intro2

August 24, 2023

1 Notwendige Basics

1.1 Installation

Über die Homepage von Python könnt ihr Python herunterladen. Falls ihr eune neue Python Version installieren wollt, empfehlen wir die Version 3.10 zu nehmen.

Da wir in diesem Praktikum mit Jupyter arbeiten werden und ihr auch euren Code in einem Notebook strukturieren und darstellen müsst, ist der nächste Schritt Jupyter zu installieren. Informationen dazu findet ihr in der Dokumentation von Jupyter.

1.2 Git

Um während der Implementation oder Weiterentwicklung von Code keine neuen Fehler reinzubauen oder Versionen, die schon funktioniert haben, zu verlieren, gibt es sogenannte *version control systems*, wie zum Beispiel Git.

Ein sauberer Programmier-Workflow beinhaltet umbedingt die Benutzung von Git und das regelmäßige *committen* des Fortschritts.

1.2.1 GitHub

In einer Gruppe gemeinsam ein Projekt zu implementieren bringt noch weitere Probleme mit sich. Wie teilt man die Aufgaben auf? Wie kann man parallel am gleichen Code arbeiten ohne den Fortschritt der anderen zu überschreiben? Wie kann man den Code zusammenführen und sicherstellen, dass alles funktioniert?

Diese Probleme können ebenfalls mit Git behoben werden. Es gibt zwei bekannte Git Online-Dienste, GitHub und GitLab. Über diese könnt ihr euren Code *pushen*, *clonen*, *mergen*, *Issues* melden, etc. Ganz nebenbei habt ihr noch ein BackUp eures Codes auf einem Server.

Eure Projekte kann man nicht mit der Komplexität der Implementation eines Programms wie zum Beispiel Signal vergleichen. Deshalb ist es nicht notwendig sich in Git und GitHub einzuarbeiten, um eure Projekte gut zu entwickeln.

1.3 Python

Wir haben ein paar ein paar Quellen für Programmierung in Python zusammengesucht. Darunter sind Anfängerkurse bis hin zu ausgiebigen Büchern, die fortgeschrittene Themen behandeln.

- EinfProg an der TF
- Youtube Programmierkurs

- Think python
- Dive into python3
- Guide to python

Diese Quellen sind nicht notwendig für diesen Kurs, sondern als mögliche Informations- oder Weiterbildungsquellen gedacht.

Falls ihr checkstyles für noch schöneren Code durchlaufen lassen wollt, empfehlen wir PEP8 via pycodestyle oder Pylint.

Nochmals zu erwähnen ist, dass Code **immer** gut dokumentiert und für alle verständlich programmiert sein muss (siehe Zen of Python).

1.3.1 Jupyter

Nachdem ihr Jupyter installiert habt, könnt ihr Notebooks erstellen und ausführen. Einfache Einführungen wie man in Jupyter Notebooks programmiert findet ihr in den folgenden Links. * Starten einer Jupyter Lab Session * Erklärung des Interface

Die einfachste Möglichkeit ist über das Terminal/Eingabeaufforderung das web-based developing Interface zu starten, wie im ersten Link erklärt. Außerhalb eines Browser kann man Jupyter Notebooks auch zum Beispiel in Visual Studio Code laufen lassen. Dafür gibt es hier eine kurze Einführung.

Markdown In Jupyter Notebooks hat man die Möglichkeit auszuwählen welchen Typ die Zelle hat. Zur guten Dokumentation kann man Markdown Zellen, wie hier demonstriert, verwenden. Dabei verwendet man die Formatierungssprache Markdown language.

Hier findet ihr eine Einführung in mögliche Befehle dieser Sprache.

Sehr lukrativ für eure Notebooks wird sein mathematische Befehle wie in LaTeX problemlos einzubinden:

$$\mathbb{E}[X] = \int_{\Omega} x \mathrm{d}\mathbb{P}$$

Als Beispiele für gut strukturierte Notebooks könnt ihr euch dieses git-hub Repository anschauen.

Plots einfügen Um direkt in einem Jupyter Notebook zu plotten gibt es verschiedene Möglichkeiten, z.B Matplotlib, Seaborn, Plotly, Altair, Bokeh.

Wir empfehlen Matplotlib und gegebenfalls zusätzlich Seaborn, was auf Matplotlib basiert, zu verwenden.

Damit Plots in dem Notebook direkt angezeigt werden muss man einen sogenannten Magic Befehl ausführen.

```
%matplotlib argument
```

Dabei gibt es verschiedene Argumente, die man hinzufügen kann.

Hier ein Beispiel:

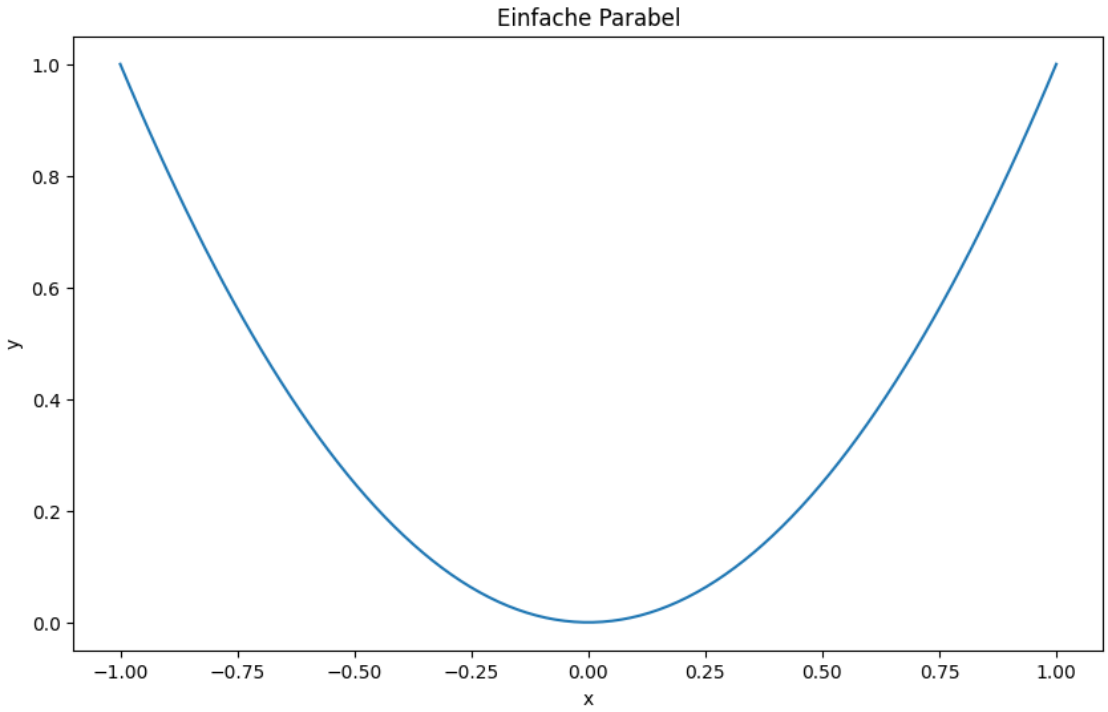
[ ]:
```
import matplotlib.pyplot as plt
import numpy as np
```

```
import ipywidgets as ipw
```

[ ]:
```
%matplotlib inline

# calc what to plot
x = np.linspace(-1, 1, 100)
y = x**2

# init figure
plt.figure(figsize=(10, 6))
# Set labels and title
plt.xlabel("x")
plt.ylabel("y")
plt.title("Einfache Parabel")
# plot the lines between the (x,y) points
plt.plot(x, y)
plt.show()
```



Dahingegen erlaubt das Argument widget interaktive Plots.

[ ]:
```
%matplotlib widget


def plot_powers(n=50, k=25):
```

```
    x = np.linspace(-1, 1, n)
    y = np.sin(x*k*np.pi)

    plt.figure(figsize=(10, 6))
    # Set labels and title
    plt.xlabel("x")
    plt.ylabel("y")
    plt.title(f"Plot sin({k}pi*x) an {n} Stützstellen")
    plt.xlim(-1, 1)
    plt.ylim(-1, 1)
    plt.plot(x, y)
    plt.show()

ipw.interact(
    plot_powers,
    n=(2, 100),
    k=(1, 50)
    )
# Dieser Plot wird leider im PDF nicht angegeben!
```

```
interactive(children=(IntSlider(value=50, description='n', min=2),␣
↪IntSlider(value=25, description='k', max=50…
```

[ ]:
```
<function __main__.plot_powers(n=50, k=25)>
```