Aufgaben zu Python Grundlagen

Aufgabe 1 – Funktionsdefinition mit Turtle Graphics

Um diese Aufgabe zu lösen, solltest Du Python-Programme ausprobieren können und verstanden haben, wie man Funktionen mit Parametern definiert und diese mit geeigneten Argumenten aufruft. Ausserdem solltest Du einfache Schleifen verstehen.

Manche Einführungen in die Programmierung (u.a. die von *Brilliant*) beginnen mit einfachen Zeichenprogrammen mittels sogenannter *Turtle Graphics*. Dabei wird eine «virtuelle Schildkröte» mit einfachen Befehlen über eine zweidimensionale Zeichenebene dirigiert, und der zurückgelegte Weg aufgezeichnet.

Der Python-Interpreter enthält für solche Programme das Modul *turtle*. Die darin implementierten Operationen sind leicht verständlich und die Namen sprechen für sich selbst.[1]

Betrachte das folgende Turtle-Graphik-Programm:

```
 1 from math import pi
 2 from turtle import *
 3
 4 left(90)
 5 forward(250)
 6 right(90)
 7 for i in range(18):
 8    forward(100 * pi / 36)
 9    right(10)
10 forward(100 * pi / 36)
11 left(180)
12 for i in range(18):
13    forward(150 * pi / 36)
14    right(10)
15 forward(150 * pi / 36)
16 left(180)
17 done()       # prevents turtle drawing window from closing immediately
```

a) Das Programm zeichnet einen Buchstaben. Kannst Du – zunächst ohne das Programm ausführen zu lassen – herausfinden, welcher das ist? Dazu kannst Du z.B. das Programm selbst mit Stift und Papier ausführen. Es genügt eine ungefähre Skizze. Du brauchst Strecken und Winkel nicht exakt auszumessen.

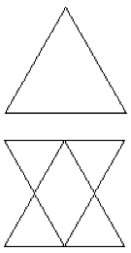
   Überprüfe Deine Analyse, indem Du den Computer das Programm ausführen lässt.

b) Der Abschnitt in Zeile 7 bis 11 ist dem in Zeile 12 bis 16 sehr ähnlich. Eine solche Situation eignet sich, um Programmteile «heraus zu faktorisieren», d.h. dafür eine *Funktion* zu definieren. Damit muss nur ein solcher Programm-Abschnitt programmiert werden, der mehrmals aufgerufen werden kann.

   Allerdings unterscheiden sich die beiden Programmabschnitte in einem Aspekt. Deswegen braucht die Funktion einen *Parameter*. Schreibe das obige Programm so um, dass es eine geeignete von Dir programmierte Funktion benutzt.

Aufgabe 2 – Gleichseitige Dreiecke

Um diese Aufgabe zu lösen, solltest Du die allerwichtigsten Funktionen der Turtle Graphik verwenden können (s. Aufgabe 1). Ausserdem brauchst Du etwas Geometrie-Wissen.



a) Programmiere für die Turtle Graphik einen Ablauf, der ein *gleichseitiges Dreieck* zeichnet.

b) Erweitere Dein Programm so, dass über das Dreieck aus Teilaufgabe a) ein weiteres Dreiecke gespiegelt und gleich gross gezeichnet wird.

   Wenn dazu Dein Programm die Turtle an einen neuen Ausgangspunkt bewegen soll, ohne dass die Spur dieser Bewegung gezeichnet wird, kann Dein Programm die Turtle-Operationen `penup()` und `pendown()` verwenden, um das Zeichnen aus- bzw. wieder einzuschalten.

[1] Ansonsten: Dokumentation zum Nachschlagen findet sich unter https://docs.python.org/3/library/turtle.html

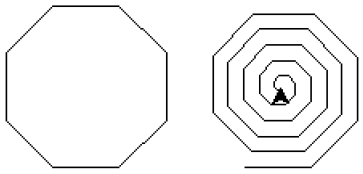
Aufgabe 3 – Oktagon-Spirale

Um diese Aufgabe zu lösen, solltest Du verstanden haben, wie man Funktionen mit Parametern definiert und diese mit geeigneten Argumenten aufruft. Ausserdem solltest Du einfache Schleifen und Berechnungen mit Variablen programmieren können.

Das Programm unten zeichnet ein Oktagon, wie im Bild links und soll nur als Denkanstoss dienen.

Programmiere eine Funktion, die eine *oktogonale Spirale* wie im Bild rechts zeichnet. Deine Funktion soll mit der Länge der längsten Kante parametrisiert werden können (im Bild 100), sowie mit einem Wert, der sagt, wieviel kürzer jeweils die nächste Strecke auf der Spirale sein soll (im Bild 1).

```
 1 from turtle import *
 2
 3 def octagon(width):
 4    edge_len = width / (1 + 2**0.5)
 5    for i in range(8):
 6       forward(edge_len)
 7       left(45)
 8
 9 octagon (100)
10 done()       # prevents turtle drawing window from closing immediately
```



Aufgabe 4 – Slicing mit Strings

Um diese Aufgabe zu lösen, solltest Du Strings, Slicing und Funktionsdefinitionen mit Resultaten verwenden können.

a) Programmiere eine Funktion `rotate_left_str` mit zwei Parametern: einem String *s* und einer Zahl *n*. Das Resultat der Funktion soll der um *n* Positionen nach links rotierte String *s* sein. D.h., die *n* ersten Zeichen von *s* befinden sich neu an dessen Ende. Beispielsweise soll ‘`abcdefgh`’ rotiert um 3 Positionen ‘`defghabc`’ ergeben. Falls *s* nicht mehr als *n* Zeichen enthält, sei das Resultat wieder *s* selbst.

b) Programmiere eine Funktion `move2_str`, die alle Zeichen im String mit einem *ungeraden Index* in der ersten Hälfte und alle mit einem *geraden Index* in der zweiten String-Hälfte sammelt. Aus ‘`abcdefgh`’ würde so ‘`bdfhaceg`’. Der Index *0* gilt als *gerade*.

Aufgabe 5 – str.count('x') selbst programmiert

Um diese Aufgabe zu lösen, solltest Du Strings, Slicing, Wiederholungsanweisungen und Funktionsdefinitionen mit Resultaten verwenden können.

Die Sprache Python bietet für Strings (u.a.) eine Funktion `count`, die zählen kann, wie oft ein String in einem anderen enthalten ist. So ergibt beispielsweise `"Das ist ein netter Versuch".count("er")` das Resultat 2.

a) Programmiere selbst eine solche Funktion `str_count(string, pattern)`, die zählt, wie oft `pattern` in `string` enthalten ist. Verwende für diese Aufgabe *keine String-Operationen* ausser *Indexierung*, *Slicing* und *Vergleichen*. Ausserdem darfst Du die Funktion `len` benutzen, damit Dein Programm die Länge der Strings herausfinden kann.

b) Programmiere noch eine andere Zählfunktion `str_count_any(string, pattern)`, die den zweiten Parameter als Menge einzelner Zeichen interpretiert und zählt, wie viele Zeichen des ersten Parameter-Strings irgendeinem Zeichen dieser Menge entsprechen.

   Das Resultat eines Aufrufs `str_count_any("Das ist ein netter Versuch", "er")` wäre dann 6.

Aufgabe 6 –max(lst) selbst programmiert

Um diese Aufgabe zu lösen, solltest Du mit Listen, Schleifen, Fallunterscheidungen und Variablen umgehen können.

Programmiere eine Funktion `max_index`, die den *Index eines grössten Elements* in einer als Argument angegebenen Liste ermittelt. Nimm dazu an, dass alle Elemente in der Liste miteinander mittels den üblichen Vergleichsoperatoren (<, >, <=, >=, ==, !=) miteinander vergleichbar sind. Python verfügt bereits über die Funktion `max`, die ein maximales Element selbst sucht, aber die sollst Du hier *nicht* verwenden. Es geht darum, zur Übung ein solches Verfahren einmal selbst zu programmieren. Ausserdem ist ja in dieser Aufgabe nicht das Element selbst, sondern dessen Index in der Liste gefragt.

Aufgabe 7 – String-Zerlegung

Um diese Aufgabe zu lösen, solltest Du mit Strings umgehen können, Operationen auf Strings wie strip, split *und* Slicing *kennen.*

Programmiere eine Funktion, die einen als Argument angegebenen String zerlegt, der folgende Struktur hat: Zwei Zahlen, getrennt durch ein '-', nach einem Leerzeichnen ein einzelner Buchstabe gefolgt von einem Doppelpunkt und nach einem weiteren Leerzeichen beliebige Buchstaben bis zum String-Ende.

Ein Beispiel könnte so aussehen: «`32-165 k: sdklffs`». Das Resultat der Zerlegung soll ein Tupel oder eine Liste sein mit den einzelnen Elementen aus dem String: `(32, 165, "k", "sdklffs")`.

Aufgabe 8 – Liste aller ganzen Zahlen i < N, bei denen i^2 mod 5 = 4

Um diese Aufgabe zu lösen, solltest Du Listen erzeugen und z.B. in einer Schleife gezielt einzelne Elemente hinzufügen können, wenn diese eine bestimmte Eigenschaft haben.

Programmiere eine Funktion, die für eine als Argument angegebene Zahl *N* eine Liste aller Zahlen *i* produziert, für die gilt $0 \le i < N$ und $i^2 \bmod 5 = 4$.

Aufgabe 9 – Turtle-Programm-Interpreter

Um diese Aufgabe zu lösen, solltest Du Wiederholungsanweisungen programmieren und mit Listen (und allenfalls mit Tupeln) umgehen können.

Einfache zusammenhängende Linienzüge mit der Turtle-Graphik lassen sich als Folge (Liste) von jeweils einer Drehung – zum Ausrichten der Turtle – und einer Linienlänge beschreiben. Z.B. beschreibt die Liste [(0, 30), (60, 50), (-120, 50), (60, 30)] die nebenstehende Graphik, unter der Annahme, dass die erste Zahl der Tupel jeweils eine Drehung nach links und die zweite Zahl die danach vorwäts zu zeichnende Distanz angibt.

Programmiere eine Funktion, die ein solches einfaches «Turtle-Programm» – also eine Liste der obigen Art – interpretiert, indem es die entsprechende Zeichnung produziert.

Aufgabe 10 – Kaprekar-Zahlen

Um diese Aufgabe zu lösen, solltest Du Variablen-Typen int *und* str *kennen und Werte zwischen solchen Typen umzuwandeln können.*

Programmiere eine Funktion oder ein Python Script, um anhand einzelner Beispiele zu überprüfen, dass 6174 die Kaprekar-Konstante[2] für vierstellige Zahlen ist.

Die spezielle Eigenschaft dieser Kaprekar-Konstanten ist, dass folgendes Verfahren immer irgendwann diese Zahl produziert:

1. Beginne mit einer beliebigen *4-stelligen* Zahl, die *nicht* aus lauter gleichen Ziffern besteht. Wenn diese Zahl kleiner als 1000 ist, ergänze sie mit führenden 0en auf 4 Stellen.
2. Berechne die Differenz zwischen derjenigen Zahl, die sich ergibt, wenn man die 4 Ziffern (ggf. einschliesslich führender 0en) *absteigend geordnet* aufschreibt und jener Zahl, die sich ergibt, wenn man die 4 Ziffern *aufsteigend geordnet* aufschreibt.

   Für die Zahl 9273 beispielsweise wäre das 9732 – 2379 = 7353.
3. Wenn sich durch diese Rechnung eine andere Zahl als die zuletzt errechnete ergeben hat, wiederhole Schritt 2, ggf. nach Ergänzung führender 0en auf 4 Stellen.

   In unserem Beispiel wäre also als nächstes zu berechnen 7533 – 3357 = 4176.

Dieses Verfahren endet immer mit der Zahl 6174, der Kaprekar-Konstanten für 4-stellige Zahlen.

Um zu sortieren, kannst Du die in Python integrierte Funktion *sorted* verwenden. Das Ergebnis ist eine geordnete Liste einer als Argument angegebenen Sequenz (das kann auch ein String sein). Elemente einer Liste können mit `"".join(lst)` zu einem String verbunden werden.

[2] s. z.B. https://de.wikipedia.org/wiki/Kaprekar-Konstante