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:

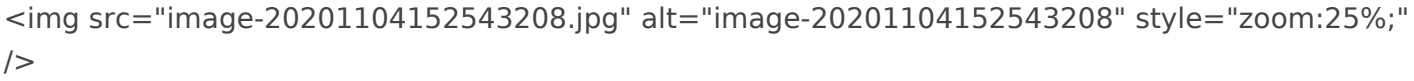
```
ausgabe = "hallo {0}, du hast noch {1} Euro auf dem Konto".format("Hermann", 123.13)


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.

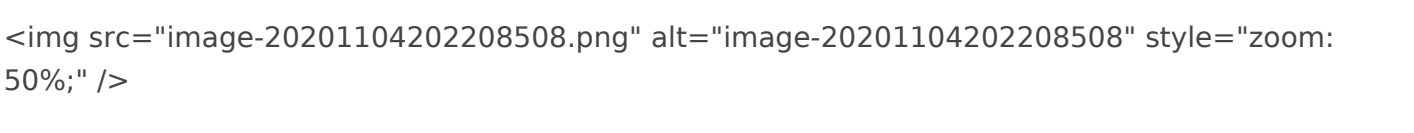
```
while True:
    try:
        zahl = int(input("Zahl: "))
        break

    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

Revision #10
Created 21 March 2023 10:16:39 by Hermann
Updated 24 November 2023 05:56:38 by Hermann