Inhaltsverzeichnis

LS 2 – Info - Tkinter

GUI-Beispiel

beispiel_ls2_01b.py

```
import tkinter as tk
from tkinter import ttk
from tkinter import messagebox

class Taschenrechner:
    """Klasse Taschenrechner"""

    def __init__(self):
        """Konstruktor, in dem das Taschenrechnerfenster aufgebaut wird"""

        self.hauptrahmen_anlegen()
        self.fensterelemente_anlegen()
        self.elementabstaende_setzen()

        self.fenster.mainloop()

    def hauptrahmen_anlegen(self):
        """ Fenster und Hauptrahmen anlegen """

        self.fenster = tk.Tk(className="Beispiel_ls2_01b")
        self.fenster.iconbitmap("bkgut.ico")

        self.hauptrahmen = ttk.Frame(master=self.fenster, padding="20")
        self.hauptrahmen.grid(column=1, row=1, sticky="NWES")
        self.fenster.columnconfigure(1, weight=1)
        self.fenster.rowconfigure(1, weight=1)

    def fensterelemente_anlegen(self):
        """ Fensterelemente einfügen """
        self.lbl_zahl_1 = ttk.Label(master=self.hauptrahmen, text="Zahl1")
        self.lbl_zahl_1.grid(column=1, row=1, columnspan=2, sticky="NW")

        self.wert_1 = tk.StringVar()
        self.zahl_1 = ttk.Entry(master=self.hauptrahmen,
textvariable=self.wert_1)
        self.zahl_1.grid(column=3, row=1, columnspan=2, sticky="NWE")

        self.lbl_zahl_2 = ttk.Label(master=self.hauptrahmen, text="Zahl2")
        self.lbl_zahl_2.grid(column=5, row=1, columnspan=2, sticky="NW")

        self.wert_2 = tk.StringVar()
        self.zahl_2 = ttk.Entry(master=self.hauptrahmen,
textvariable=self.wert_2)
        self.zahl_2.grid(column=7, row=1, columnspan=2, sticky="NWE")

        self.btn_add = ttk.Button(
            master=self.hauptrahmen, text="+", command=lambda:
self.ausfuehren("+")
        )
        self.btn_add.grid(column=1, row=2, sticky="NW")
```

```
        self.btn_sub = ttk.Button(
            master=self.hauptrahmen, text="-", command=lambda:
self.ausfuehren("-")
        )
        self.btn_sub.grid(column=2, row=2, sticky="NW")

        self.btn_mul = ttk.Button(
            master=self.hauptrahmen, text="x", command=lambda:
self.ausfuehren("*")
        )
        self.btn_mul.grid(column=3, row=2, sticky="NW")

        self.btn_div = ttk.Button(
            master=self.hauptrahmen, text="/", command=lambda:
self.ausfuehren("/")
        )
        self.btn_div.grid(column=4, row=2, sticky="NW")

        self.lbl_ergebnis = ttk.Label(master=self.hauptrahmen, text="Ergebnis")
        self.lbl_ergebnis.grid(column=1, row=3, columnspan=2, sticky="NW")

        self.wert_ergebnis = tk.StringVar()
        self.ergebnis = ttk.Entry(
            master=self.hauptrahmen, textvariable=self.wert_ergebnis
        )
        self.ergebnis.grid(column=3, row=3, columnspan=2, sticky="NWE")


    def elementabstaende_setzen(self):
        """ Abstände setzen """
        for element in self.hauptrahmen.winfo_children():
            element.grid_configure(padx="10", pady="10")

    def ausfuehren(self, rechenart):
        """Allgemeine Rechenmethode mit Exception-Handling"""

        try:
            z_1 = float(self.wert_1.get())
            z_2 = float(self.wert_2.get())
            if rechenart == "+":
                erg = z_1 + z_2
            if rechenart == "-":
                erg = z_1 - z_2
            if rechenart == "*":
                erg = z_1 * z_2
            if rechenart == "/":
                erg = z_1 / z_2
        except ValueError:
            messagebox.showerror("Fehlermeldung", "Sie müssen eine Zahl
eingeben!")
            self.wert_ergebnis.set("")
            return
        except ZeroDivisionError:
            messagebox.showerror("Fehlermeldung", "Sie dürfen nicht durch Null
teilen!")
            self.wert_ergebnis.set("")
            return
        self.wert_ergebnis.set(str(erg))
```

```
Taschenrechner()
```

Tkinter Widgets

- button
- canvas
- checkbutton
- combobox
- entry
- frame
- label
- labelframe
- listbox
- menu
- menubutton
- message
- notebook
- tk_optionMenu
- panedwindow
- progressbar
- radiobutton
- scale
- scrollbar
- separator
- sizegrip
- spinbox
- text
- treeview

*) es gibt noch mehr; Erweiterungen

TTK Widgets

- Button
- Checkbutton
- Entry
- Frame
- Label
- LabelFrame
- Menubutton
- PanedWindow
- Radiobutton
- Scale
- Scrollbar
- Spinbox
- Combobox
- Notebook
- Progressbar
- Separator
- Sizegrip
- Treeview

Tkinter Layout-Manager

- **place**
- **grid**
- **pack**

Tipps zum Programmieren von GUIs mit Tkinter

- Nutzen Sie, wo vorhanden, die neueren themed-Klassen von Tkinter (TTK)
- Nutzen Sie die Grid-Technologie zur Gestaltung
- Strukturieren Sie das GUI objekt-orientiert
- Nutzen Sie Styles zur Formatierung; möglichst wenig direkte Formatierung am Objekt selber

Wo finden ich weitere Informationen zu Tkinter?

- tkinter_tutorial1.pdf
- wiki.python.org/moin/TkInter
- www.python-kurs.eu/python_tkinter.php
- realpython.com/python-gui-tkinter/

LS 2 – Info – Objekt-orientierter Aufbau einer Tkinter-Anwendung

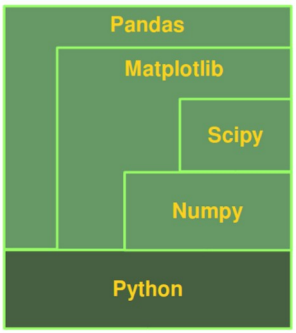
Im folgenden sind einige Hinweise zu einer sinnvollen objekt-orientierten Strukturierung einer Tkinter Anwendung aufgelistet.

- Pro Fenster eine GUI-Klasse erstellen. Wir werden in der Schule nur GUIs betrachten, die aus einem Fenster bestehen.
- Im eigentlichen Python-Skript nur ein Objekt der GUI-Klasse anlegen.
- Den Fensteraufbau im Konstruktor programmieren.
- Zur Formatierung des Fensters TTK-Styles nutzen. Diese in einer eigenen Methode programmieren, welche vom Konstruktor aufgerufen wird.
- Keine GUI-Elemente direkt im Fenster platzieren. Stattdessen im Fenster einen Hauptrahmen platzieren und in diesen die GUI-Elemente platzieren. Das Anlegen von Fenster und Hauptrahmen in einer eigenen Methode programmieren, welche vom Konstruktor aufgerufen wird.
- Muss man auf GUI-Elemente später noch einmal zugreifen, muss man sich diese als Attribute der Klasse merken.
- Muss man auf GUI-Elemente später nicht mehr zugreifen, braucht man keine Klassenattribute hierfür und kann sie anonym anlegen.
- Das Anlegen der GUI-Elemente in einer eigenen Methode programmieren, welche vom Konstruktor aufgerufen wird. Wird diese Methode zu groß, kann man auf mehrere Methoden aufteilen.
- Die Ereignis-Methoden, die z.B. bei einem Kopfdruck aufgerufen werden, als Methoden in der GUI-Klasse programmieren.
- Wenn die Ereignis-Methoden komplexe Anwendungslogik erfordern, diese in eigenen Klassen realisieren.
- Nutzen Sie, wo vorhanden, die neueren themed-Klassen von Tkinter (TTK).
- Nutzen Sie die Grid-Technologie zur Gestaltung.

LS 2 – Info - NumPy

Was ist NumPy?

- Python Modul zum effezienten Umgang mit Feldern
- Homogenes Feld, d.h. alle Elementen besitzen den gleichen Datentypen
- besonders schnell; in C implementiert
- für große Felder gut geeignet
- wird von einigen anderen Modulen im Bereich Big-Data / Data Engineeering als Basis genutzt



Wie legt man ein NumPy-Feld an?

a) mit Python-Feldern → `array()`

b) mit festen Zahlen → `zeros(), ones(), full()`

c) mit Zufallszahlen → `random.random(), random.randint()`

d) mit gleichmäßiger Aufteilung → `arange(), linspace()`

Wie kann man auf ein NumPy-Feldelement zugreifen?

- über Index → `[5]` → `[3,7,8]`

Welche Feldeigenschaften gibt es?

a) Form des Feldes → `.shape`

b) Anzahl Dimenesionen → `.ndim`

c) Anzahl Elemente → `.size`

d) Datentyp → `.dtype`

Welche Funktionen zum Atbeiten mit NumPy-Feldern gibt es?

a) Feldelement imt Einzelwert, arithmetische Operationen → `+, ...`

b) Math. Funktionen auf Feldelementen → `sin(), cos(), ...`

c) Aggregat-Funktionen → `sum(), min(), max(), ...`

d) Funktionen, um Feld umzuformen → `flatten(), transpose(), reshape(), ...`

e) Funktionen um Felder zusammenzufügen und zu trennen → `concatenate(), stack(), splitt(), ...`

f) Funktionen um Elemente hinzuzufügen und zu entfernen → `append(), insert(),`

`delete(), …`

Wie kann man NumPy-Felder speichern und laden?

a) im NumPy-Format → `save(), load()`

b) im CSV-Format → `savetxt(), loadtxt()`

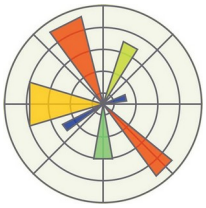
Wo finden ich weitere Informationen zu NumPy?

https://www.w3schools.com/python/numpy/default.asp

https://www.python-kurs.eu/numpy.php

https://numpy.org/

LS 2 – Info - Matplotlib



Was ist Matplotlib?

- Python Bibliothek zur Datenvisualisierung
- wird häufig bei Big-Data / Data-Science / Scientific-Computing genutzt

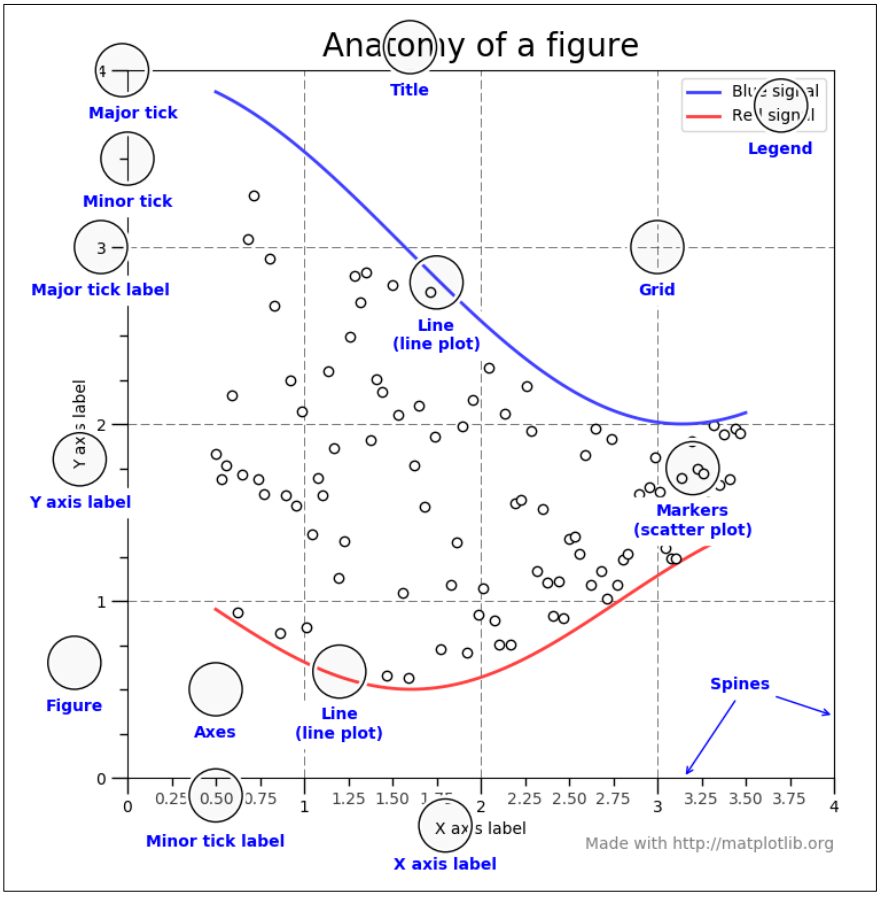
Welche Programmier-Schittstellen gibt es?

1. **Objekt-Orientierte-Schnittstelle** → **für Programmierung empfohlen**
2. Vereinfachte prozedurale Schnittstelle → für interaktive Nutzung empfohlen
3. Matlab-kompatible Schnittstelle → veraltet; nicht mehr empfohlen

Welche Daten-Schittstellen gibt es?

- Als Daten für die gezeichneten Graphiken dienen NumPy Felder (oder ähnliche)

Elemente einer Matplotlib Graphik



Zentrale Objekte bei der Programmierung einer Matplotlib Graphik

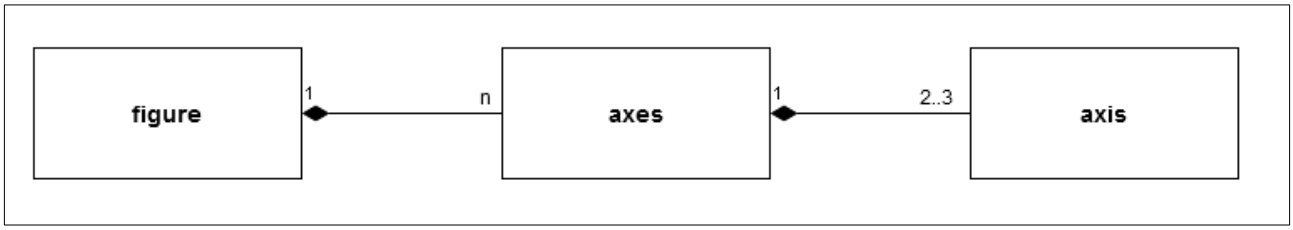
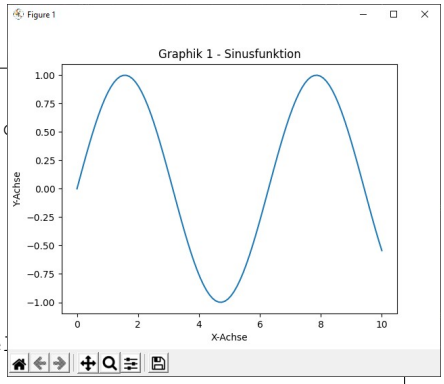


figure	Bild, Rahmen
axes	Zeichenbereich in dem Rahmen; es kann mehrere geben; hier wird gezeichnet
axis	Achse; je nach Dimension 2 oder 3

Beispielcode einer einfachen Matplotlib-Graphik



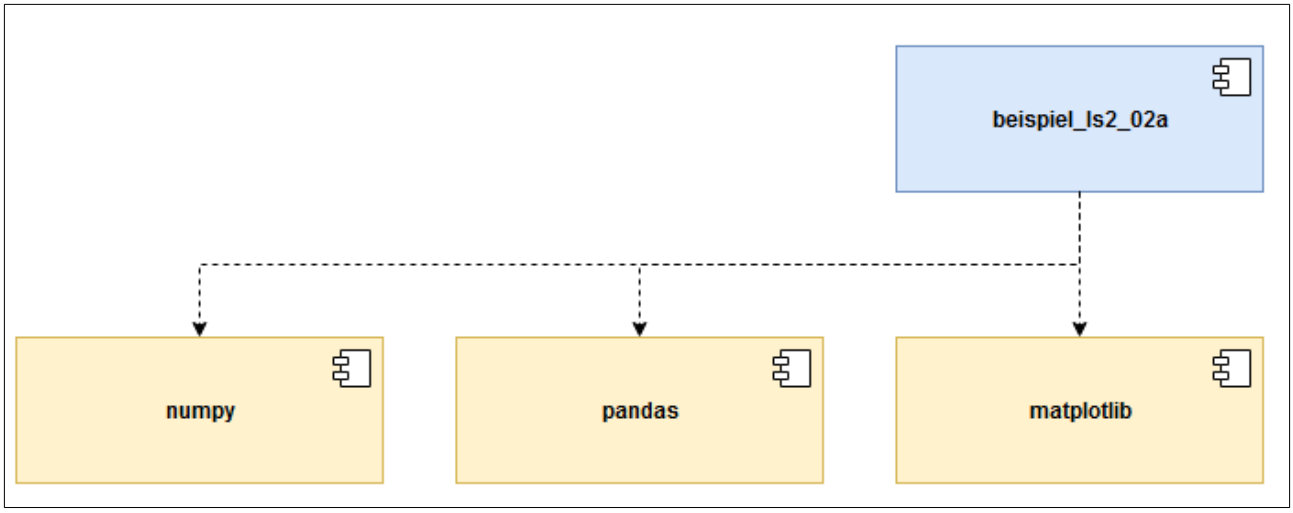

```
def graphik1():
    """ Graphik 1 - Darstellung einer Sinus-Funktion im Bereic

    xWerte = np.linspace(0, 10, 100)  # x-Werte setzen
    yWerte = np.sin(xWerte)  # y-Werte berechnen

    fig, ax = plt.subplots()  # Rahmen und Zeichenfäche erste
    ax.plot(xWerte, yWerte)  # Werte des Graphen festlegen

    ax.set_title("Graphik 1 - Sinusfunktion")  # Diagramm- und Achsentitel hinzufügen
    ax.set_xlabel("X-Achse")
    ax.set_ylabel("Y-Achse")

    plt.show()  # Graphik darstellen
```

Wo finden ich weitere Informationen zu Matplotlib?

- https://www.w3schools.com/python/matplotlib_intro.asp
- https://matplotlib.org/stable/tutorials/
- matplotlib_tutorial.pdf

LS 2 – Info – Matplotlib-Beispiel

Matplotlib-Beispiel - Komponentendiagramm



Matplotlib-Beispiel – Komponentendiagramm