ALteP-Einfuehrungsveranstaltung

Installation

```
conda create -n jupyter python=3.10
```

```
conda activate jupyter
```

```
pip install jupyterlab scipy numpy pandas matplotlib tqdm sympy
```

```
python -m IPython
```

- Case-Sensitive!
- wenn Anaconda installiert ist (nicht miniconda), dann über das Startmenü startbar

```
python -m jupyterlab
```

```
python -m jupyterlab <Pfad zu einem Ordner oder IPYNB>
```

Beispiel Einrückung / Funktionen / If

in C

```c
int top(int n) {
    if (n % 2 != 0) {
        n = n + 1;
        return n / 2;
    } else {
        return n / 2;
    }
}
```

oder auch:

```
int top(int n) {if (n % 2 != 0) {n = n + 1; return n / 2;} else {return n / 2;} }
```

in Python

```
def top(n):
    if n % 2 != 0:
        n = n + 1
        return n / 2
    else:
        return n / 2
```

Aufgabe 1 Collatz (Funktionsdefinition & while & if):

Collatz: ungelöstes mathematisches Problem. Einfach formuliert:

- Beginne mit irgendeiner natürlichen Zahl n > 0
- ist n gerade: nimm als nächste Zahl n / 2
- ist n ungerade: nimm als nächstes 3*n + 1
- unendlich Wiederholen mit neuer Zahl

Collatz Hypothese ist, dass egal mit welcher Zahl man beginnt, die Sequenz immer in den unendlichen Zyklus 4, 2, 1 mündet

Aufgabe 2 Monte-Carlo-Simulation (Funktionsdefinition & for & if):

Monte-Carlo-Simulation: stochastische Methode, welche zufällige Stichproben verwendet, um Ergebnisse in komplexen System zu approximieren.

In dieser Aufgabe soll Pi approximiert werden:

- zufällig generierte Punkte innerhalb eines Einheitsquadrats
- Bestimmung der Anzahl der Punkte, welche innerhalb des Einheitskreises liegen
- das Verhältnis der Punkte im Einheitskreis zur Gesamtzahl der Punkte ermöglicht eine Annäherung an den Wert von Pi

$$A_{EK} = \frac{\pi}{4} \cdot D^2$$

$$A_{EQ} = D^2$$

$$A_{EK} = \frac{\pi}{4} \cdot A_{EQ}$$

$$\pi = 4 \cdot \frac{A_{EK}}{A_{EQ}}$$

- da von gleichverteilten zufälligen Punkten ausgegangen wird, können die Anzahlen der Punkte gleich den Flächen in obiger Formel gesetzt werden

List, Tuple, Set & Dictionary

In Python sind Listen, Tuple, Sets und Dictionaries vier verschiedene Arten von Sammlungen, die jeweils unterschiedliche Eigenschaften und Verwendungszwecke haben:

1. Listen (Lists):
    - Listen sind geordnete Sammlungen von Objekten
    - Die Elemente einer Liste können verändert werden ("mutable")
    - Beispiel: `my_list = [1, 2, 3, 4, 5]`
2. Tuple:
    - Tuple sind ähnlich wie Listen geordnete Sammlungen von Objekten
    - Die Elemente eines Tuple können NICHT verändert werden ("immutable")
    - Tuple werden durch runde Klammern () dargestellt, wobei die Elemente durch Kommas getrennt werden
    - Beispiel: `my_tuple = (1, 2, 3, 4, 5)`
3. Sets:
    - Sets sind ungeordnete Sammlungen von eindeutigen Elementen
    - Sets erlauben keine doppelten Elemente, d.h. jedes Element kommt nur einmal vor
    - Sets sind veränderlich und können Elemente hinzufügen oder entfernen
    - Auf die Elemente eines Sets kann NICHT mittels Index zugegriffen werden
    - Sets werden durch geschweifte Klammern {} oder mit der Funktion `set()` erstellt.
    - Beispiel: `my_set = {1, 2, 3, 4, 5}`
4. Dictionaries (Dicts):
    - Dictionaries, oder kurz Dicts, sind ungeordnete Sammlungen von Schlüssel-Wert-Paaren
    - Jeder Schlüssel in einem Dictionary ist eindeutig und mit einem Wert verknüpft
    - Die Elemente in einem Dictionary können verändert werden ("mutable")
    - Dicts werden durch geschweifte Klammern {} dargestellt, wobei Schlüssel und Werte durch einen Doppelpunkt getrennt sind.
    - Beispiel: `my_dict = {"Name": "John", "Alter": 25, "Stadt": "Berlin"}`

Aufgabe 3 "Random Walk":

- Es wird ein Startpunkt (`x0`, `y0`) vorgegeben
- mit `x0` & `y0` jeweils eine Liste initialisieren
- Folgende Abfolge wird n-mal wiederholt:
    - die letzten Werte der Liste jeweils zufällig mit +1 oder -1 addieren
    - die neuen Koordinaten `x` & `y` den jeweiligen Listen hinzufügen
- das Ergebnis mit matplotlib plotten