
Übung zum Kompaktkurs
Einführung in die Programmierung zur Numerik mit Python
Wintersemesterferien 2015/2016 — Blatt 4

Aufgabe 1 (Klasse für rationale Zahlen)

- (a) Schreiben Sie eine Klasse `RationaleZahl`, die eine rationale Zahl $\frac{p}{q}$ mit $p, q \in \mathbb{Z}$ und $q \neq 0$ darstellt. Sorgen Sie beim Erstellen eines `RationaleZahl`-Objektes dafür, dass p und q teilerfremd sind. Dividieren Sie dazu durch den größten gemeinsamen Teiler von p und q , den Sie durch folgenden rekursiven Algorithmus herausfinden können:

$$\text{ggT}(p, q) = \begin{cases} p & \text{falls } q = 0 \\ \text{ggT}(q, p \bmod q), & \text{sonst.} \end{cases}$$

Die Modulo-Operation wird in Python durch `p % q` realisiert. Zwei `RationaleZahl`-Objekte sollen sich addieren, subtrahieren, multiplizieren und dividieren lassen. Implementieren Sie dazu die sogenannten *magic members* `__add__`, `__sub__`, `__mul__` und `__div__`, die neben dem obligatorischen Argument `self` noch ein weiteres Objekt `other` erhalten und ein neues Objekt vom Typ `RationaleZahl` – gerade das Ergebnis der Rechenoperation – zurückliefern. Testen Sie in jeder dieser Funktionen, ob `other` tatsächlich vom Typ `RationaleZahl` ist und werfen Sie ggf. eine `TypeError`-Exception (den Typ eines Objektes können Sie mit `type(obj)` erfahren).

- (b) Implementieren Sie auch die magic members `__str__` (liefert eine `str`-Darstellung der Klasse und ermöglicht damit Aufrufe der Form `print(obj)`), `__float__` (liefert eine `float`-Darstellung), `__eq__` (testet zwei `RationaleZahl`-Objekte auf Gleichheit) sowie `__ne__` (testet auf Ungleichheit). Die ersten beiden erhalten nur das Argument `self`, die letzten beiden zusätzlich noch ein Objekt `other`. Testen Sie auch hier, ob `other` von geeignetem Typ ist.

Aufgabe 2 (Klassenhierarchien)

- (a) Erarbeiten Sie eine Klassenhierarchie für die folgenden geometrischen Formen:
- Rechteck
 - Parallelogramm
 - Raute

Überlegen Sie zunächst, welches der Objekte als Basisklasse in Frage kommt und programmieren Sie dann die Klassen entsprechend. Sorgen Sie dafür, dass jede der drei geometrischen Formen die Methoden `umfang` und `flaeche` besitzt und per `print(obj)` ausgegeben werden kann. Überlegen Sie insbesondere, welche Argumente die jeweiligen `__init__`-Methoden benötigen.

- (b) Programmieren Sie zusätzlich noch eine Klasse `Quadrat`, die ein Quadrat repräsentieren und als Basisklassen sowohl `Rechteck` als auch `Raute` haben soll.

Hinweise:

- Ein Parallelogramm ist beispielsweise durch Angabe der zwei Seitenlängen a, b und des Winkels α eindeutig bestimmt.
- Wenn Sie zu Beginn Ihres Skriptes den Befehl `import math` einfügen, können sie die Funktion `math.sin(x)` und die Konstante `math.pi` benutzen (Die Sinus-Funktion rechnet im Bogenmaß!)
- Wenn `B` von `A` abgeleitet ist, kann man durch den Befehl `A.__init__(self, arg)` die Initialisierung der Basisklasse aufrufen, wobei `arg` durch die geforderten Argumente zu ersetzen ist.

Aufgabe 3 (Polynominterpolation)

Es sei $f : \mathbb{R} \rightarrow \mathbb{R}$ eine Funktion, $x_0, \dots, x_n \in \mathbb{R}$ die Stützstellen und $f_i := f(x_i)$ für $i = 0, \dots, n$ die Auswertungen von f in den Stützstellen. Das Interpolationspolynom $P_f : \mathbb{R} \rightarrow \mathbb{R}$ ist das (eindeutig bestimmte) Polynom vom Grad n mit $P_f(x_i) = f_i$. Mit dem Schema der *dividierten Differenzen* berechnet man die Koeffizienten $c_0, \dots, c_n \in \mathbb{R}$ des Interpolationspolynoms $P_f(x)$ auf folgende Weise: Definiere für $0 \leq i, j \leq n$ mit $i \leq j$ rekursiv die Ausdrücke $D_i^j \in \mathbb{R}$ durch

$$D_i^j := \begin{cases} f_i & \text{falls } i = j, \\ \frac{D_{i+1}^j - D_i^{j-1}}{x_i - x_j} & \text{sonst.} \end{cases}$$

Dann gilt $c_i = D_0^i$ und das Interpolationspolynom lautet

$$P_f(x) = \sum_{i=0}^n c_i N_i(x)$$

mit $N_0(x) = 1$ und $N_i(x) = \prod_{j=0}^{i-1} (x - x_j)$ für $i = 1, \dots, n$. Schreiben Sie nun eine Klasse `PolynomInterpolation` zur Interpolation von Funktionen. Die Klasse soll folgende Funktionalität beinhalten:

- Die Initialisierung erfordert die Angabe der zu interpolierenden Funktion und einer Menge von Stützstellen, an denen sie interpoliert werden soll. Berechnen Sie die Koeffizienten c_i bereits bei der Initialisierung.
- Eine Methode `auswerten` soll für einen gegebenen Punkt x das Interpolationspolynom an der Stelle x auswerten. Verwenden Sie dazu das *Schema von Horner* (s.u.).

- (c) Zur Visualisierung der Interpolationsgüte soll eine Methode **plotten** bereitgestellt werden, die sowohl die Funktion selbst als auch ihre Interpolierende in **ein** Koordinatensystem plottet.

Testen Sie die Klasse an der Funktion $f(x) = \tan(x)$ für eine von Ihnen gewählte Menge an Stützstellen auf dem Intervall $[-1.5, 1.5]$.

Hinweise: Das Schema von Horner geht so: Seien x_i die Stützstellen und c_i die Koeffizienten von oben. Für gegebenes $x \in \mathbb{R}$ definiere $b_n := c_n$ und rekursiv

$$b_i := b_{i+1}(x - x_i) + c_i.$$

Dann gilt $P_f(x) = b_0$.