

Aufgabenblatt 4

Das sind die Aufgaben der Woche 4. Es handelt sich um Loops (Schleifen) in Programmen, also for und while.

Aufgabe 16 - Schleifen

Implementieren Sie Python-Code für eine Schleife, die für jede 3er-Potenz die letzten 3 Ziffern der zugehörigen Dezimalzahl ausgibt. Die Anzahl der Schleifendurchläufe soll hierbei durch eine Nutzereingabe festgelegt werden. **(2 P.)**

Was ist die kleinste Dreierpotenz grösser 3, die auf „001“ endet? **(1 P.)**

Können Sie das gleiche Problem auch für 7er-Potenzen lösen? **(1 P.)**

Lösung:

Die Lösung ist in diesem Screenshot zu sehen, wie auch im Jupyter Notebook oder im Aufgabe16.py File (befindet sich im .zip) zu finden.

```

19 #%%
20
21 num = 3
22 anzLoops = int(input("Wie oft soll die Zahl " + str(num) + " hoch gerechnet werden?")) num: 3
23 index = 1
24
25 result = basis = num num: 3
26 exponent = firstToEnd = 1
27
28 # Teil 1
29 # Durchläufe bis Eingabe
30 while index <= anzLoops: index: 101 anzLoops: 100
31     result = basis ** exponent basis: 3 exponent: 101
32     print(str(index) + ". Durchlauf: [" + "letzte drei Stellen: " + str(result % 1000) + "]") in
33     if str(result).endswith("001"): result: 515377520732011331036461129765621272702107522001
34         firstToEnd = exponent exponent: 101
35     exponent += 1 exponent: 101
36     index += 1 index: 101
37
38 # Teil 2
39 if firstToEnd != 1: firstToEnd: 100
40     print("Dieser Exponent zur Basis " + str(basis) + " endet mit 001: " + str(firstToEnd)) basi
41 else:
42     print("Die eingegebene Anzahl an Loops ist zu Klein.\U0001F612") # Der Code ist ein "Lätsch"
43
44 # Teil 3:
45 # 3 hoch 100 endet auf 001
46 # 7 hoch 20 endet auf 001
  
```

Abbildung 1 - Aufgabe 16

Wichtig zu vermerken ist:

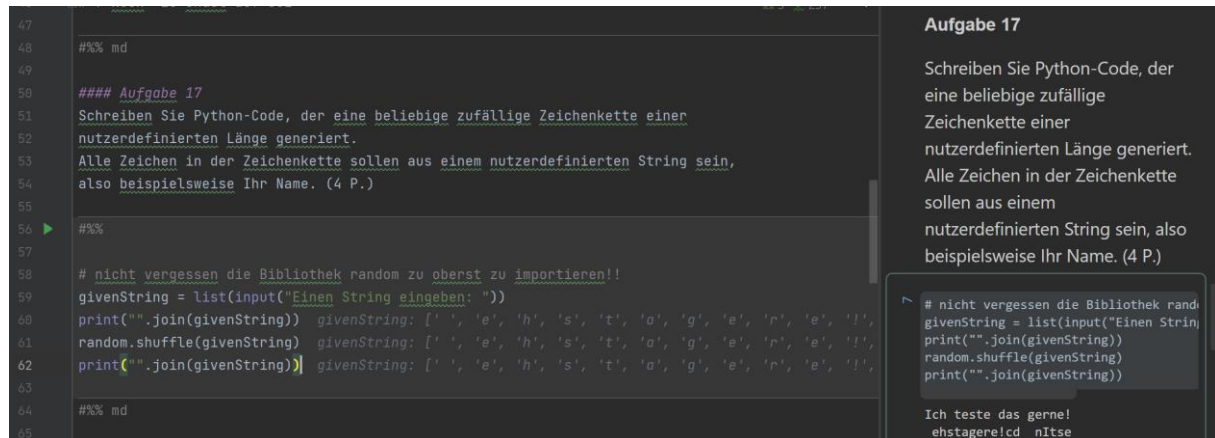
- Im Teil 2 am Ende wird, falls die Zahl zu klein ist, ein Emoticon ausgegeben.
- Im Aufgabe16.py File ist der Algorithmus in einer Funktion und am Ende 2-mal ausgegeben, einmal mit der Basis 3 und einmal mit der Basis 7
- Die Respektiven Lösungen zur Fragen 2 und 3 sind:
- Potenz von 3 welche auf 001 endet: 100
- Potenz von 7 welche auf 001 endet: 20

Aufgabe 17 - Zufallsstring

Schreiben Sie Python-Code, der eine beliebige zufällige Zeichenkette einer nutzerdefinierten Länge generiert. Alle Zeichen in der Zeichenkette sollen aus einem nutzerdefinierten String sein, also beispielsweise Ihr Name. **(4 P.)**

Lösung:

Auf dem Screenshot ist die Lösung zu sehen:



```
47
48 #%% md
49
50 ### Aufgabe 17
51 Schreiben Sie Python-Code, der eine beliebige zufällige Zeichenkette einer
52 nutzerdefinierten Länge generiert.
53 Alle Zeichen in der Zeichenkette sollen aus einem nutzerdefinierten String sein,
54 also beispielsweise Ihr Name. (4 P.)
55
56 #%%
57
58 # nicht vergessen die Bibliothek random zu oberst zu importieren!!
59 givenString = list(input("Einen String eingeben: "))
60 print("".join(givenString)) givenString: [' ', 'e', 'h', 's', 't', 'a', 'g', 'e', 'r', 'e', 'l', 'c', 'd', ' ', 'n', 'I', 't', 's', 'e']
61 random.shuffle(givenString) givenString: [' ', 'e', 'h', 's', 't', 'a', 'g', 'e', 'r', 'e', 'l', 'c', 'd', ' ', 'n', 'I', 't', 's', 'e']
62 print("".join(givenString)) givenString: [' ', 'e', 'h', 's', 't', 'a', 'g', 'e', 'r', 'e', 'l', 'c', 'd', ' ', 'n', 'I', 't', 's', 'e']
63
64 #%% md
65
```

Aufgabe 17

Schreiben Sie Python-Code, der eine beliebige zufällige Zeichenkette einer nutzerdefinierten Länge generiert. Alle Zeichen in der Zeichenkette sollen aus einem nutzerdefinierten String sein, also beispielsweise Ihr Name. (4 P.)

```
# nicht vergessen die Bibliothek rand
givenString = list(input("Einen Strin
print("".join(givenString))
random.shuffle(givenString)
print("".join(givenString))
```

Ich teste das gerne!
ehstagerelcd nItse

Abbildung 2 - Aufgabe 17

Zu beachten ist, dass am Anfang des Codes die Bibliothek random importiert werden muss und dass der eingegebene String als Liste gespeichert wird. Das ermöglicht das einfache shuffeln und joinen der einzelnen Charakter.

Es gibt noch weitere Bibliotheken, welche es ermöglichen eine Zeichenkette zu randomisieren bzw. zu shuffeln:

- Numpy
- String_utils
- More_itertools

Alle diese Bibliotheken besitzen eine shuffle Funktion.

Aufgabe 18

Berechnen Sie alle Produkte von $i*j*k$ für $0 < i \leq 10$, $0 < j \leq 15$, $0 < k \leq 20$, für die das Produkt $i*j*k$ eine Quadratzahl ist. Geben Sie die Lösungsprodukte aus. **Hinweis zur Einfachheit der Aufgabe:** Eine ganze positive Zahl ist eine Quadratzahl, wenn ihre Quadratwurzel eine ganze Zahl ist. **Alternativ:** Eine ganze positive Zahl ist eine Quadratzahl, wenn sie als Summe von aufeinanderfolgenden ungeraden Zahlen, beginnend bei 1 geschrieben werden kann (Beispiel: $1 + 3 + 5 + 7 + 9 = 25$). (4 P.)

Lösung:

Meine Lösung funktioniert folgenderweise:

1. Alle drei Variablen in eine for-Schleife setzen
2. Das resultat der drei Variablen berechnen (Zeile 88)
3. Das Resultat evaluieren
 - a. Wurzel bilden
 - b. Wurzel auf Ganzzahligkeit überprüfen
4. Ergebnis ausgeben

So werden ALLE Produkte berechnet und nur die Quadratischen ausgegeben. (siehe unten)

```

66 ##### Aufgabe 18
67 Berechnen Sie alle Produkte von i*j*k für 0 < i <= 10, 0 < j <= 15, 0 < k <= 20,
68 für die das Produkt i*j*k eine Quadratzahl ist.
69 Geben Sie die Lösungsprodukte aus.
70 Hinweis zur Einfachheit der Aufgabe:
71 Eine ganze positive Zahl ist eine Quadratzahl,
72 wenn ihre Quadratwurzel eine ganze Zahl ist.
73 Alternativ: Eine ganze positive Zahl ist eine Quadratzahl,
74 wenn sie als Summe von aufeinanderfolgenden ungeraden Zahlen,
75 beginnend bei 1 geschrieben werden kann (Beispiel: 1 + 3 + 5 + 7 + 9 = 25). (4 P.)
76
77 #%%
78
79 # Variable vordefinieren
80 resultat = 1
81
82 # range benutzt die beiden ersten Parameter nicht
83 # und die Inkrementierung ist default 1, deswegen 0 und 11 weil die Zahlen bis und mit 10 sind
84 for i in range(0, 11):
85     for j in range(0, 16):
86         for k in range(0, 21):
87             # Resultat wird berechnet
88             resultat = i * j * k
89
90             # Wurzel wird mit der Potenzierung von 0.5 errechnet
91             # und mit Modulo 1 == 0 wird überprüft ob das Ergebnis eine Ganzzahl ist.
92             if resultat != 0 and resultat**0.5%1 == 0:
93                 print("Quadratwurzel: " + str(resultat) + " : [i=\n" + str(i) + "\n", j=\n" + str(j) + "\n", k=\n" + str(k) + "\n")
94

```

Quadratwurzel: 1 : [i="1", j="1", k="1"]
 Quadratwurzel: 4 : [i="1", j="1", k="4"]
 Quadratwurzel: 9 : [i="1", j="1", k="9"]
 Quadratwurzel: 16 : [i="1", j="1", k="16"]
 Quadratwurzel: 4 : [i="1", j="2", k="2"]
 Quadratwurzel: 16 : [i="1", j="2", k="8"]
 Quadratwurzel: 36 : [i="1", j="2", k="18"]
 Quadratwurzel: 9 : [i="1", j="3", k="3"]
 Quadratwurzel: 36 : [i="1", j="3", k="12"]
 Quadratwurzel: 4 : [i="1", j="4", k="4"]
 Quadratwurzel: 16 : [i="1", j="4", k="16"]
 Quadratwurzel: 36 : [i="1", j="4", k="36"]
 Quadratwurzel: 64 : [i="1", j="4", k="64"]
 Quadratwurzel: 25 : [i="1", j="5", k="5"]
 Quadratwurzel: 100 : [i="1", j="5", k="20"]
 Quadratwurzel: 36 : [i="1", j="6", k="6"]
 Quadratwurzel: 49 : [i="1", j="7", k="7"]
 Quadratwurzel: 16 : [i="1", j="8", k="8"]
 Quadratwurzel: 64 : [i="1", j="8", k="32"]
 Quadratwurzel: 144 : [i="1", j="8", k="18"]
 Quadratwurzel: 9 : [i="1", j="9", k="9"]
 Quadratwurzel: 81 : [i="1", j="9", k="9"]
 Quadratwurzel: 144 : [i="1", j="9", k="16"]
 Quadratwurzel: 100 : [i="1", j="10", k="10"]
 Quadratwurzel: 121 : [i="1", j="11", k="11"]
 Quadratwurzel: 36 : [i="1", j="12", k="12"]
 Quadratwurzel: 144 : [i="1", j="12", k="12"]
 Quadratwurzel: 169 : [i="1", j="13", k="13"]
 Quadratwurzel: 196 : [i="1", j="14", k="14"]
 Quadratwurzel: 225 : [i="1", j="15", k="15"]
 Quadratwurzel: 4 : [i="2", j="1", k="2"]
 Quadratwurzel: 16 : [i="2", j="1", k="8"]
 Quadratwurzel: 36 : [i="2", j="1", k="18"]
 Quadratwurzel: 4 : [i="2", j="2", k="4"]
 Quadratwurzel: 16 : [i="2", j="2", k="16"]
 Quadratwurzel: 36 : [i="2", j="2", k="36"]
 Quadratwurzel: 64 : [i="2", j="2", k="64"]
 Quadratwurzel: 25 : [i="2", j="3", k="3"]

Abbildung 3 - Aufgabe 18

Wichtig zu erwähnen ist, dass das Ergebnis auf der rechten Seite um einiges Grösser ist als in der Abbildung 3 zu sehen.

Wenn das alle Quadratzahlen gesehen werden wollen, so kann das Python File Aufgabe18.py dazu verwendet werden.

Aufgabe 19

Wandeln Sie folgende for-Schleife in eine while-Schleife um, die die gleiche Ausgabe wie die for-Schleife erzeugt. (4 P.)

```
j = 0
for i in range(1, 100):
    j += i
    print(j, end = " ")
```

Abbildung 4 - Aufgabenstellung Aufgabe 19

Lösung:

Die Lösung ist eine Umwandlung der Zeile in welcher das for verwendet wird. Grundsätzlich sind folgende Dinge zu beachten:

1. Range beginnt nicht bei 0
2. Die Variable j wird um i erhöht, i wird bei jeder Iteration ebenfalls erhöht
3. Im Print werden die einzelnen j mit einem Abstand verkettet

Die Lösung mit der while-Schleife kann nun beispielsweise folgendermassen aussehen:

```

96
97 ##### Aufgabe 19
98 Wandeln Sie folgende for-Schleife in eine while-Schleife um,
99 die die gleiche Ausgabe wie die for-Schleife erzeugt. (4 P.)
100
101 ###
102
103 # for Schleife (Aufgabenstellung)
104 j = 0
105 for i in range(1, 100):
106     j += i
107     print(j, end = " ")
108
109 ###
110
111 # while Schleife (Mein Code)
112 print("\n\nWhile Schleife: ")
113 # Index ist zwei, weil bei range der Wert 1 nicht mitgezählt wird
114 index = 2
115
116 # j ist 1, weil Bei der Zahl 1 begonnen wird
117 j = 1
118 while index < 101:
119     print(j, end = " ")
120     j += index
121     index += 1
122
123
124 ##### md

```

Aufgabe 19

Wandeln Sie folgende for-Schleife in eine while-Schleife um, die die gleiche Ausgabe wie die for-Schleife erzeugt. (4 P.)

```

# for Schleife (Aufgabenstellung)
j = 0
for i in range(1, 100):
    j += i
    print(j, end = " ")

```

1 3 6 10 15 21 28 36 45 55 66 78 91 105

```

# while Schleife (Mein Code)
print("\n\nWhile Schleife: ")
# Index ist zwei, weil bei range der Wert 1 nicht mitgezählt wird
index = 2

# j ist 1, weil Bei der Zahl 1 begonnen wird
j = 1
while index < 101:
    print(j, end = " ")
    j += index
    # Als letztes den Index erhöhen
    index += 1

```

While Schleife:

1 3 6 10 15 21 28 36 45 55 66 78 91 105

Abbildung 5 - Aufgabe 19

Alternativ kann das ganze auch als Rekursion gelöst werden:

```

123
124 ###
125 print("Rekursion: ")
126
127 def rekursion(j, zaehler):
128     if zaehler < 100:
129         j += zaehler
130         print(j, end = " ")
131         rekursion(j, zaehler+1)
132
133 rekursion(0, 1)
134 ##### md
135

```

While Schleife:

1 3 6 10 15 21 28 36 45 55 66 78 91 105

```

print("Rekursion: ")

def rekursion(j, zaehler):
    if zaehler < 100:
        j += zaehler
        print(j, end = " ")
        rekursion(j, zaehler+1)

rekursion(0, 1)

```

Rekursion:

1 3 6 10 15 21 28 36 45 55 66 78 91 105

Abbildung 6 - Aufgabe 19 als Rekursion

Das ist nicht nach Aufgabenstellung, aber trotzdem schön.

Aufgabe 20

Wie oft muss man einen Würfel im Durchschnitt würfeln (**random**), bis zum ersten Mal 20 mal eine 6 gewürfelt wurde?

Schreiben Sie Python-Code, der ein solches kleines Häufigkeitsexperiment durchführt.

Wie viele Teilerperimente müssen Sie durchführen? Diskutieren Sie. (Beispiel, siehe Vorlesung). **(4 P.)**

Lösung:

Mein Code wählt zufällige Zahlen von 1 bis 6 und speichert den jeweiligen Wert in einer Variablen. Dieser Wert (in der Variablen) wird überprüft. Falls er eine 6 ist, dann wird die Variable `anzSechser` um 1 erhöht.

Zum Schluss wird bei jedem Durchgang die Variable `anzWurf` um 1 höher gezählt, damit festgehalten wird, wie oft gewürfelt werden musste bis 20 6er gewürfelt wurden:



```
131
132 #%%
133 # Die Bibliothek import nicht vergessen zu importieren
134 def wurfBisZwanzig():
135     # Wird für jeden gewürfelten 6er um 1 erhöht
136     anzSechser = 0
137
138     # Wird bei jedem würfeln um 1 erhöht (also alle Zahlen)
139     anzWurf = 0
140
141     while anzSechser < 20:
142         zufallsZahl = random.randint(1, 6)
143         if zufallsZahl == 6:
144             anzSechser += 1
145             print(str(zufallsZahl) + " <---- " + str(anzSechser))
146             anzWurf += 1
147             print(str(zufallsZahl))
148
149     print("Anzahl Sechser: " + str(anzSechser))
150     print("Anzahl Würfe: " + str(anzWurf))
151     return anzWurf
152
153 wurfBisZwanzig()
```

Output:

```
1
3
1
6 <---- 18
6
4
3
1
1
5
1
5
4
6 <---- 19
6
3
4
5
1
2
2
5
2
1
2
3
6 <---- 20
6
Anzahl Sechser: 20
Anzahl Würfe: 145
```

Abbildung 7 - Aufgabe 20

Die While Schleife ist zu Ende, wenn 20-mal ein 6er gewürfelt wurde.

In der Zip Datei ist nebst der Python Datei Aufgabe20.py noch die Python Datei Aufgabe20_Haufigkeit zu finden. In dieser Methode wird die oben gezeigt Funktion mehrmals durchprobiert und die Anzahl an benötigten Würfel-Versuchen addiert und durch die Anzahl an Durchläufen geteilt. So wird ein Mittelwert errechnet, welcher die Ungefähre Anzahl an Würfel-Versuchen anzeigt, welche benötigt werden, um 20-mal eine 6 zu würfeln.

Dieser Wert ist etwa 120. Der Grund dafür ist einfach mathematisch zu beweisen:

Allgemeine Chance einer 6: $\frac{1}{6}$

20-mal eine 6: $\frac{1}{6} / 20 \rightarrow \frac{1}{120}$

ACHTUNG: Wenn die Anzahl der Durchgänge zu hoch ist, kann das Programm durchaus 5 Minuten benötigen, um die Anzahl zu berechnen. (Weil jeweils noch Ausgaben getätigt werden)