11_Uebungsaufgaben_Loesungen

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:

[22]:
```
import random

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

```
23
```

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]:
```
import random

secret_number = random.randint(1,100)
# secret_number = 42 # während der Entwicklungsphase

number_min = 1
number_max = 100
number_exit = -1

# Annahme: Benutzer*in rät falsch
correct_guess = False
```

```
while not correct_guess:
    # Eingabevalidierung
    guess = -1234567
    while True:
        guess = int(input("Enter your guess: "))
        if guess == number_exit:
            break
        if guess >= number_min and guess <= number_max:
            break
        else:
            print("Please try again with a natural number between 1 and 100")
    # Ergebnisbeurteilung
    if guess == number_exit:
        # geschachtelte Schleifen, also müssen wir hier nochmal raus
        print("Sorry that you gave up")
        break
    if guess < secret_number:
        print("Your guess is below the secret number")
    elif guess > secret_number:
        print("Your guess is above the secret number")
    else:
        print("Congratulations, you nailed it")
        break
```

```
Enter your guess: 42
Your guess is below the secret number
Enter your guess: 78
Your guess is above the secret number
Enter your guess: 66
Your guess is above the secret number
Enter your guess: 50
Your guess is above the secret number
Enter your guess: 47
Your guess is below the secret number
Enter your guess: -1
Sorry that you gave up
```

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 stre-

icht 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.

[5]:
```
number = 20

# Listen der Zahlen und der prim-Eigenschaft per comprehension
# um uns mit den Indizes nicht zu verwirren
list_values = [i for i in range(number+1)]
list_prime = [True]*len(list_values)
# die letzte letzung stellt sicher, dass 2 automatisch bereits als Primzahl␣
 ↪erkannt ist

# äußerste Schleife von 3 bis Ende, mit range(), damit wir beide Listen␣
 ↪identisch indizieren können
# Verwirrungsfaktor: die Zahl 3 steht an Index 1 in unseren Listen
for i in range(len(list_values)+1):
    # Streichen möglicher Vielfacher und Sonderfälle
    if list_prime[i] and i != 0 and i != 1:
        # wenn wir keine Primzahl haben, müssen wir nichts tun
        # wenn doch, dann iterieren wir mit der aktuellen Zahl als Schrittweite
        print("found prime", list_values[i])
        for j in range(i,number+1,i):
            # die erste Bedingung ist nötig, damit nicht die frisch gefundene␣
 ↪Primzahl
            # direkt wieder als nicht prim markiert wird
            # die zweite Bedingung ist nötig, um Zugriffe über das Ende der␣
 ↪Liste hinaus zu vermeiden
            if j != i and j <= number:
                list_prime[j] = False
        # es fehlt noch die Abbruchbedingung
        if list_values[i]**2 > number:
            break
    # else ist wegen unserer Initialisierung nicht nötig

# Ergebnis
print("all other primes found by exclusion:")
for i in range(2,len(list_values)):
    if list_prime[i]:
        print(list_values[i])
```

```
found prime 2
```

```
found prime 3
found prime 5
all other primes found by exclusion:
2
3
5
7
11
13
17
19
```

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.

[5]:
```
list_input = [10,30,30,2,25,100,13]
print(list_input)

# Beachtung des Hinweis
list_sorted = []

# Der Algorithmus selbst ist einfach: Minimum und dessen Index
# bestimmen, und dann mit dem Index löschen und den Wert hinzufügen
# an der richtigen Stelle, nämlich dem Listenende
while len(list_input) > 0:
    min_value = min(list_input)
    min_index = list_input.index(min_value)
    del list_input[min_index]
    list_sorted.append(min_value)

print(list_input)
print(list_sorted)
```

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

Diese Variante läuft jedoch zu oft durch die Listen. Eigentlich sollte ein Listendurchlauf ausreichen, um **simultan** das Minimum und dessen Index zu bestimmen. Das sieht dann so aus:

[9]:
```
list_input = [10,30,30,2,25,100,13]
print(list_input)

list_sorted = []

while len(list_input) > 0:
    # Minimumbestimmung: Initialisierung mit erstem Listenelement
    min_value = list_input[0]
    min_index = 0
    # Minimumbestimmung für den Rest der Liste
    for index in range(1,len(list_input)):
        if list_input[index] < min_value:
            min_value = list_input[index]
            min_index = index
    del list_input[min_index]
    list_sorted.append(min_value)

print(list_input)
print(list_sorted)
```

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

0.2 Impressum

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

Version vom April 2023

Gefördert durch die Stiftung Innovation in der Hochschullehre. (alle Rechte am Logo vorbehalten)

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

[ ]: