Vorsemesterkurs
Informatik
Übungsaufgaben

Übungszettel 4b - Python

Aufgabe 1: Geschachtelte Funktionen

Schreibe eine Funktion, die zwei Integer addiert und dann eine andere Funktion aufruft, in der das Ergebnis der ersten Funktion mit 2 multipliziert wird.

Solution:

```
1  def ad(a, b):
2      Ergebnis=mu(a+b)
3      return(Ergebnis)
4
5  def mu (c):
6      return(c*2)
7
8  print("Ergebnis ist: " + str(ad(2,5)))
```

Aufgabe 2: Selectionsort

Um eine Liste mit n Zahlen zu sortieren, arbeitet der Selectionsort-Algorithmus in $(n-1)$ Phasen. Zunächt wird die kleinste Zahl der gesamten Liste bestimmt und diese dann mit der Zahl, die an erster Position der Liste steht, vertauscht. Dann wird die kleinste Zahl der letzten $(n-1)$ Elemente der Liste bestimmt und diese mit der Zahl an Posititon 2 vertauscht, usw., bis die gesamte Liste durchgearbeitet ist.

(a) Schreibe eine Funktion, die den Index der kleinsten Zahl einer ihr übergebenen Liste findet, und diesen zurückgibt.

(b) Schreibe eine Funktion `selsort(<list>)`, die eine unsortierte Liste von Zahlen entgegennimmt und eine sortierte Liste zurückgibt.

(c) Schreibe ein Programm, das Zahlen aus einer Datei liest, diese mit der Funktion `selsort( )` sortiert und wieder in die Datei zurückschreibt.

Solution:

```
1  ########################################
2  # sortieren mit Selectionsort
3  ########################################
4
5  def getMin(liste): # index der kleinsten Zahl
6      index = 0
7      for i in range(1,len(liste)):
8          if(liste[i]<liste[index]):
9              index = i
10     return index
11
12 def selsort(liste):
```

```
    for i in range(0,len(liste)):
        mini = i + getMin(liste[i:len(liste)]) # index des Minimums
        tmp = liste[i]
        liste[i] = liste[mini]
        liste[mini] = tmp
    return(liste)
```

Aufgabe 3: Rekursion und Iteration

(a) Eine positive ganze Zahl a ist durch eine positive ganze Zahl b ganzzahlig ohne Rest teilbar, wenn sich b mehrmals von a subtrahieren lässt, sodass das Ergebnis schließlich 0 ergibt. Implementiere eine Funktion, die überprüft, ob a durch b teilbar ist. Verwende an mathematischen Operationen ausschließlich die Subtraktion!
Schreibe sowohl eine iterative als auch eine rekursive Variante.

Solution:

```
#!/usr/bin/env python
# encoding: utf-8
import unittest

#Zur Erinnerung:

# Divident (Die Zahl die geteilt wird)
# Divisor (Die zahl durch die geteilt wird)

def teilbarkeit_iterativ(divident , divisor):
    while divident > 0:
        divident -= divisor
    return (divident == 0)

def teilbarkeit_rekursiv(divident, divisor):
    if divident > 0:
        return teilbarkeit_rekursiv(divident - divisor, divisor)
    elif divident == 0:
        return True  # a ist ganzzahlig durch b teilbar
    elif divident < 0:
        return False # a ist nicht ganzzahlig durch b teilbar

#Alles ab hier ist Testcode, den du ignorieren kannst.
class TestCode(unittest.TestCase):

    def test_teilbarkeit_iterativ(self):
        self.assertTrue(teilbarkeit_iterativ(6, 2))
        self.assertTrue(teilbarkeit_iterativ(8, 2))

        self.assertFalse(teilbarkeit_iterativ(6, 5))
        self.assertFalse(teilbarkeit_iterativ(6, 9))


    def test_teilbarkeit_rekursiv(self):
        self.assertTrue(teilbarkeit_rekursiv(6, 2))
        self.assertTrue(teilbarkeit_rekursiv(8, 2))

        self.assertFalse(teilbarkeit_rekursiv(6, 5))
        self.assertFalse(teilbarkeit_rekursiv(6, 9))

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

(b) Ein Palindrom ist eine Zeichenkette, die von vorne und von hinten gleich, d.h. symmetrisch ist. 'otto', 'anna', 'abcba', '666' und 'X' sind beispielsweise Palindrome.
Implementiere eine Funktion, die als Parameter einen String erwartet, und überprüft, ob dieser String ein Palindrom darstellt. Der Rückgabewert der Funktion soll `True` bzw. `False` lauten.
Schreibe sowohl eine iterative als auch eine rekursive Variante.
Hinweis:

- Unterscheiden sich bereits das erste und das letzte Zeichen, kann kein Palindrom vorliegen.
- Gleichen sich das erste und das letzte Zeichen...

Solution:

```python
def palindrom_rek(s):
    if len(s) <= 1: return True
    if s[0] != s[-1]: return False
    return palindrom_rek(s[1:-1])

def palindrom_it(s):
    l = 0         # left
    r = len(s)-1  # right
    while l < r:
        if s[l] != s[r]: return False
        l += 1
        r -= 1
    return True
```

Aufgabe 4: Collatz-Funktion

Die Collatz-Funktion ist mathematisch wie folgt definiert:

$$c(n) := \begin{cases} c(n/2), & \text{falls n gerade ist} \\ c(3n+1), & \text{falls n ungerade ist} \end{cases}$$

(a) Implementiere die Collatz-Funktion als rekursive Python-Funktion. Die Funktion soll einen beliebigen Startwert als Parameter erhalten, bei jedem Aufruf zunächst die aktuelle Zahl ausgeben, und terminieren, sobald das erste Mal die Zahl 1 in der Folge auftaucht.

Solution:

```python
def collatz(n):
    print(n)
    # Basisfall
    if n == 1:
        return
    # Rekursionsfall
    elif n%2==0:
        collatz(n//2)
    else:
        collatz(3*n+1)
```

(b) Erweitere die Funktion so, dass die Anzahl der Schritte bis zum ersten Auftauchen der Zahl 1 ausgegeben wird.

Solution:

```python
def collatz2(n,step=1):
    # print("step:",step, "n:",n)
    # Basisfall
    if n == 1:
```

```
        return step
    # Rekursionsfall
    elif n%2==0:
        return collatz2(n//2, step+1)
    else:
        return collatz2(3*n+1, step+1)
```

(c) Führe die Funktion für die Startwerte $1, 2, 3, \ldots, 1000$ aus und speichere jeweils die Anzahl der Schritte in einer Liste.

Solution:

```
schrittliste = []
for i in range(1,1001):
    schrittliste = schrittliste + [collatz2(i)]
```

(d) Werte die in der Liste gespeicherten Schrittzahlen aus. Was ist die durchschnittliche Schrittzahl? Bei welchen Startwerten wurde die maximale Schrittzahl erreicht, bei welchen die minimale Schrittzahl? Woran liegt das?

Solution:

```
mini = min(schrittliste)
maxi = max(schrittliste)
for i in range(0,1000):
    if schrittliste[i] == mini:
        print("minimale Schrittzahl bei", i+1)
    if schrittliste[i] == maxi:
        print("maximale Schrittzahl bei", i+1)
```

Es gibt nur eine Möglichkeit, dass die Rekursion nach einem Schritt beendet ist, nämlich für $n = 1$.
Gemäß Konstruktion der Collatz-Folge ist 871 ziemlich weit weg von der Zweierpotenz (16).

(e) Implementiere die Collatz-Funktion nun iterativ statt rekursiv.

Solution:

```
def collatzit(n,step=1):
    while(n>1):
        step = step+1
        if n%2==0:
            n = n//2
        else:
            n = 3*n+1
    return(step)
```

(f) **(Achtung, schwierig!)** Finde einen Startwert $n \in \mathbb{N}$, sodass die Collatz-Folge niemals die Zahl 1 erreicht. Was bedeutet das für den Startwert? Merke Dir die Zahl und setze Dich umgehend mit Deinem Tutor in Verbindung.

Solution: Wer diese Aufgabe löst, hat die Collatz-Vermutung widerlegt, kann den Vorkurs überspringen und sich auf viel Anerkennung freuen.
Stand 2009: Die Vermutung stimmt für alle Zahlen bis $20 \cdot 2^{58} = 5764607523034234880$, siehe: `http://www.ieeta.pt/~tos/3x+1.html`.
Es könnten nur die folgenden drei Fälle auftreten:

- Die Folge endet in einem 4-2-1-Zyklus.
- Die Folge wächst unbegrenzt.
- Die Folge gerät in einen anderen Zyklus.

Aufgabe 5: Programmieraufgaben

In dieser Aufgabe ist eine einfache Relation in einem Python-Programm umzusetzen, die die Liebesbeziehungen zwischen verschiedenen Personen festhält. Eine Person x liebt eine Person y, wenn (x, y) in der Relation der Liebesbeziehungen enthalten ist.

Die Relation können in Python beispielsweise in Form einer Liste von Tupeln festgehalten werden (Siehe auch [1]).

(a) Schlage in einem Lexikon Deiner Wahl folgende Begriffe nach: *Symmetrie (bei Relationen), Reflexivität, Transitivität.* Überlege Dir, ob die hier vorgestellte Relation der Liebesbeziehungen symmetrisch, reflexiv und/oder transitiv ist.

(b) Schreibe eine Funktion *verliebt_sich*, die die Namen des/der Verliebten und der/des Geliebten als Argumente übergeben bekommt und diese in die Liebesrelation einfügt.

(c) Schreibe eine Funktion *wen_liebt*, die einen Namen des/der Verliebten als Argument übergeben bekommt, und eine Liste der Personen, die diese Person liebt, zurückgibt.

(d) Schreibe eine Funktion *liebespaare*, die eine Liste aller Liebespaare zurückliefert. Ein Liebespaar liegt vor, wenn sowohl (x,y) als auch (y,x) in der Liebesrelation vorhanden sind.

Solution:

```python
#!/usr/bin/env python
# encoding: utf-8
"""
relationen0.py

Created by Johannes Seitz on 2010-08-15.
"""

LIEBES_RELATION = [("Hans", "Marianne"),
                   ("Marianne", "Hans"),
                   ("Erna","Hans"),
                   ("Hans", "Helmut"),
                   ("Frieda","Toni"),
                   ("Toni", "Alex"),
                   ("Marianne", "Erna")]

def verliebt_sich(verliebter, geliebter):
    LIEBES_RELATION.append((verliebter, geliebter))

def liebespaare():
    return [(x,y) for (x,y) in LIEBES_RELATION
            if (y,x) in LIEBES_RELATION]

def wen_liebt(verliebter):
    return [y for (x,y) in LIEBES_RELATION
            if x == verliebter]

def main():
    verliebt_sich("Horst", "Erna")
    print("Die Relationen:")
    print(LIEBES_RELATION)
```

[1] http://docs.python.org/library/datatypes.html

```
32      print("Die Liebespaare:")
33      print(liebespaare())
34      print("Wen liebt Hans?")
35      print(wen_liebt("Hans"))
36      print("Wen liebt Stefan?")
37      print(wen_liebt("Stefan"))
38
39  if __name__ == '__main__':
40      main()
```

Viel Erfolg!