Aufgabe 1:

Erstellen Sie eine Klasse `MySet`, mit denselben Funktionalitäten des `set` Datentyps. Ein Objekt der Klasse `MySet` soll erstellt werden können, indem eine Liste an den Konstruktor übergeben wird. Schon der Konstruktor soll dafür sorgen, dass keine doppelten Elemente vorkommen. Implementieren Sie mindestens die **magic methods** für `+` (das Soll die Vereinigung der Mengen sein) und `-` (das soll die Mengendifferenz sein). Die Liste der wichtigsten **magic methods** können Sie im Vorlesungsskript finden. Implementieren Sie neben den magic methods noch die Methode:

`add(self,a)`: Fügt `a` zur Menge hinzu.

Sie dürfen für die Aufgabe **keine** Funktionen für klassische `set`s verwenden.

Aufgabe 2:

Erstellen Sie eine Klasse `Matrix` zur Repräsentierung von Matrizen in $\mathbb{R}^{n\times m}$. Ein Objekt der Klasse soll erstellt werden können, indem eine geschachtelte Liste an den Konstruktor übergeben wird. Implementieren Sie die **magic methods** für Matrix Addition `+` und Matrix Multiplikation `*`. Stellen Sie dabei sicher, dass nur Matrizen mit passenden Dimensionen addiert bzw. multipliziert werden können.

Verwenden Sie Fehlerbehandlung um ungültige Eingaben (zum Beispiel das Addieren von Matrizen mit ungleichen Dimensionen) abzufangen und eine entsprechende Fehlermeldung auszugeben.

Aufgabe 3:

Installieren Sie das Modul `matplotlib`. Und plotten Sie eine beliebige Funktion.

Aufgabe 4:

Erstellen Sie eine Klasse `MyFunction`, welche Funktionen $f:[a,b]\to\mathbb{R}$ speichern kann. Dabei soll die Funktion selbst und der Definitionsbereich $[a,b]$ gespeichert werden. Implementieren Sie die **magic method** `__call__(self, x)`, sodass Objekte der Klasse `MyFunction` wie normale Funktionen aufgerufen werden können. Dabei soll der Definitionsbereich beachtet werden, d.h. wenn ein Wert außerhalb des Definitionsbereichs übergeben wird, soll eine entsprechende Fehlermeldung ausgegeben werden.

Aufgabe 5:

Gegeben sei ein lineares Gleichungssystem der Form

$$\begin{aligned} u_{11}x_1 + \cdots + u_{1n}x_n &= b_1 \\ \vdots \quad \vdots \qquad\qquad &, \\ u_{n1}x_1 + \cdots + u_{nn}x_n &= b_n \end{aligned}$$

wobei U eine obere Dreiecksmatrix ist. Für eine obere Dreiecksmatrix U gilt, $u_{ij} = 0$ für $i > j$. Durch die Dreiecksstruktur von U kann das Gleichungssystem effizient mit Rückwärtseinsetzen gelöst werden:

1. Berechne $x_n = b_n / u_{nn}$
2. Für i von $n-1$ bis 1 berechne

$$x_i = \frac{1}{u_{ii}} \left(b_i - \sum_{j=i+1}^{n} u_{ij} x_j \right)$$

Implementieren Sie die Funktion `back_substitution(U,b)`, die als Eingabe eine obere Dreiecksmatrix `U` und einen Vektor `b` erhält und den Lösungsvektor `x` zurückgibt.

Fangen Sie dabei mögliche Laufzeitfehler systematisch ab. Zeigen Sie, wie Ihr Code mithilfe des Python Debuggers auf Fehler getestet werden kann.

Aufgabe 6:

Erstellen Sie eine Klasse `Polynom` welche die Koeffizenten des Polynoms in einer Liste speichert, also

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

Die Liste der Koeffizienten soll and den Konstruktor übergeben werden, also zum Beispiel `Polynom([1,0,-2,3])` für das Polynom $p(x) = 1 - 2x^2 + 3x^3$. Implementieren Sie die **magic method** `__mul__(self,other)` die das Produkt zweier Polynome berechnen und als `Polynom` zurückgeben soll.