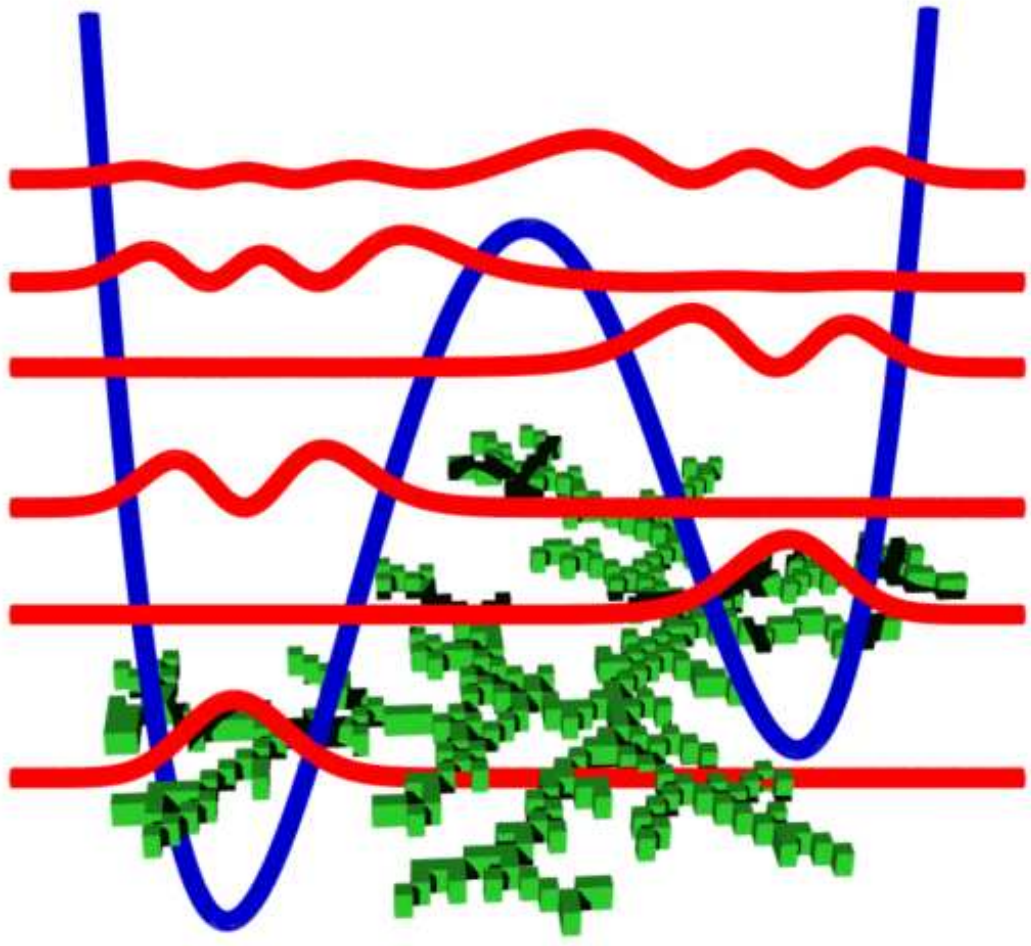
Computational Physics – OOP Einführung



Was ist objektorientiertes Programmieren (OOP)?

- Bisher: Prozedurale Programmierung
  - Folge von Anweisungen
  - Funktionen für einzelne Teilaufgaben
  - Daten und Funktionen getrennt
- Jetzt: Erweiterung durch Objekte - Grundgedanke
  - Abgeschlossene Objekte enthalten Daten **und** Funktionen

```
1  class Linie:
2      def __init__(self, x0, y0, x1, y1):
3          self.x0, self.x1 = x0, x1
4          self.y0, self.y1 = y0, y1
5
6      def laenge(self):
7          return np.sqrt((self.x1 - self.x0)**2 +
8                         (self.y1 - self.y0)**2)
```

Was ist objektorientiertes Programmieren (OOP)?

Beispiel

```
1  from linie import Linie     # importiere Klasse aus linie.py
2
3  linie1 = Linie(0.0, 0.0, 1.0, 1.0)      # erzeuge 1. Linie
4  linie2 = Linie(3.0, 4.0, 1.0, 4.0)      # erzeuge 2. Linie
5
6  linie1.laenge()                  # zwei unabhaengige Datensaetze
7  # Output: 1.4142135623730951
8
9  linie2.laenge()
10 # Output: 2.0
```

Vorteile:

- Klassen unabhängig voneinander
  - keine globalen Variablen
  - Klasse enthält alle Daten (in sich abgeschlossen)

=> Modularer Kode einfacher wiederverwendbar

Zum Ausprobieren

```
In [1]: import numpy as np
In [2]: class Linie:
...:     def __init__(self, x0, y0, x1, y1):
...:         print("Konstruktor wurde aufgerufen.")
...:         self.x0, self.x1 = x0, x1
...:         self.y0, self.y1 = y0, y1
...:
...:     def laenge(self):
...:         print("Laenge wird berechnet.")
...:         return np.sqrt((self.x1 - self.x0)**2 +
...:                        (self.y1 - self.y0)**2)
...:
In [3]: l1 = Linie(0, 0, 1, 1)
Konstruktor wurde aufgerufen.
In [4]: l2 = Linie(3, 4, 1, 4)
Konstruktor wurde aufgerufen.
In [5]: print(l1.laenge())
Laenge wird berechnet.
1.41421356237
In [6]: print(l2.laenge())
Laenge wird berechnet.
2.0
```

Teil 2 $\rightarrow$ nächste Woche

Umfangreicheres Beispiel

Ziel

- Stempel mit Symbolen auf Bildschirm zeichnen
- Dabei sollen unterschiedliche Symbole zur Auswahl stehen

Eigenschaften eines Stempels (Attribute)

- Farbe
- Größe
- Informationen über das Aussehen

Was soll der Stempel können? (Methoden)

- Stempel mit speziellen Parametern erzeugen (Konstruktor)
- Stempeln! (`Stempel.plotten()`)

Gemeinsame Eigenschaften: Basisklasse

Idee: Alle Stempel sind strukturell gleich!

=> Fasse Grundstruktur in Basisklasse zusammen

```
1  class Stempel:
2      """Stempel-Basisklasse"""
3
4      def __init__(self, farbe='black', groesse=1.0):
5          """Basiseigenschaften festlegen"""
6          self.farbe = farbe
7          self.groesse = groesse
8          self.daten = None
9
10     def plotten(self, x, y):
11         """Ausgabe mittels matplotlib"""
12         plt.plot(x + self.daten[0]*self.groesse,
13                  y + self.daten[1]*self.groesse,
14                  color=self.farbe)
```

Ein Spezifischer Stempel: Vererbung von Klassen

- Attribute und Methoden der Basisklasse werden übernommen
- können aber auch neu definiert / überschrieben werden

```
1  class LinienStempel(Stempel):
2      """Plotte eine einfache Linie.
3
4      Nutze dazu die Basisklasse Stempel.
5      """
6
7      def __init__(self, farbe=(0, 0, 0), groesse=1.0):
8          """Basiseigenschaften festlegen"""
9          # ueberschriebene Methode aufrufen
10         super(LinienStempel, self).__init__(farbe, groesse)
11
12         # ,,echte'' Plotdaten hinterlegen
13         self.daten=(np.array([-0.5, 0.5]), np.array([-0.5, 0.5]))
```

- Und das war alles!

Das Hauptprogramm

- Für das Hauptprogramm sehen *alle* Stempel gleich aus
- Sie verfügen alle über die Methode `plotten`
  sowie die Attribute `farbe` und `groesse`

```
1  st = LinienStempel(farbe='red')
2  st.plotten(0.6, 0.8)
3  print(st.groesse)
```

- Es ist dabei unwichtig, um welchen Stempel es sich handelt