

Informatik I: Einführung in die Programmierung

Prof. Dr. Peter Thiemann
Hannes Saffrich
Wintersemester 2020

Universität Freiburg
Institut für Informatik

Übungsblatt 5

Abgabe: Montag, 07.12.2020, 9:00 Uhr morgens, über git¹

Hinweis

Aufgabenteile werden mit **0 Punkten** bewertet wenn:

- Dateien und Funktionen nicht so benannt sind, wie im Aufgabentext gefordert;
- Dateien falsche Formate haben, z.B. PDF statt plaintext; oder
- Pythonskripte wegen eines Syntaxfehlers nicht ausführbar sind.

Hinweis

Gruppenaufgaben müssen von allen Mitgliedern abgegeben werden und in der ersten Zeile müssen die Mitglieder in einem Kommentar vermerkt werden, z.B:

```
# Gruppe: xy123, yz56, zx934
```

Aufgabe 5.1 (Galgenmännchen; Datei: `hangman.py`; Punkte: 2+2+4+1)

In dieser Aufgabe sollen Sie das Spiel Hangman (zu Deutsch: Galgenmännchen) implementieren.

Ziel des Spiels ist es ein geheimes, zufällig ausgewähltes Wort zu erraten und dabei nur eine begrenzte Anzahl an Fehlversuchen zu machen. Zu Beginn werden alle Buchstaben des Wortes durch Platzhalter `_` ersetzt. Ist das geheime Wort beispielsweise `weather`, so ist nur Folgendes sichtbar:

```
-----
```

Es kann nun ein Buchstabe geraten werden. Ist der Buchstabe im Wort enthalten, werden alle Vorkommnisse des Buchstabens im Wort aufgedeckt. Wird beispielsweise `e` geraten, so ist nun Folgendes sichtbar:

```
_e___e_
```

Wird ein Buchstabe geraten, der nicht im Wort enthalten ist, so zählt dies als Fehlversuch. Ist eine bestimmte Anzahl an Fehlversuchen überschritten, so ist das Spiel verloren.

Das Raten ist nicht auf einzelne Buchstaben beschränkt, sondern es können auch zusammenhängende Teilwörter oder das ganze Wort geraten werden. Zum Beispiel könnte man als nächstes `th` raten und würde dann in folgenden Zustand kommen:

```
_e_the_
```

¹<https://inpro.informatik.uni-freiburg.de/>

Misslingt dies, so zählt es trotzdem nur als ein einzelner Fehlversuch.

Gehen Sie bei der Implementierung wie folgt vor:

- (a) Schreiben Sie eine Funktion `input_choice`, die eine Frage und eine Liste an möglichen Antworten als Argumente nimmt, und so lange mit `input` nach einer Eingabe fragt, bis die Eingabe exakt einem String aus der Liste entspricht, und diese Eingabe dann zurückgibt.

Die Funktion soll dabei *exakt* die folgende Ausgabe erzeugen, wenn die Benutzereingaben 'klaiefh', 'yes!!!' und 'yes' getätigt werden:

```
>>> answer = input_choice('Do you want to play a game?', ['yes', 'no'])
Do you want to play a game? [yes | no]
> klaiefh
Invalid answer. Try again.
> yes!!!
Invalid answer. Try again.
> yes
```

Die Variable `answer` bekommt dabei den Wert 'yes' zugewiesen.

- (b) Schreiben Sie eine Funktion `shape`, die das geheime Wort `word` und einen String der bisher erratenen Zeichen `guesses` als Argumente nimmt, alle Zeichen in `word` durch '_' ersetzt, die nicht in `guesses` enthalten sind, und diesen String dann zurückgibt.

Beispiele:

```
>>> shape('weather', '')
'_____'
>>> shape('weather', 'e')
'_e___e_'
>>> shape('weather', 'eth')
'_e_the_'
```

- (c) Schreiben Sie eine Funktion `hangman` die das geheim Wort `word` und die Anzahl der erlaubten Fehlversuche als Eingabe nimmt und mit Ihnen eine Runde Hangman spielt. Verwenden Sie dabei eine `while`-Schleife um wiederholt zum Raten aufzufordern bis die Anzahl der erlaubten Fehlversuche überschritten wurde. In jedem Schleifendurchlauf soll dabei mindestens die folgenden Ausgaben gemacht werden:

- das aktuelle “`shape`” des geheimen Wortes, z.B. `'_e_the_'`;
- die Anzahl der Fehlerversuche die noch erlaubt sind;
- eine Aufforderung erneut zu raten, falls dies noch erlaubt ist.

Desweiteren soll an der Ausgabe erkennbar sein, ob das Spiel gewonnen oder verloren wurde.

Ein Aufruf von `hangman('weather', 5)` könnte also wie folgt aussehen:

```
_____; 5 mistakes left; make a guess: e
_e__e_; 5 mistakes left; make a guess: i
_e__e_; 4 mistakes left; make a guess: a
_ea__e_; 4 mistakes left; make a guess: th
_eathe_; 4 mistakes left; make a guess: th
_eathe_; 4 mistakes left; make a guess: weather
```

You won! The word was 'weather'! You're the best! Everyone loves you!

- (d) Fügen Sie z.B. folgenden Code² zu Ihrer Implementierung hinzu um ein vollständiges Hangman-Spiel zu erhalten:

```
# Importieren des Moduls für das Generieren von Zufallszahlen
import random

def input_choice ... # Aufgabenteil (a)

def shape ...      # Aufgabenteil (b)

def hangman ...    # Aufgabenteil (c)

words = [ 'apple', 'tree', 'python', 'bench', 'float' ]

max_fails = int(input("Number of allowed mistakes: "))

while input_choice("Wanna play a game?", ['yes', 'no']) == 'yes':
    word = random.choice(words) # Wähle ein zufälliges Wort aus.
    hangman(word, max_fails)
```

²Den Code gibt es auch zum Download unter <http://proglang.informatik.uni-freiburg.de/teaching/info1/2020/exercise/sheet05/hangman.py>

Aufgabe 5.2 (Gruppenaufgabe: Mandelbrot; Datei: `mandelbrot.py`; Punkte: 4+3+0+2)

Die Mandelbrot-Menge M ist die Menge aller komplexen Zahlen $c \in \mathbb{C}$, sodass die folgende Zahlenfolge beschränkt bleibt:

$$\begin{aligned}z_0 &= 0 \\z_{n+1} &= z_n^2 + c\end{aligned}$$

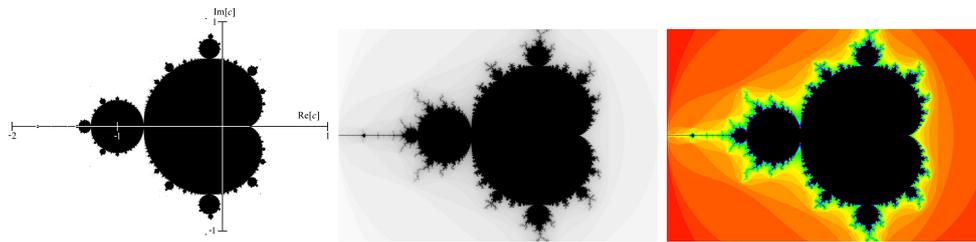
Die Folge gilt dabei als *beschränkt*, wenn jede Zahl der Folge innerhalb des Kreises mit Radius 2 um den Ursprung liegt, d.h. wenn für alle $n \in \mathbb{N}$ gilt, dass $|z_n| \leq 2$. Um uns kürzer ausdrücken zu können, nennen wir eine Zahl $c \in \mathbb{C}$ beschränkt, wenn die zu c gehörende Folge beschränkt ist.

Zum Beispiel ist $0.5 + 0i$ nicht beschränkt, da $|z_5| = |3.1533 + 0i| = 3.1533 > 2$, wobei z_5 wie folgt berechnet werden kann:

$$\begin{aligned}z_0 &= 0 \\z_1 &= z_0^2 + c = 0^2 + 0.5 = 0.5 \\z_2 &= z_1^2 + c = 0.5^2 + 0.5 = 0.75 \\z_3 &= z_2^2 + c = 0.75^2 + 0.5 = 1.0625 \\z_4 &= z_3^2 + c = 1.0625^2 + 0.5 = 1.6289 \\z_5 &= z_4^2 + c = 1.6289^2 + 0.5 = 3.1533\end{aligned}$$

Ob ein $c \in \mathbb{C}$ beschränkt ist, kann im Allgemeinen nicht in endlicher Zeit berechnet werden: wenn überprüft wurde, dass die ersten n Zahlen der Folge innerhalb des Kreises liegen, könnte z_{n+1} trotzdem außerhalb des Kreises liegen - unabhängig davon wie groß n gewählt wurde.

Die Beschränktheit kann aber näherungsweise bestimmt werden - ähnlich dem Newton-Verfahren. Hierzu wird eine maximale Zahl m festgelegt und überprüft ob die ersten m Zahlen der Folge innerhalb des Kreises liegen. Ist dies der Fall, so wird einfach angenommen, dass dies auch für den Rest der Folge gilt und die Folge daher beschränkt ist. Je größer m gewählt wird, desto höher ist der Rechenaufwand, aber auch die Chance unbeschränkte Folgen nicht fälschlicherweise als beschränkt zu erkennen.



(a) die beschränkten Zahlen in schwarz (b) mit Einfärbung der nicht-beschränkten Zahlen (c) mit Gradienten für die nicht-beschränkten Zahlen

Abbildung 1: Visualisierung der Mandelbrot-Menge

Berechnet man mit diesem Verfahren welche Zahlen $c \in \mathbb{C}$ zur Mandelbrot-Menge gehören und zeichnet diese dann auf der komplexen Ebene als schwarze Punkte ein, so ergibt sich ein Bild wie in Abbildung 1a.

Für die komplexen Zahlen, die nicht beschränkt sind, kann das kleinste n mit $|z_n| > 2$ bestimmt werden. Färbt man dann die nicht-beschränkten Zahlen so ein, dass die Helligkeit von n abhängt, so ergibt sich ein Bild wie in Abbildung 1b.

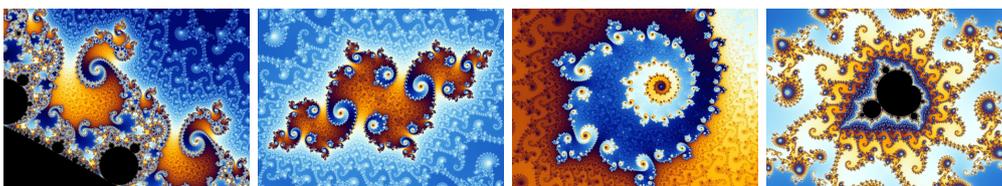
Wird nicht die Helligkeit von n abhängig gemacht, sondern je nach n ein Farbton auf einem Gradienten ausgewählt, so ergibt sich ein Bild wie in Abbildung 1c.



Das Tolle an der Mandelbrotmenge ist, dass man an jede Stelle beliebig nahe heranzoomen kann. Insbesondere am Rand der Mandelbrotmenge passieren dabei sehr interessante Dinge: da die Folgen z_n chaotisches Verhalten aufweisen, entstehen dort immer wieder neue unvorhersehbare Muster, wie man z.B. in Abbildung 2 sieht.

Aber ein Video³ illustriert das viel besser als ein paar einzelne Bilder.

³<https://www.youtube.com/watch?v=u1pwtSBTnPU>



(a) Zoom 1 (b) Zoom 2 (c) Zoom 3 (d) Zoom 4

Abbildung 2: Variationen in der Mandelbrot-Menge

In dieser Aufgabe sollen Sie ein Programm schreiben, welches ein Bild wie in Abbildung 1c erzeugt. Gehen Sie dabei wie folgt vor:

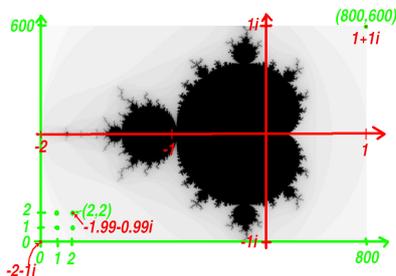
- (a) Schreiben Sie eine Funktion `mandelbrot`, die eine komplexe Zahl c und eine ganze Zahl m als Argumente nimmt und das kleinste $0 \leq i < m$ zurückgibt, sodass $|z_i| \geq 2$ ist. Gibt es so ein i nicht, so soll m selbst zurückgegeben werden.

Beispiel aus der Aufgabeneinleitung:

```
>>> mandelbrot(0.5, 50) # 0.5 ist nicht in der Mandelbrotmenge,
5                          # da 5 < 50.
>>> mandelbrot(0.5, 4)  # 0.5 ist näherungsweise in der
4                          # Mandelbrotmenge, da 4 == 4.
                          # Hier schlägt die Approximation fehl,
                          # da m zu klein gewählt wurde.
```

- (b) Digitale Bilder bestehen aus einer Matrix von Farbpunkten - den Pixeln. Wenn wir also ein Bild mit Auflösung 800x600 (800 Pixel breit, 600 Pixel hoch) berechnen wollen, so müssen wir für jedes einzelne der 480000 Pixel einen Farbwert berechnen. Jedes Pixel wird durch ein Paar von ganzzahligen Koordinaten (x, y) adressiert, wobei $0 \leq x < 800$ und $0 \leq y < 600$.

Wir möchten jetzt jedem Pixel (grün) einen Punkt (rot) in einem Intervall $[a + bj, c + dj)$ der komplexen Zahlen zuordnen. Damit dies wohldefiniert ist, verlangen wir $a < c$ und $b < d$.



Schreiben Sie nun eine Funktion `sample`, sodass `sample(a+bj, c+dj, x, y, sx, sy)` für alle gültigen Pixel-Koordinaten (x, y) eines Bildes der Auflösung $sx \times sy$ eine entsprechende komplexe Zahl aus dem o.g. Intervall liefert.

Beispiele für beliebige Gleitkommazahlen a, b, c, d :

```
sample(a+bj, c+dj, 0, 0, 800, 600) == a+bj # unten-links
sample(a+bj, c+dj, 0, 600, 800, 600) == a+dj # oben-links
sample(a+bj, c+dj, 800, 0, 800, 600) == c+bj # unten-rechts
sample(a+bj, c+dj, 800, 600, 800, 600) == c+dj # oben-rechts
sample(a+bj, c+dj, 400, 300, 800, 600) == 0.5*(a+bj) + 0.5*(c+dj) # mittig
```

- (c) Installieren Sie die `pillow`-Bibliothek. Diese stellt Funktionalität bereit um Bilder zu erstellen, zu manipulieren und abzuspeichern.

Sie können die Bibliothek installieren, indem Sie auf der Kommandozeile folgenden Befehl ausführen:

```
python3 -m pip install pillow
```

Testen Sie anschließend, ob die Bibliothek gefunden wird, indem Sie folgenden Code in die Datei `pillow_test.py` schreiben⁴ und ausführen:

```
# Die pillow-Bibliothek importiert das PIL-Modul (Python Image Library)
from PIL import Image

# Erstelle ein Bild mit Auflösung 800x600 wobei die einzelnen
# Pixel das HSV-Farbformat verwenden (Hue-Saturation-Value).
size = (800, 600)
img = Image.new('HSV', size)

# Setze die Farbe von jedem Pixel auf rot.
for x in range(size[0]):
    for y in range(size[1]):
        # Bei Interesse siehe HSV-Farbraum auf Wikipedia.
        # Sie müssen für die Aufgabe nicht verstehen, wieso
        # folgendes Tupel der Farbe Rot entspricht.
        red = (255, 255, 255)
        img.putpixel((x,y), red)

# Speichere das Bild als JPEG-Datei im aktuellen Verzeichnis.
# Wir konvertieren das HSV-Farbformat zu RGB (Red-Green-Blue),
# da JPEG kein HSV unterstützt.
img.convert('RGB').save('my_red_image.jpg', quality=95)
```

Es sollte sich nun ein Bild `my_red_image.jpg` der Auflösung 800x600 in dem Ordner befinden wo sie `pillow_test.py` ausgeführt haben.

- (d) Schreiben Sie eine Funktion `render_mandelbrot`, die ein Bild der Mandelbrotmenge generiert. Ruft man die Funktion dabei wie folgt auf, so soll das Bild aus Abbildung 1c erstellt werden:

```
>>> render_mandelbrot(
    -2-1j, 1+1j, # Intervall auf der komplexen Ebene.
    900, 600,   # Auflösung des zu erzeugenden Bildes.
    50,        # Anzahl maximaler Schleifendurchläufe.
    'output.jpg') # Dateiname des zu erzeugenden Bildes.
```

Verwenden Sie hierfür den Beispielcode aus dem vorherigen Aufgabenteil als Grundgerüst und die `sample`-Funktion, um die Pixelkoordinaten auf die komplexe Ebenen zu projizieren, bevor Sie dann die `mandelbrot`-Funktion anwen-

⁴Den Code gibt es auch als Download unter http://proglang.informatik.uni-freiburg.de/teaching/info1/2020/exercise/sheet05/pillow_test.py

den.

Um den Rückgabewert der `mandelbrot`-Funktion in eine HSV-Farbe umzuwandeln, können Sie dabei folgende Funktion⁵ verwenden:

```
def color(i: int, max_i: int) -> (int, int, int):
    # Farbton in Abhängigkeit der benötigten Schleifendurchläufe.
    hue = int(255 * (i / max_i))

    # Volle Helligkeit 255, außer wenn c Teil der Mandelbrotmenge ist.
    # Dadurch wird das innere schwarz.
    value = 255 if i < max_i else 0

    # Volle Sättigung
    saturation = 255

    return (hue, saturation, value)
```

Aufgabe 5.3 (Erfahrungen; Datei: `erfahrungen.txt`; 2 Punkte)

Notieren Sie hier Ihre Erfahrungen mit diesem Übungsblatt (benötigter Zeitaufwand, Probleme, Bezug zur Vorlesung, Interessantes, etc.).

Schreiben Sie den groben Zeitaufwand, den Sie für das *gesamte* Blatt benötigt haben, bitte wie folgt in die erste Zeile:

Zeitaufwand: 3.5h

[... Freitext, wie bisher ...]

Wenn sich genug Leute daran halten, dann können wir ein Pythonskript schreiben, welches automatisiert die durchschnittliche Bearbeitungszeit berechnet ;-)

⁵Den Code gibt es auch als Download unter <http://proglang.informatik.uni-freiburg.de/teaching/info1/2020/exercise/sheet05/color.py>