Übungszettel 4a - Python

Aufgabe 1: Listen

Mache dich im Python-Interpreter mit dem Umgang mit Listen vertraut. Kapitel 2.6 im Skript (S.40) enthält nützliche Informationen.

(a) Lege eine Liste mit Namen "zahlen" an, die mindestens 10 Elemente enthält.

(b) Gib das 7. Element der Liste aus.

(c) Gib das 2.-8. Element der Liste als Liste aus.

(d) Gib das 2.-8. Element der Liste als einzelne Elemente aus.

(e) Füge die Zahlen 15, 23 und 95 zur Liste hinzu.

(f) Lösche das 3. Element der Liste

(g) Lösche die Zahl 23 aus der Liste

Solution: Die komplette Aufgabe soll im Interpreter bearbeitet werden. Im folgenden die einzelnen Befehle

```
##############################
# Interpretereingaben zur Bearbeitung der Aufgabe zu Listen
##############################

zahlen = [2,6,10,3,6,52,87,17,10,65] # Liste anlegen

zahlen[6] # 7.(!) Element ausgeben

zahlen[1:8] # Elemente 2-8 als Liste

for i in range(1,8): # als einzelne Elemente; nach print muss z mal Enter
    print(zahlen[i]) # gedrueckt werden, um das Ende der for-Schleife zu
        signalisieren

zahlen.append(15)  #Zahlen hinzufuegen
zahlen.append(23)
zahlen.append(95)

del zahlen[2] # 3.Element loeschen

zahlen.remove(23) # 23 aus Liste loeschen
```

Aufgabe 2: Funktionen

Schreibe eine Funktion, die eine Liste von Zahlen entgegennimmt und

(a) Das kleinste Element der Liste ausgibt

(b) Das größte Element der Liste ausgibt

(c) Alle Elemente der Liste ausgibt, die durch 5 teilbar sind.

(d) Alle Elemente der Liste ausgibt, die durch eine beliebige Zahl, die der Funktion übergeben wird, teilbar sind.

Verwende nicht die eingebauten Funktionen `min` oder `max`.

Solution:

(a)

```python
#!/usr/bin/env python
# encoding: utf-8
"""
j-simpleFunction.py

Created by Johannes Seitz on 2010-09-21.
"""

def minimum(list):
    min = float("infinity")
    for number in list:
        if number < min:
            min = number
    return min

def maximum(list):
    max = 0
    for number in list:
        if number > max:
            max = number
    return max

def byFive(list):
    return [x for x in list if x % 5 == 0]


def divisable(list, number):  # schick in ganz kurz
    return [x for x in list if x % number == 0]

# alternativ:

def divisable(list, number):
    teilbar = [] # leere Liste anlegen fuer Ergebnis
    for x in list:  # durch Liste laufen
        if(x % number == 0):  #falls Element durch number teilbar
            teilbar.append(x) # Element in Ergebnisliste einfuegen
    return teilbar

#  Programm zum testen der Funktionen
def main():
    list = [5,12,434,5464,232,12,6]
    assert min(list) == minimum(list)
    assert max(list) == maximum(list)
    assert [5] == byFive(list)
    assert [5] == divisable(list,5)

if __name__ == '__main__':
    main()
```

Aufgabe 3: Namensräume und Shadowing

Folgender Programmcode ist gegeben:

```python
x = "foo"
def x():
```

```
3     x = "bar"
4     print(x)
5 print(x)
```

(a) Was wird das Programm ausgeben?

a) Das Programm terminiert mit einem Fehler.
b) Das Programm gibt zuerst “bar”, dann “foo” aus.
c) Das Programm gibt etwas wie “<function x at 0x100495578 >” aus.
d) Das Programm gibt nur “foo” aus.

(b) Wie wäre die Ausgabe, wenn die Funktion einen anderen Namen als “x” hätte?

Solution:

(a) Die Leute sollen das am besten selbst ausprobieren.

Die 3. Option ist richtig. *Erklärung:* In Zeile 2 wird die Variable x, die zunächst auf den String “foo” zeigt, an eine Funktion gebunden. Aufgrund der dynamischen Typisierung führt diese Zuweisung zu keinem Fehler. Die Zuweisung von “bar” an Variable x in Zeile 3 findet im Namensraum der Funktion statt, diese wird jedoch nicht ausgeführt, weshalb “bar” nicht ausgegeben wird. In der letzten Anweisung gibt Python die String-Darstellung der Funktion aus, auf die die variable x nun zeigt.

(b) In dem Fall wäre die 4. Antwort richtig. *Erklärung:* Siehe oben.

Aufgabe 4: Dateien lesen und schreiben

Erstelle mit einem Editor eine .txt-Datei, wie folgt:

- Gib eine Zahl ein.
- Wechsel mit „Enter“ in die nächste Zeile und gib noch eine Zahl ein,

Speichere diese Datei als eingabe.txt ab.
Erstelle nun ein Programm, welches die beiden Zahlen aus der Datei ausliest und eine Grundrechenoperation auf diesen ausführt (z.B. die beiden Zahlen addiert). Das Ergebnis dieser Operation soll in einer Datei ausgabe.txt abgespeichert werden.

Solution:

```
#!/usr/bin/env python
# encoding: utf-8
"""
inputOutput.py

Created by Linda Luy on 2010-09-26.
"""

file = open("eingabe.txt", 'r')
x = file.readline()
y = file.readline()
file.close
z = int(x)+int(y)

file = open("ausgabe.txt", 'w')
file.write(str(z))
file.close
```

Aufgabe 5: Rekursion und Iteration

Schreibe eine Funktion, die eine natürliche Zahl n als Parameter bekommt und das Produkt aller natürlichen Zahlen von 1 bis einschließlich n ausgibt.

(a) Implementiere diese Funktion iterativ

(b) Implementiere diese Funktion rekursiv

Hinweis: Das Beispiel der Summen-Funktion im Skript (S.60) könnte hilfreich sein.

Solution:

```
########################################
# Berechnet Fakultaet fuer n
#     iterativ und rekursiv
########################################

def prod(n):
    ergebnis = 1
    for i in range(1,n+1):
        ergebnis = ergebnis * i
    return ergebnis


def prod_rek(n):
    if(n == 1):
        return 1
    else:
        return n * prod_rek(n-1)
```

Aufgabe 6: Karnickel

Die Fibonacci-Folge ist eine unendliche Folge natürlicher Zahlen, die folgendermaßen definert ist:

$$fib(n) := \begin{cases} 1, & \text{falls } n = 1 \text{ oder } n = 2 \\ fib(n-1) + fib(n-2), & \text{sonst.} \end{cases}$$

(a) Implementiere eine Funktion `fib(n)`, die die n-te Fibonacci-Zahl rekursiv berechnet.

(b) Implementiere die Funktion iterativ.

(c) Vergleiche wie lange die iterative und die rekursive Funktion benötigt um `fib(35), fib(40)` oder `fib(45)` zu berechnen.
Hinweis: mit der Tastenkombination Strg + c kann ein laufendes Programm im Python-Interpreter abgebrochen werden.

Solution:

(a) Steht im Skript S. 61, muss man nur abtippen

(b)

```
####################################
# fibonacci rekursiv
####################################

def fib_rek(n):
    if(n == 1 or n == 2):
        return 1
    else:
        return fib(n-1) + fib(n-2)




####################################
# fibonacci iterativ
```

```
16 ###################################
17
18 def fib(n):
19     f1 = 1
20     f2 = 1
21     for i in range(2,n):
22         tmp = f2
23         f2 = f1 + f2
24         f1 = tmp
25     return f2
```

(c) Die rekursive Funktion läuft sehr lang. Erkärung siehe Skript S. 62. Die Laufzeit ist exponentiell.

Viel Erfolg!