Aufgabenblatt Repetition Python OF

(Lösungen)

Bearbeiten Sie die Aufgaben 1 bis 4a in der Shell

Aufgabe 1

Überlegen Sie zuerst im Kopf und berechnen Sie anschliessend in der Shell folgende Werte:

29*30, 16789-23.5, 23%6, 17/5, 17//5

870, 16765.5, 5, 3.4, 3

Aufgabe 2

a) Wie viel Rest erhalten Sie, wenn Sie 1234567 durch 3456 dividieren? Berechnen Sie das Ergebnis möglichst einfach und speichern Sie es in der Variable *a* ab.

b) Drucken Sie den Wert von *a* aus und zwar so, dass als Ergebnis «a=*[Wert von a]*» ausgegeben wird

Lösung:

```
>>> a=1234567 % 3456
>>> print("a=",a)
a= 775
```

Aufgabe 3

a) Speichern Sie ihren Namen in einer Variable *mein_name* ab.
b) Drucken Sie auf der Konsole folgendes aus «Guten Tag, *[mein_Name]*! Wie geht es Dir?»

Lösung:

```
>>> name = "Eveline"
>>> print("Guten Tag, " + name + "! Wie geht es Dir")
Guten Tag, Eveline! Wie geht es Dir
```

Aufgabe 4

a) Berechnen Sie die Nullstellen von $f(x) = 4x^2 - 34x + 66$ in der Shell. Nutzen Sie dazu ihnen bekannte Formeln aus der Mathematik

Lösung:

```
>>> x1=(-(-34)+((-34)**2-4*4*66)**0.5)/(2*4)
>>> x2=(-(-34)-((-34)**2-4*4*66)**0.5)/(2*4)
>>> print("x1=",x1,"x1=",x2)
x1= 5.5 x1= 3.0
```

b) Erstellen Sie ein Programm mit dem Namen allgemeineLoesungsformel.py. Im Programm werden in den ersten 3 Zeilen die Werte der Koeffizienten a, b und c einer quadratischen Funktion $f(x) = ax^2 + bx + c$ in Variablen abgelegt. In der vierten Zeile wird die erste und in der fünften Zeile die zweite Nullstelle der entsprechenden Funktion berechnet. In der sechsten Zeile steht ein print-Befehl, welcher die Ergebnisse in geeigneter Form ausgibt.

Lösung:

```
a = 4
b = -34
c = 66
x1 = (-b+(b**2-4*a*c)**0.5)/(2*a)
```

```
x2 = (-b-(b**2-4*a*c)**0.5)/(2*a)
print("x1=", x1,"x2=",x2)
```

c) Erweitern Sie ihr Programm so, dass die Koeffizienten durch den User über die Shell eingegeben werden können. Nutzen Sie dazu den Befehl input() und konvertieren Sie den eingelesenen Wert entweder in eine Fliesskommazahl (float) oder einen Integerwert (int).

Lösung:

```
print("Nullstellen für f(x)=ax^2+bx+c")
a = float(input("Wert für a:"))
b = float(input("Wert für b:"))
c = float(input("Wert für c:"))
x1 = (-b+(b**2-4*a*c)**0.5)/(2*a)
x2 = (-b-(b**2-4*a*c)**0.5)/(2*a)
print("x1=", x1,"x2=",x2)
```

Aufgabe 5

Schreiben Sie ein Programm, das von einer Anwenderin/ einem Anwender solange Strings einliest und in einer Liste speichert, bis der String `'exit'` eingegeben wird, der als letzter in der Liste abgelegt wird. Anschliessend sollen alle Werte der Liste einzeln in der Shell ausgegeben werden. Vergleichen Sie Ihre Lösungen untereinander.

Lösung:

Hier gibt es verschiedene Lösungen. Die Aufgabe hat zwei Teile.

Der **Einlesenteil:** Hier werden Sie sicher einen while-Loop brauchen, da es beliebig viele Eingaben geben kann, bis das erste Mal `'exit'` eingegeben wird. Dieses Verhalten kann weder mit aneinandergefügten if-Anweisungen noch mit einem for-Loop erreicht werden. Bei der Bedingung für den while-Loop haben Sie verschiedene Möglichkeiten: Varianten 1 und 2 nutzen eine boolesche Variable, die speichert, ob `'exit'` eingegeben wurde. Variante 3 nutzt das Schlüsselwort break, mit dem ein Loop jederzeit verlassen werden kann und Variante 4 arbeitet ebenfalls mit einer Hilfsvariable, speichert darin aber einen grundsätzlich fast beliebigen String, überschreibt diesen jeweils mit der Eingabe und prüft dann ob dieser Hilfsstring gleich `'exit'` ist.

Um die Eingaben zu speichern, müssen Sie noch vor dem Loop eine leere Liste erstellen, die Sie dann schrittweise befüllen.

Der **Ausgabeteil**: Es gibt im Wesentlichen zwei Varianten, wie Sie Listen mittels for-Loop durchlaufen (traversieren) können: in Varianten 1, 3 und 4 werden die Listenelemente im for-Loop direkt aufgerufen, während sie in der Variante 2 über ihren Index angesprochen werden. Je nach Situation ist die eine oder die andere Variante vorzuziehen.

Variante 1:

```
fertig = False
liste = []
while not fertig:
    wort = input("nächste Eingabe: ")
    liste.append(wort)
    if wort == 'exit':
        fertig = True
for wort in liste:
    print(wort)
```

Variante 2:

```
fertig = False
```

```
liste = []
while not fertig:
    wort = input("nächste Eingabe: ")
    liste.append(wort)
    if wort == 'exit':
        fertig = True
for i in range(len(liste)):
    print(liste[i])
```

Variante 3:

```
liste = []
while True:
    wort = input("nächste Eingabe: ")
    liste.append(wort)
    if wort == 'exit':
        break
for wort in liste:
    print(wort)
```

Variante 4:

```
wort = ''
liste = []
while wort != 'exit':
    wort = input("nächste Eingabe: ")
    liste.append(wort)
for wort in liste:
    print(wort)
```

Aufgabe 6

Schreiben Sie ein Programm *primzahlen.py*, das von der Anwenderin / dem Anwender zuerst eine ganze Zahl einliest und dann eine Liste mit allen Primzahlen kleiner gleich der gesuchten Zahl erzeugt und diese in der Shell wieder ausgibt.

Tipp: Nutzen Sie im Programm eine selbst definierte Funktion, die Ihnen zurückgibt, ob eine Zahl eine Primzahl ist oder nicht. Das macht Ihr Programm übersichtlicher und leichter lesbar.

Lösung: Verschiedene Herangehensweisen möglich!

Folgende Funktion prüft, ob n eine Primzahl ist. Primzahlen sind alle diejenigen natürlichen Zahlen, die genau zwei Teiler haben: 1 und sich selbst. Entsprechend ist 1 keine Primzahl (hat nur ein Teiler), 2 ist eine Primzahl, 3 auch, 4 nicht, da 4 drei Teiler hat: 1,2 und 4. Wir gehen also verschiedene Optionen durch. Für Zahlen kleiner 2 sagen wir in der dritten Zeile bereits: «Keine Promzahl». Bei allen anderen Zahlen testen wir, ob es zwischen 1 und der Zahl selbst, also bei den Zahlen von 2 bis n-1 eine Zahl gibt, die n ohne Rest teilt. Wenn das der Fall ist, wissen wir, dass n keine Primzahl ist und geben entsprechend False zurück und verlassen die Funktion. Falls die letzte Zeile mit «return True» erreicht wird, wissen wir, dass die Zahl nicht kleiner als 2 ist und keine weiteren Teiler als 1 und sich selbst hat. Daher wissen wir, dass sie eine Primzahl sein muss.

```
def primzahl(n):
    if n<2:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
```

```
zahl = int(input("Gib eine ganze Zahl ein: "))
liste = []
for i in range(2, zahl+1):
    if primzahl(i):
        liste.append(i)
print("Liste mit den Primzahlen <",zahl,":",liste)
```

Der Rest des Programms macht folgendes:

Zuerst wird eine Zahl von der Anwenderin/dem Anwender eingelesen und als ganze Zahl in der Variable zahl abgelegt.

Dann wird eine leere Liste mit dem Namen liste erstellt

Schliesslich werden in einem for-Loop alle Zahlen zwischen 2 und der Zahl selbst dahingehend überprüft, ob sie Primzahlen sind, indem die eben definierte Funktion genutzt wird. Wenn sie Primzahlen sind, werden sie zur eben erstellten Liste hinzugefügt.

In der letzten Zeile wird die erstellte Liste in der Shell ausgegeben.