Programmierworkshop

Python

Teil 1

Übungsaufgaben

und Musterlösungen

Übung zur Verwendung von if / elif / else

Übung: Folgendes Programmfragment sei vorgegeben:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

n1, n2, n3 = 6, 5, 7 # Or any other numbers

print "Unsorted: ", n1, n2, n3

#
# Insert sorting here
#

print "Sorted: ", n1, n2, n3
```

Aufgabe: Sortiere n1, n2, n3 mit Verwendung von if / elif / else.

Dabei auch rumspielen mit:

a) Sequence assignments

b) Ternären Operatoren

Musterlösung:

```
if (n1 < n2 and n1 < n3):
    n1, n2, n3 = ((n1, n2, n3)
                  if n2 < n3 else (n1, n3, n2))
elif (n2 < n1 and n2 < n3):
    n1, n2, n3 = ((n2, n1, n3)
                  if n1 < n3 else (n2, n3, n1))
else:
    n1, n2, n3 = ((n3, n1, n2)
                  if n1 < n2 else (n3, n2, n1))
```

Übungen zu while

1) Schreibe ein Programm, das einen Integerwert in Binärdarstellung anzeigt.

Beispiel:

```
167 as binary:  1 0 1 0 0 1 1 1
```

2) Schreibe ein Programm, das für einen Einkauf das Rückgeld in den jeweiligen Münzarten ausgibt.
(Ganzzahlige Geldwerte mit nur 1€ / 2€ / 5€ Rückgeld ist auch ok)

Beispiel:

```
Bill:  5.89
Given:  10.00
Change:  4.11
Your change is:
    2 coins a 2 Euros
    1 coins a 10 Cents
    1 coins a 1 Cent
```

Musterlösungen:

```
n = 167
factor = 1
while (factor <= n//2):
    factor *= 2

print n, " in Binary: ",
while (factor >= 1):
    if n >= factor:
        print '1',
        n -= factor
    else:
        print '0',
    factor //= 2
print
```

```
change = given - bill
print "Given: ", given
print "Bill: ", bill
print "Change: ", change
change = int(change * 100)

if (change == 0):
    print "Exact amount! Thank you!"
elif (change < 0):
    print "That is not enough!"
else:
    change_2euros = 0
    change_1euros = 0
    change_50cents = 0
    change_20cents = 0
    change_10cents = 0
    change_5cents = 0
    change_2cents = 0
    change_1cents = 0
```

```
while (change >= 200):
    change_2euros += 1;  change -= 200
while (change >= 100):
    change_1euros += 1;  change -= 100
while (change >= 50):
    change_50cents += 1; change -= 50
while (change >= 20):
    change_20cents += 1; change -= 20
while (change >= 10):
    change_10cents += 1; change -= 10
while (change >= 5):
    change_5cents += 1;  change -= 5
while (change >= 2):
    change_2cents += 1;  change -= 2
while (change >= 1):
    change_1cents += 1;  change -= 1

print "Your change is: "
if (change_2euros > 0):
    print "   ", change_2euros, "coins a 2 Euros"
if (change_1euros > 0):
    print "   ", change_1euros, "coins a 1 Euro"
if (change_50cents > 0):
    print "   ", change_50cents, "coins a 50 Cents"
if (change_20cents > 0):
    print "   ", change_20cents, "coins a 20 Cents"
if (change_10cents > 0):
    print "   ", change_10cents, "coins a 10 Cents"
if (change_5cents > 0):
    print "   ", change_5cents, "coins a 5 Cents"
if (change_2cents > 0):
    print "   ", change_2cents, "coins a 2 Cents"
if (change_1cents > 0):
    print "   ", change_1cents, "coins a 1 Cent"
print
```

Übungen zu for

1) Schreibe ein Programm, das eine Umrechnungstabelle Grad Celsius nach Fahrenheit sowie Kelvin ausgibt. Die Tabelle soll dabei z.B. von 0°C bis 100°C in 10-er Schritten berechnet werden.

°F = °C * 1,8 + 32,0
°K = °C + 273,15

2) Berechne das Reiskorn-Schachbrett Problem für N x N Schachbretter. Das erste Feld wird mit 1 Reiskorn belegt, das zweite mit 2, das dritte mit 4; jedes folgende Feld wird mit dem doppelten des Vorgängerfeldes belegt. Wieviele Reiskörner sind es insgesamt, in Abhängigkeit von N?

Musterlösungen:

```
print "   °C      °F      °K"
for celsius in range(0,101,10):
    fahrenheit = celsius * 1.8 + 32.0
    kelvin = celsius + 273.15
    print "%6.2f %6.2f %6.2f" % (celsius, fahrenheit,
                                 kelvin)
print
```

```
rice = 0
counter = 1
for i in range(n*n):
    rice += counter
    print "Field No. ", i+1,
    print " corns added ", counter,
    print " corns total ", rice
    counter *= 2
print n*n, " fields ", rice, "corns"
print
```

Hinweis: Die folgenden Übungen zu Strings und Listen wurden der Google Python Class entnommen. Laut den Angaben auf den Seiten der Google Python Class wurden die dortigen Texte unter der Creative Commons License 2.5 veröffentlicht. Die Übungsmaterialien der Google Python Class wurden unter der Apache License 2.0 veröffentlicht.

http://code.google.com/intl/de/edu/languages/google-python-class/index.html

Die Creative Commons Licence 2.5 kann hier eingesehen werden:

http://creativecommons.org/licenses/by/2.5/

Die Apache Licence 2.0 kann hier eingesehen werden:

http://www.apache.org/licenses/LICENSE-2.0

Die Veröffentlichung dieser Sammlung von Übungsaufgaben sowie Musterlösungen erfolgt unter Berufung auf die Bestimmungen der jeweiligen Lizenzen, welche Weitergabe sowie Modifizierung erlauben falls der ursprüngliche Autor genannt wird und die ursprüngliche Copyrightnotiz mitgeliefert wird, was hiermit getan wird:

Code for Google's Python Class
Copyright 2010 Google Inc.

This code developed by Nick Parlante at Google Inc.

Übungen zu Strings

a) Schreibe eine Funktion *both_ends(s)*, die von einem String nur dessen erste sowie letzte zwei Zeichen zurückgibt. Gib einen leeren String zurück, falls s kürzer als 4 Zeichen ist.

```
"Herbst" -> "Hest"
```

b) Schreibe eine Funktion *fix_start(s)*, die in einem String alle Vorkommen des ersten Zeichens durch einen Stern * ersetzt, aber das erste Zeichen unverändert lässt

```
"google" -> "goo*le"
```

c) Schreibe eine Funktion *mix_up(a, b)*, die einen einzelnen String zurückgibt, der aus beiden Strings a und b besteht, aber die ersten zwei Zeichen jeweils vertauscht sind

```
"Sommer", "Herbst" -> "Hemmer Sorbst"
```

d) Schreibe eine Funktion *not_bad(s)*, die in einem String nach dem Vorkommen von 'not' und irgendwann danach 'bad' sucht. Falls vorhanden, ersetze den kompletten Substring 'not ... bad' mit 'good'

```
"The dinner is not that bad!" -> "The dinner is good!"
```

Schaue bei diesen Übungen auch mit `help(str)`, ob es Stringmethoden gibt, die die Aufgaben vereinfachen können.

Musterlösungen:

```
def both_ends(s):
    if len(s) < 4:
        return ""
    else:
        return s[0:2] + s[-2:]

def fix_start(s):
    front = s[0]
    back = s[1:]
    fixed = back.replace(front, '*')
    return front + fixed

def mix_up(s1, s2):
    return s2[:2] + s1[2:] + " " + s1[:2] + s2[2:]

def not_bad(s):
    n = s.find("not")
    b = s.find("bad")
    if n != -1 and s != -1 and b > n:
        s = s[:n] + "good" + s[b+3:]
    return s
```

Übungen zu Listen

a) Schreibe eine Funktion *match_ends(words)*, die aus einer Liste die Zahl der Strings zurückgibt, die eine Länge > 2 haben und das erste und letzte Zeichen das gleiche ist.

b) Schreibe eine Funktion *front_x(words)*, die eine sortierte Liste zurückgibt. Allerdings sollen alle Strings, die mit "x" anfangen, sortiert am Anfang dieser Liste stehen.

```
["mix", "xyz", "apple", "xanadu"] -> ["xanadu", "xyz",
"apple", "mix"]
```

c) Schreibe eine Funktion *remove_multiples(e)*, die eine Liste zurückgibt, in der sämtliche Einträge der Originalliste nur einmal vorkommen.

```
[1, 2, 3, 3, 2] -> [1, 2, 3]
```

d) Schreibe eine Funktion *sort_last(tuples)*, die eine Liste von Tupeln sortiert zurückgibt. Es soll aber nach dem jeweils letzten Element der Tupel sortiert werden. Die Tupel können unterschiedliche Länge haben.

```
[(1, 7), (1, 3), (3, 4, 5), (2, 2)] -> [(2, 2), (1, 3),
(3, 4, 5), (1, 7)]
```

(Hint: In der Funktion *sorted()* bzw. der Listenmethode *list.sort()* gibt es den optionalen Parameter *key=* , mit dem man eine Funktion übergeben kann)

Musterlösungen:

```
def match_end(words):
    count = 0
    for s in words:
        if len(s) > 2 and s[0] == s[-1]:
            count += 1
    return count

def front_x(words):
    ret1 = []
    ret2 = []
    for i in words:
        if i[0] == 'x':
            ret1.append(i)
        else:
            ret2.append(i)
    return sorted(ret1) + sorted(ret2)

def remove_multiples(s):
    ret = []
    for i in s:
        if i not in ret:
            ret.append(i)
    return ret

def get_last(a):
    return a[-1]
def sort_last(s):
    return sorted(s,key=get_last)
```

Hinweis:

Die Funktion sort_last kann auch ohne eine Hilfsfunktion mit Hilfe eines Lambda-Ausdruckes geschrieben werden:

```
def sort_last(s):
    return sorted(s,key=lambda a:a[-1])
```

Übung zu Listen / Dictionaries

Schreibe ein Programm, was von der Eingabe solange einliest, bis der Benutzer mit einer Leerzeile die Eingabe abschließt. Danach sollen die eingebenen Worte als Tabelle ausgegeben werden, zusammen mit der Angabe wie oft ein Wort vorgekommen ist.

Beispiel:

Eingabe:

```
abc def
ghi def abc jkl abc
```

Ausgabe:

```
3 abc
2 def
1 ghi
1 jkl
```

Hints:

Für die Eingabe benutze die Funktion *raw_input()*, die solange Input von stdin einliest bis eine Leerzeile eingegeben wird.

Zum Teilen von Strings gibt es die Methode *a = s.split()*, was eine Liste der durch Whitespace getrennten Wörter des Strings liefert.

Musterlösung:

```
def get_frequencies():
    l = []
    a = raw_input()
    while (a):
        l.extend(a.split())
        a = raw_input()

    d = {}
    for s in l:
        if s not in d:
            d[s] = l.count(s)

    for (k,v) in d.items():
        print v, k
```