Sekunden
date +%s

#Datum in Nanosekunden und Random wert
echo "$(date +%s%N)$RANDOM"

# Hashwert erstellen
echo "$(date +%s%N)$RANDOM" | sha512sum

# Reduzieren auf die ersten 8 Zeichen
echo "$(date +%s%N)$RANDOM" | sha512sum | head -c8
```

# For-Schleife mit Zufallspasswort Random und While-Schleife

```
#!/bin/bash

# Wert in einer For-Schleife verarbeiten
for USER in $@
do
  useradd -m $USER
  PASSWORD=$(echo "$(date +%s%N$RANDOM)" | sha512sum | head -c8)
  echo "$USER:$PASSWORD" | chpasswd
  passwd -e $USER # Dadurch muss der User das Passwort bei der ersten Anmeldung ändern
  echo "User: $USER - Passwort: $PASSWORD"
done
```

# Parameter aus einer Datei auslesen

```
#!/bin/bash

# Wert in einer For-Schleife verarbeiten
for USER in $(cat /home/eric/bin/user.txt)
do
  useradd -m $USER
  PASSWORD=$(echo "$(date +%s%N$RANDOM)" | sha512sum | head -c8)
  echo "$USER:$PASSWORD" | chpasswd
```

```
  passwd -e $USER # Dadurch muss der User das Passwort bei der ersten Anmeldung ändern
  echo "User: $USER - Passwort: $PASSWORD"
done
```

# Mit Laufvariable

```
#!/bin/bash

for (( i=1; 1<11; i++ )); do
  echo $i
done

echo "Die schleife wurde beendet"
```

# While Schleife

```
#!/bin/bash

# While-Schleife PoC

WEITER="j"

while [[ "$WEITER" 0 "j" ]]; do
  echo "While-Schleife wird ausgeführt"
  read -p "Weiter? j/n " WEITER
done
echo "Schleife beendet"
```

# Case - Kontrollstruktur

## Beispiel mit if

```
#!/bin/bash

# Programmverzweigung mit
if [[ "#$" -ne 2 ]]; then
  echo "Syntax: case.sh AKTION DIENST"
  exit 1
fi

if [[ "$1" = "start" ]];then
  echo "$2 wird gestartet"
  exit 0
fi

elif [[ "$1" = "stop" ]];then
  echo "$2 wird gestopt"
  exit 0
fi
```

## Umsetzung mit Case

```
#!/bin/bash

if [[ "#$" -ne 2 ]]; then
  echo "Syntax: case.sh AKTION DIENST"
  exit 1
fi

# Verzweigung mit case
```

```
case $1 in
  start)
    echo "$2 wird gestartet"
    ;;
  stop)
    echo "$2 wird gestopt"
    ;;
  restart)
    echo "$2 wird neu gestartet"
    ;;
  status|state) # mit der pipe können wir mehrere möglichkeiten bieten diesen teil aufzurufen
    echo "Der Satus von $2 wird angezeigt"
    ;;
  *)
    echo "Der Parameter ist unbekannt"
    exit 1
    ;;
esac
```

# For-Schleife

```
#!/bin/bash

echo "Die übergebenen Usernamen sind: $@"

# Alle Parameter (User) ausgeben
for USER in $@
do
  echo "User im Aktuellen For-Schleifen-Durchlauf: $USER"
done
```

# Datei per Skipt verschlüsseln

anschließend eine Datei mit `ccrypt` verschlüsselt:

```
#!/bin/bash

# Datei, die verschlüsselt werden soll
read -p "Bitte den Dateinamen eingeben, der verschlüsselt werden soll: " datei

# Passwort verdeckt eingeben
echo -n "Bitte das Passwort eingeben: "
read -s passwort
echo

# Datei mit ccrypt verschlüsseln
echo "$passwort" | ccrypt -e -k - "$datei"

# Überprüfen, ob die Verschlüsselung erfolgreich war
if [ $? -eq 0 ]; then
    echo "Die Datei wurde erfolgreich verschlüsselt."
else
    echo "Fehler bei der Verschlüsselung der Datei."
fi
```

Speichere dieses Skript in einer Datei, z.B. `encrypt.sh`, und mache es ausführbar mit:

```
chmod +x encrypt.sh
```

Dann kannst du es ausführen mit:

```
./encrypt.sh
```

Dieses Skript fordert den Benutzer auf, den Dateinamen und das Passwort einzugeben. Das Passwort wird verdeckt eingegeben (d.h. es wird nicht auf dem Bildschirm angezeigt). Anschließend wird die Datei mit `ccrypt` verschlüsselt.

# Shell programmieren

# Flutter

# Flutter Programmierung

Wichtige Dart Befehle für Flutter

Importieren von Flutter in Dart

```
import 'package:flutter/material.dart';
```

Einstiegspunkt

```
void main () {
  runApp(const MyApp());
}
```

Klasse App

```
class MyApp extends StatelessWidget {
  const MyApp({super.key});



  @override
  Widget build(BuildContext context) {
    return MaterialApp();
  }
}
```

Scaffold

Ist eine "leere Leinwand". Das Grundgerüst für die Ansicht

Man kann sich ein vordefiniertes StatelessWidget erstellen lassen wenn man st eingibt und dann auswählt

Pubspec.yaml Datei anpassen um Bilder importieren zu können.

# Dart

Dart

# Grundlegendes zu Flutter

# □□ Async/Await - Vollständige Erklärung

# Das Problem: Warum ist Dateiarbeit langsam?

Stell dir vor, du machst eine Pizza:

text

```
SYNCHRON (blockierend - FALSCH für Dateien):

┌─────────────────────────────────────────┐
│ 1. Pizza in den Ofen (15 Min)           │
│    ← Der Pizzabäcker WARTET hier!       │
│    ← Kann nichts anderes machen!        │
│                                         │
│ 2. Pizza rausnehmen                     │
│ 3. Nächste Pizza machen                 │
└─────────────────────────────────────────┘
```

Resultat: Sehr ineffizient!

text

```
ASYNCHRON (nicht-blockierend - RICHTIG für Dateien):

┌─────────────────────────────────────────┐
│ 1. Pizza in den Ofen (15 Min)           │
│    ← Der Pizzabäcker WARTET NICHT!      │
│                                         │
│ 2. Macht währenddessen Pizza 2 (parallel)
│ 3. Macht Pizza 3                        │
│ 4. Bekommt Signal: "Pizza 1 ist fertig!"
│ 5. Holt Pizza 1 raus                    │
└─────────────────────────────────────────┘
```

Resultat: Sehr effizient!

---

# Code-Beispiel: Sync vs Async

## ❌ SYNCHRON (Falsch für Dateien):

dart

```
void main() {
  print('Start');

  // Annahme: Datei braucht 2 Sekunden
  final file = File('data/users.json');
  final content = file.readAsStringSync();  // ← BLOCKIERT!

  print('Datei gelesen: $content');
  print('Fertig');
}

// Output:
// Start
// [warte 2 Sekunden... nichts läuft!]
// Datei gelesen: ...
// Fertig

// Die App war 2 Sekunden "eingefroren"!
```

**Problem:** Während die App wartet, kann sie NICHTS anderes machen!

---

## ✅ ASYNCHRON (Richtig für Dateien):

dart

```
void main() async {  // ← main() wird async!
  print('Start');

  final file = File('data/users.json');
```

```
  final content = await file.readAsString();  // ← BLOCKIERT NICHT!

  print('Datei gelesen: $content');
  print('Fertig');
}

// Output:
// Start
// [warte 2 Sekunden... aber die App läuft weiter!]
// Datei gelesen: ...
// Fertig

// Die App war NICHT eingefroren!
```

**Vorteil:** Die App kann während des Wartens andere Dinge machen!

---

# □□ Was bedeuten die Keywords?

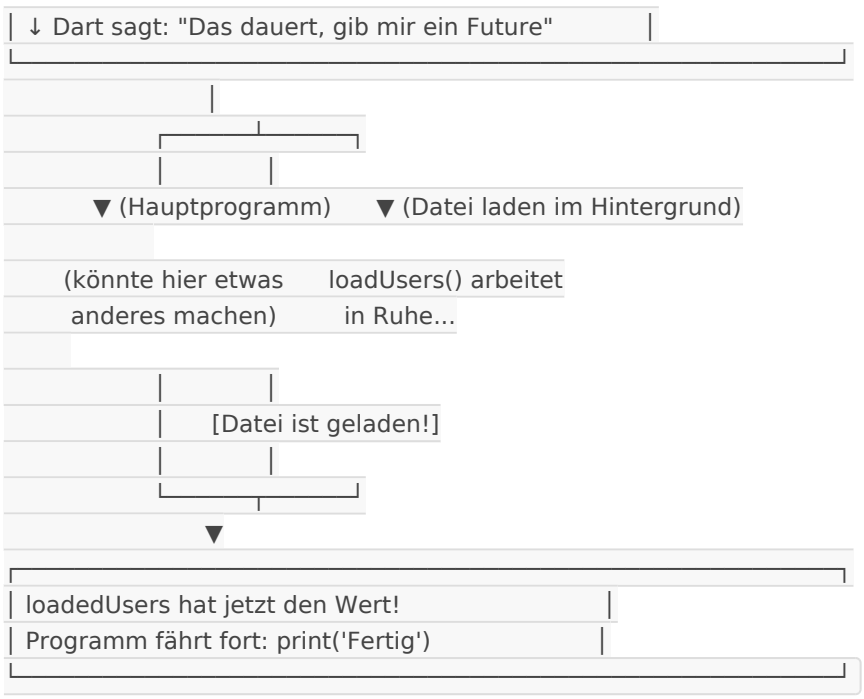
| Keyword | Bedeutung | Beispiel |
|---|---|---|
| `async` | "Diese Methode ist langsam, kann warten" | `Future<String> getData() async { ... }` |
| `await` | "Warte hier, bis Ergebnis kommt" | `final data = await loadFile();` |
| `Future<T>` | "Ein Ergebnis vom Typ T kommt später" | `Future<int> getAge() { ... }` |

---

# □□ Visualisierung: Der Fluss von Async/Await

text

```
┌──────────────────────────────────────┐
│ Code ausführen                       │
└──────────────────────────────────────┘
                  │
                  ▼
┌──────────────────────────────────────┐
│ await StorageService.loadUsers()     │
```




---

# □□ Praktisches Beispiel: Async testen

Erstelle `bin/async_demo.dart`:

dart

```dart
import 'dart:io';

// Beispiel 1: Synchrone Funktion (NICHT async)
void syncFunction() {
  print('Sync Start');
  sleep(Duration(seconds: 2));  // Blockiert die App!
  print('Sync Ende (nach 2 Sekunden)');
}

// Beispiel 2: Asynchrone Funktion (mit async/await)
Future<void> asyncFunction() async {
  print('Async Start');
  await Future.delayed(Duration(seconds: 2));  // Blockiert NICHT!
  print('Async Ende (nach 2 Sekunden)');
}

// Beispiel 3: Mehrere Async-Funktionen (parallel!)
Future<void> multipleAsync() async {
  print('\n=== Mehrere Async-Tasks parallel ===');

  // Alle 3 starten gleichzeitig!
```

```
  final task1 = doTask('Task 1', 1);
  final task2 = doTask('Task 2', 2);
  final task3 = doTask('Task 3', 3);

  // Warte auf alle
  await Future.wait([task1, task2, task3]);

  print('Alle Tasks fertig!');
}

Future<void> doTask(String name, int seconds) async {
  print('$name startet...');
  await Future.delayed(Duration(seconds: seconds));
  print('$name fertig!');
}

void main() async {
  print('=== SYNCHRON (blockierend) ===');
  syncFunction();  // Freezt die App!

  print('\n=== ASYNCHRON (nicht-blockierend) ===');
  await asyncFunction();  // App läuft weiter!

  await multipleAsync();
}
```

**Starte das Programm:**

bash

```
dart bin/async_demo.dart
```

**Erwartete Ausgabe:**

text

```
=== SYNCHRON (blockierend) ===
Sync Start
Sync Ende (nach 2 Sekunden)

=== ASYNCHRON (nicht-blockierend) ===
Async Start
Async Ende (nach 2 Sekunden)

=== Mehrere Async-Tasks parallel ===
Task 1 startet...
Task 2 startet...
Task 3 startet...
Task 3 fertig!
Task 2 fertig!
Task 1 fertig!
Alle Tasks fertig!
```

**Beobachtung:** Task 3 endet zuerst (nur 1 Sekunde), obwohl sie nicht zuerst startete! Das zeigt, dass sie **parallel** laufen! □□

# □□ Die 3 Zustände eines Future

Ein `Future` hat immer einen von 3 Zuständen:

text

```
┌─────────────────────────────────────┐
│        Future Status                │
├─────────────────────────────────────┤
│                                     │
│  1. PENDING (wartet noch)           │
│     await loadUsers()               │
│     ↓ (Datei wird gelesen...)       │
│                                     │
│  2. COMPLETED (fertig!)             │
│     ↓ (Datei komplett gelesen)      │
│     loadedUsers = [User(...), ...]  │
│                                     │
│  3. ERROR (Fehler!)                 │
│     ↓ (Datei nicht gefunden)        │
│     catch (e) { print(e); }         │
│                                     │
└─────────────────────────────────────┘
```

---

# ⚠ Wichtige Regeln:

**Regel 1:** Du kannst **nur `await` innerhalb von `async` Funktionen** nutzen!

dart

```
// □  RICHTIG:
Future<void> myFunction() async {
  await Future.delayed(Duration(seconds: 1));
}

// □  FALSCH:
void myFunction() {
  await Future.delayed(Duration(seconds: 1));  // ERROR!
}
```

**Regel 2:** Wenn du `await` brauchst, muss die Funktion `async` sein!

dart

```
// □  RICHTIG:
void main() async {  // ← main() wird async!
```

```
  await StorageService.loadUsers();
}

// □  FALSCH:
void main() {
  await StorageService.loadUsers();  // ERROR!
}
```

**Regel 3:** `async` macht deine Funktion automatisch zu einem `Future`!

```
Future<String> getData() async { // ← Gibt automatisch Future<String> zurück!
  return "Daten"; } // Ist gleichbedeutend mit:

Future<String> getData2() { // ← Musst du manuell Future sagen
  return Future.value("Daten"); }
```

---

# □□ Kurze Analogie:

```
Synchron:
Du: "Barista, mach mir einen Kaffee"
Barista: "Ja, warte hier... [5 Minuten warten]... Fertig!"
Du: "Okay, danke!" ← Du konntest nur herumsitzen!

Asynchron:
Du: "Barista, mach mir einen Kaffee" (mit Future-Ticket)
Barista: "Ja, kein Problem, ich ruf dich auf!"
Du: "Gut, ich setz mich hin und lies Zeitung" ← Du machst etwas anderes! [5 Minuten später]
Barista: "Dein Kaffee ist fertig!" (Future resolved!)
Du: "Danke!" ← Du warst nicht blockiert!
```

# Methoden und Klassen

Eine typische Funktion in Dart besteht aus dem Rückgabewert, dem Namen, optionalen Parametern und dem Funktionskörper.

# Aufbau einer Funktion

dart
Copy code

```
Rueckgabewert funktionsName(Parameter) {
  // Funktionskörper
  return wert; // nur wenn Rueckgabewert nicht void ist
}
```

Beispiel:

dart
Copy code

```
int addiere(int a, int b) {
  return a + b;
}
```

# Aufbau einer Klasse

Eine Klasse bündelt Werte (Felder), einen Konstruktor und Methoden.

dart
Copy code

```
class Auto {
  // Felder
  final String marke;
  int kilometerstand;

  // Konstruktor
  Auto(this.marke, this.kilometerstand);

  // Methode
  void fahre(int kilometer) {
    kilometerstand += kilometer;
  }
```

```
  // Eine Methode mit Rückgabewert
  String beschreibung() {
    return 'Marke: $marke, Kilometerstand: $kilometerstand';
  }
}
```

# Nutzung der Klasse

dart
Copy code

```
void main() {
  final auto = Auto('Tesla', 0);   // Konstruktor
  auto.fahre(120);                 // Methode
  print(auto.beschreibung());      // Methode mit Rückgabewert
}
```

So siehst du den typischen Aufbau: Felder speichern den Zustand, der Konstruktor initialisiert ihn, und Methoden ändern oder lesen ihn aus.

# Dart

Bei einem Dart Programm Argument auf Konsolenebene mitgeben:

```
void main(List<String> arguments) {
  print('Hello world: ${notepad_app.calculate()}!');
}
```

arguments ist eine liste die Argumente entgegennimmt von außen.

Das bedeutet konkret:

- Wenn du dein Dart-Programm in der Konsole z.B. startest mit dart run meinprogramm.dart arg1 arg2 arg3, dann ist arguments eine Liste mit den Strings ["arg1", "arg2", "arg3"].
- Du kannst diese Argumente nutzen, um das Verhalten deines Programms dynamisch zu steuern, z.B. verschiedene Modi, Dateinamen oder Einstellungen von außen zu übergeben.
- Wenn du aber keine Argumente übergibst, ist die Liste einfach leer.

# Flutter, Dart App Programmierung

# Flutter Befehle

## Build Webapp

flutter build web --release --base-href="/appringana/web/"

## Run Andorid emulator

flutter run -d emulator-5554 // Pixel 7

## Appbundle

flutter build appbundle