

# Einführung in die wissenschaftliche Programmierung

## Übungsblatt 8

### 1.) Telefonbuch – Exception Handling

Wandeln Sie das Telefonbuch vom letzten Übungsblatt so ab, dass fehlerhafte Eingaben über Exception-Handling abgefangen werden.

- Wird als Telefonnummer eine ungültige Zahl eingelesen, d.h. eine Zeichenfolge, die nicht nur aus den Zahlen 0 bis 9 besteht, soll ein `ValueError` verarbeitet werden.
- Versucht der Benutzer, die Eingabe von Telefonbuch-Daten frühzeitig durch die Eingabe von `Ctrl+C` abzuberechnen, soll der Text „Fehlerhafter Abbruch! Bitte Quit-Befehl nutzen“ ausgegeben werden und anschliessend wieder das Hauptmenü erscheinen. Welchen großen Vorteil haben wir durch das Abfangen dieses Eingabe-Fehlers? Was würde mit den vorhandenen Telefonbuch-Daten passieren ohne dieses Exception-Handling?

### 2.) Komplexe Zahlen (Hands-On)

Erstellen Sie eine Klasse `Komplex`, um komplexe Zahlen der Form  $x = a + ib$ ,  $a, b \in \mathbb{R}$  darzustellen. Sie soll zwei Klassenvariablen `a` und `b` von Typ `float` enthalten, die dem Realteil und Imaginärteil einer komplexen Zahl entsprechen.

Diese Klasse soll folgende Methoden implementieren:

- `abs`: bekommt eine komplexe Zahl  $x$  eingegeben und gibt den Betrag  $|x| = \sqrt{a^2 + b^2}$  zurück.
- `konjugiert`: bekommt eine komplexe Zahl  $x = a + ib$  eingegeben und gibt eine neue komplexe Zahl  $x^* = a - ib$  zurück.
- `real` und `imag`: geben den Realteil bzw. Imaginärteil von  $x$  zurück.

- Vier Methoden `__add__`, `__sub__`, `__mul__` und `__div__`, die die Operatoren `+`, `-`, `*` und `/` überladen.
- Zuletzt muss die Funktion `__str__` überladen werden, so dass jede komplexe Zahl eine vernünftige Darstellung als String hat.

Sie können Ihr Programm folgendermaßen testen:

```
>>> x = Komplex(1, 10)           # ruft __init__ auf
>>> y = Komplex(1, 20)           # ruft __init__ auf
>>> z = x + y                     # ruft __add__ auf
>>> print z                       # ruft __str__ auf
2.0 + i30.0
>>> print y.konjugiert()          # ruft konjugiert und __str__ auf
1.0 - i20.0
>>> print x/y                     # ruft __div__ und __str__ auf
0.501246882793 - i0.0249376558603
>>> print x.abs()                 # ruft abs und __str__ auf
10.0498756211
```

### 3.) Mandelbrot Menge

Die Mandelbrot-Menge  $M$  ist die Menge aller komplexen Zahlen  $c$ , für welche die Folge

$$z_{n+1} = z_n^2 + c, \quad z_0 = 0,$$

beschränkt bleibt. Visualisiert wird diese Menge üblicherweise wie in der Abbildung unten. Hierzu wird die Anzahl der Iterationen gezählt, bis **der Absolutwert der** Folge über z.B. den Wert 2.0 wächst. Erzeugen Sie nun die Mandelbrot-Menge für  $c \in [-2, 0.5] \times [1, -1]$  und stellen Sie diese mit TKInter graphisch dar.

- Diskretisieren Sie zuerst den  $c$ -Bereich in  $100 \times 100$  Teilstücke.
- Zählen Sie für jeden Wert von  $c$  die Anzahl der Iterationen, bis  $|z_{n+1}| > 2.0$ . Setzen Sie die maximale Anzahl von Iterationen auf 20 fest.
- Schreiben Sie eine Funktion `getColor(iter)`, die die Anzahl von Iterationen bekommt und eine Farbe zurückgibt. Diese soll folgende Fälle unterscheiden:
  - `iter < 4`, gibt 'magenta' zurück
  - `4 ≤ iter < 8`, gibt 'blue' zurück
  - `8 ≤ iter < 12`, gibt 'green' zurück

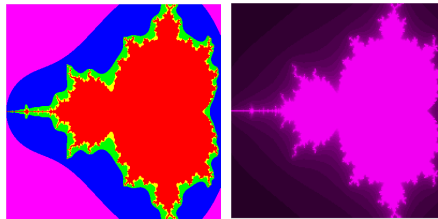
- $8 \leq \text{iter} < 16$ , gibt 'yellow' zurück
- $16 \leq \text{iter}$ , gibt 'red' zurück

iv) Malen Sie für jeden dieser  $100 \times 100$  Punkte ein Rechteck der Größe  $4 \times 4$ . Hinweis: Haben Sie die TK Klasse mit `root = Tk()` instanziiert, können Sie mit `can = tk.Canvas(root, width=200, height=200)` eine "Leinwand" erzeugen. Danach muss `can.pack()` aufgerufen werden. Auf der Leinwand können Sie mit

```
can.create_rectangle(x, y, x + 4, y + 4, fill = 'blue', outline = 'blue')
```

ein  $4 \times 4$  blaues Rechteck (mit blauem Rand) an der Stelle  $(x, y)$  erzeugen. Sie müssen `root.update()` nach dem Aufruf von `create_rectangle` aufrufen, um das Rechteck zu zeichnen.

Verwenden Sie die Klasse `Komplex` von Aufgabe 1 (oder alternativ die in Python eingebauten komplexen Zahlen). Testen Sie weitere Farbschemata, um die Mandelbrot Menge zu visualisieren!



## 4.) Newton-Verfahren (Hausaufgabe)

Mit dem Newton-Verfahren kann eine Approximation der Lösung zu der Gleichung  $f(x) = 0$  für eine stetig differenzierbare Funktion  $f : \mathbb{R} \rightarrow \mathbb{R}$  bestimmt werden. Beschrieben wird das Verfahren über die Iterationsvorschrift

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)},$$

wobei  $x_0 \in \mathbb{R}$  gegeben ist.

Schreiben Sie eine Funktion `newton(f, df, x, eps, N)` mit der Funktion `f`, der zugehörigen Ableitung `df`, dem Anfangswert `x`, einer Schranke `eps` und der maximalen Anzahl der Iterationen `N` als Parameter. Es soll dann bis zur Genauigkeit `eps` iteriert werden, jedoch maximal `N` viele Iterationen. Die Funktion `newton` soll die Nullstelle und den Wert an der Nullstelle (d.h. den Fehler) zurückgeben.

Testen Sie Ihre Implementierung mit folgenden Funktionen und Anfangswerten:

i)  $f(x) = 3x - x^2$ ,  $x_0 = 1.0$ ,  $x_0 = 1.5$  und  $x_0 = 2$ .

ii)  $f(x) = 1/x - 2$ ,  $x_0 = .25$  und  $x_0 = 1$ .

iii)  $f(x) = x^3 - 2x + 2$ ,  $x_0 = -1.0$  und  $x_0 = 0$ .

Verwenden Sie dafür  $\mathbf{eps} = 10^{-7}$  und  $N = 100$ . Was stellen Sie fest?