Übungsblatt 3

Willkommen zum Praktikum zu Programmieren in Python.

Aufgabe 1. Schreiben Sie folgende `for`-Schleife als `while`-Schleife um.

```
lis = [1, 2, 3]
for ele in lis:
  print(ele)
```

Schreiben Sie folgende `while`-Schleife als `for`-Schleife um.

```
dic = {1: "eins", 2: "zwei", 3: "drei"}
lis = list(dic.keys())
idx = 0
while idx < len(lis):
  key = lis[idx]
  print(key, dic[key])
  idx = idx + 1
```

Schreiben Sie die Programme so kompakt wie möglich.

Aufgabe 2. Schreiben Sie eine rekursive Funktionen `ggTr`, die den größten gemeinsamen Teiler von zwei Zahlen berechnet. Erinnern Sie sich daran, dass

$$\forall x, y : \texttt{ggT}(x, y) = \texttt{ggT}(y, x) \land \forall x > y : \texttt{ggT}(x, y) = \texttt{ggT}(x - y, y)$$

Testen Sie Ihre die Funktionen mit einigen Beispielen wie `(10,30)`, `(20,30)`, `(2,5)`, `(8,6)`, `(7,3)`. Die naive rekursive Version ist leider nicht sehr effizient. Erinnern Sie sich, dass auch gilt

$$\forall x > y, x\%y \neq 0 : \texttt{ggT}(x, y) = \texttt{ggT}(x\%y, y) \tag{1}$$

Schreiben Sie eine iterative Funktion `ggT`, die den größten gemeinsamen Teiler von zwei Zahlen mit Hilfe von (1) berechnet. Lesen Sie die Zahlen aus der Datei `ggts.dat` ein. Jede Zeile repräsentiert eine Zahl. Berechnen Sie den `ggT` aller Zahlen der Zeilen i und $i+1$ für gerade $i \geq 0$. Was ist der Mittelwert aller `ggT`s?

Schreiben Sie darauf aufbauend eine Funktion `ggTl`, die den größten gemeinsamen Teiler einer Liste von Zahlen berechnet. Beachten Sie, dass der `ggT` dreier Zahlen `x,y` und `z` gleich dem `ggT` von dem `ggT` zweier Zahlen (zum Beispiel `ggT(x,y)`) und der dritten Zahl (zum Beispiel `ggT(ggT(x,y),z)`) ist. Was ist der `ggT` von 10, 80, 20 und 75?

Ab funktionales Programmieren: Definieren Sie die Funktion `ggTl` unter Verwendung von `ggT` noch einmal, aber verwenden Sie weder Rekursion noch `while` noch `for`.

Aufgabe 3. Abgleich von Argumenten. Definieren Sie die folgenden Funktionen und werten Sie die folgenden Ausdrücke (immer nur einen) aus. Erklären Sie das jeweilige Ergebnis anhand des Argumentenabgleichs wie in der Vorlesung vorgestellt.

```
def f(a, b, c=1):
  print(a, b, c)
def g(a, b, *c, **d):
  print(a, b, c, d)
def h(a, b, c=1, *d, **e):
  print(a, b, c, d, e)

f(1, 2, 3),  f(1), f(1, 2), f(1, 2, 3, 4)
g(1, 2),  g(1, 2, 3, 4),  g(1, 2, 3, 4, bla="bla")
h(1, 2, 3, 4, 5, 6, c=7),  h(1, 2, 3, 4, 5, 6, x=7)
```

Aufgabe 4. In Programm-Quellcode von C/C++ oder anderen blockbasierten Sprachen haben Sie häufig geschweifte beziehungsweise runde Klammern, die Blöcke von zusammenhängenden Text umschließen. So ist zum Beispiel

```
void fun(int x, int y) {
  cout << " korrekt ";
}
```

ein korrektes C++-Programm.

```
void fun(int x, int y) { {
  cout << " offene Klammer zu viel ";
}
```

ein fehlerhaftes C++ Programm.

Schreiben Sie ein Python-Programm `checkblock.py`, das überprüft, ob per Kommandozeile übergebene blockbasierte Programme korrekt geklammert ist. Dazu muss für jede öffnende Klammer eines Typs eine passende schließende Klammer kommen, wir berücksichtigen `{}` und `()`. Zählen der Klammern reicht nicht, Sie brauchen einen Stack. Bei einem Klammerfehler geben Sie Zeile und Spalte der fehlerhaften Stelle aus. Zum Testen sind ein paar Programme als `samples/samplex.cpp` gegeben.

Beachten Sie, dass es uns nur um die Klammern geht, die restliche Syntax sollen (und können) wir nicht prüfen. Wir gehen auch davon aus, dass in Kommentaren und Strings keine Klammern vorkommen.

Testen Sie Ihr Programm mit den Unit-Tests in `test_checkblock.py`. Dazu muss eine Funktion `checkblock(content)` vorhanden sein, die den ersten Fehler in dem mehrzeiligen String `content` als Tupel `(zeile, spalte, msg)` zurück gibt, wobei `msg` eine passende Nachricht ist. Falls am Ende eine schließende Klammer fehlt, wird diese am Anfang der nächsten Zeile erwartet. Falls kein Fehler vorliegt wird `None` zurückgegeben.

Aufgabe 5. Unter Linux können Sie sich mit dem Befehl `fortune` ein zufällig gewähltes Zitat ausgeben lassen Schreiben Sie ein Skript, das `fortune` nachbildet. Ohne Kommandozeilenparameter soll ein beliebiges Zitat ausgegeben werden. Mit der Option `-m <pat>` soll ein beliebiges Zitat ausgegeben werden, das `<pat>` enthält.

Die Zitate liegen unter `/usr/share/games/fortunes`, für die Windows-Jünger liegt ein Ordner in `Y:\Python\fortunes`. Interessant sind die Dateien, die weder die Endung `.dat` noch `.u8` haben. Die Zitate sind durch `\n%\n` voneinander separiert. Die Bibliotheken `os`, `random`, `sys`, `string` sind hilfreich.

moodle Python_Barth