11_Uebungsaufgaben

0.1 Vorschläge für Übungsaufgaben

Die folgenden Übungsaufgaben sind ausführlicher als die unterwegs vorgeschlagenen Mini-Übungen. Das Ziel ist es hier, nicht nur einzelne Themen direkt auszuprobieren, sondern vielmehr verschiedene Themen zu kombinieren, um so ein verknüpfendes Verständnis der Lerninhalte zu erreichen.

0.1.1 Zahlen raten

Diese Aufgabe trainiert Schachtelungen, Schleifen und bedingte Ausführungen.

Schreiben Sie ein Programm, indem eine zufällige natürliche Zahl zwischen 1 und 100 erzeugt wird. Mit dem Befehl randint(a,b) können Sie eine zufällige ganze Zahl zwischen a und b erzeugen:

[1]:
```
import random

random_int = random.randint(1,100)
print(random_int)
```

```
46
```

Der/Die Benutzer*in soll dann aufgefordert werden diese Zahl zu erraten. Wenn die Zahl nicht erraten wurde, soll ein Hinweis gegeben werden ob die gesuchte Zahl kleiner oder größer ist und es soll weiter geraten werden bis die Zahl erraten wurde. Sie können davon ausgehen, dass der/die Benutzer*in eine natürliche Zahl eingibt. Falls die Eingabe keine Zahl zwischen 1 und 100 ist, soll die/der Benutzer*in es erneut probieren. Sobald das Frustrationslevel hoch genug ist, beendet die Eingabe von -1 das Programm.

[2]:
```
# ...
```

0.1.2 Sieb des Eratosthenes

Diese Aufgabe perfektioniert die verschiedenen Primzahlbeispiele aus der Lerneinheit.

Das Sieb des Eratosthenes ist eine Methode zur Bestimmung einer Liste aller Primzahlen kleiner gleich einer vorgegebenen Zahl n. Dazu werden die Zahlen $2, 3, 4, ..., n$ aufgeschrieben. Alle zunächst unmarkierten Zahlen sind potentielle Primzahlen. Die kleinste unmarkierte Zahl ist immer eine Primzahl. Nachdem eine Primzahl gefunden wurde, werden alle Vielfachen dieser Primzahl markiert. Dann bestimmt man die nächstgrößere nicht markierte Zahl. Da sie kein Vielfaches von Zahlen kleiner als sie selbst ist (sonst wäre sie markiert worden), kann sie nur

durch 1 und sich selbst teilbar sein. Folglich muss es sich um eine Primzahl handeln. Man streicht wieder alle Vielfachen und führt das Verfahren analog fort. Sobald man bei einer Primzahl p ankommt, für welche gilt: $p^2 > n$, sind alle zusammengesetzten Zahlen markiert und alle nicht markierten Zahlen sind Primzahlen.

Beispiel: Im ersten Durchlauf wird die 1 als Primzahl identifiziert. Im zweiten Durchlauf wird die 2 identifiziert, und es werden alle geraden Zahlen als nicht prim markiert, usw.

Implementieren Sie das oben beschriebene Verfahren. Tipp: Verwenden Sie zwei Listen mit identischer Indizierung, eine mit den Zahlen und eine mit bools, die kennzeichnen, ob die zugehörige Zahl eine Primzahl ist.

[3]:
```
number = 20

# ...
```

0.1.3 Selection Sort

Verstehen Sie den Algorithmus bspw. ausgehend von der zugehörigen Wikipedia-Seite und implementieren Sie ihn. Sie dürfen sich das Leben erheblich dadurch vereinfachen, dass Sie den Algorithmus nicht "in-place" realisieren, sondern sukzessive Elemente von der Eingabeliste in die neue Liste verschieben, bis letztere voll oder erstere leer ist. Außerdem sparen Sie sich Arbeit, wenn Sie sich vorab schlau machen, was die Funktionen list.min() und list.index(element) genau tun.

[4]:
```
list_input = [10,30,30,2,25,100,13]
print(list_input)
```

```
[10, 30, 30, 2, 25, 100, 13]
```

0.2 Impressum

0.2.1 Programmierkurs Python, Dominik Göddeke https://www.ians.uni-stuttgart.de, Universität Stuttgart

Version vom April 2023

Gefördert mit Mitteln der Deutschen Forschungsgemeinschaft (EXC 2075 - 390740016) im Rahmen der Exzellenzstrategie.

[ ]: