Übungen zu Computergrundlagen WS 2018/2019

Übungsblatt 12: Python 3 & C

25. Januar 2019

Allgemeine Hinweise

- Abgabetermin für die Lösungen ist **Freitag, 01.02.2019, 11:00 Uhr**
- Schickt die Lösungen bitte per Email an Euren Tutor:
  - Montag 14:00–15:30: Grant Cates (`gcates@icp.uni-stuttgart.de`)
  - Dienstag 9:45–11:15: Kai Szuttor (`kai@icp.uni-stuttgart.de`)
  - Dienstag 15:45–17:15: Julian Michalowsky (`jmichalowsky@icp.uni-stuttgart.de`)
  - Mittwoch 15:45–17:15: Patrick Kreissl (`pkreissl@icp.uni-stuttgart.de`)
  - Donnerstag 9:45–11:15: Frank Maier (`fmaier@icp.uni-stuttgart.de`)
- Die Übungen sollen in Gruppen von jeweils *zwei bis drei* Leuten bearbeitet werden. Abgaben von Einzelpersonen werden nicht akzeptiert. Bitte gebt *nur eine Lösung pro Gruppe* ab und nennt in eurer Abgabe alle Mitglieder eurer Gruppe!

Aufgabe 12.1: Python: Funktionen, Rekursionen und Schleifen (7 Punkte)

Auf Blatt 9 habt Ihr bereits (in Pseudocode) eine rekursive und eine nicht-rekursive Funktion (unter Benutzung einer Schleife) für die Berechnung der Fakultät geschrieben. Um die Performance-Unterschiede der verschiedenen Implementierungsarten zu verdeutlichen, sollen in dieser Aufgabe zunächst verschiedene Python-Funktionen zur Berechnung der Fibonacci-Zahlen implementiert werden, um anschließend deren Laufzeiten zu vergleichen.

- **12.1.1** Die Sequenz der Fibonacci-Zahlen ist wie folgt definiert:

$$\mathrm{fib}(n) = \begin{cases} 0 & \text{falls } n = 0 \\ 1 & \text{falls } n = 1 \\ \mathrm{fib}(n-1) + \mathrm{fib}(n-2) & \text{sonst} \end{cases} \tag{1}$$

  Schreibe eine Python-Funktion `fib1(n)`, die die n-te Fibonacci-Zahl wie in Gleichung 1 berechnet. (1 Punkt)

- **12.1.2** Schreibe eine Funktion `fib2`, die die Fibonacci-Zahlen mit Hilfe einer Schleife berechnet, also ohne den rekursiven Aufruf der Funktion. (2 Punkte)

  Hinweis: Berechne zunächst auf dem Papier von Hand die ersten paar Fibonacci-Zahlen. Das macht es einfacher, zu verstehen, wie man die Schleife implementieren kann.

- **12.1.3** Eine andere, aber äquivalente, Definition der Fibonacci-Zahlen sieht so aus:

$$\text{fib}_{a,b}(n) = \begin{cases} a & \text{falls } n = 0 \\ b & \text{falls } n = 1 \\ \text{fib}_{b,(a+b)}(n-1) & \text{sonst} \end{cases} \quad (2)$$

$$\text{fib}(n) = \text{fib}_{0,1}(n) \quad (3)$$

Schreibe eine Python-Funktion `fib3`, die die Fibonacci-Zahlen wie in Gleichung 3 berechnet. (2 Punkte)

Hinweis: Am besten definierst Du die Indizes a und b in Python als optionale Parameter:

```
def fib3(n, a=0, b=1):
  # Hier kommt der Code!
```

- **12.1.4** Berechne mit Hilfe der Funktionen `fib1`, `fib2` und `fib3` aus den vorigen Aufgaben die Fibonacci-Zahlen für $n \in (0, 1, 2, \ldots 34, 35)$. Miss dabei jeweils die Laufzeiten der einzelnen Funktionsaufrufe und plotte mit Matplotlib diese Zeiten über n. Was fällt Dir bei den Laufzeiten auf? Wieso sind sie so unterschiedlich? (2 Punkte)

Hinweis: Die Laufzeit eines Funktionsaufrufs lässt sich mit dem Python-Modul `timeit` messen:

```
import timeit

def f(x):
    # Irgendein Code, dessen Performance wir gerne messen wollen

for n in range(10):
    timing = timeit.timeit("f(n)",
                           setup="from __main__ import f, n",
                           number=100) / 100.
    print(timing)
```

Beachte, dass aufgrund der Funktionsweise des `timeit`-Moduls der Funktionsaufruf `f(n)` als String übergeben wird und zudem sowohl die Funktion `f` als auch die zu übergebende Variable `n` aus dem Hauptteil des Skripts (`__main__`) importiert werden müssen. Diese Anweisung wird mittels des Arguments `setup` ebenfalls als String übergeben.
Je nach übergebenem Argument `n` kann die Laufzeit eines einzelnen Funktionsaufrufs `f(n)` mitunter sehr kurz sein. Um hierbei die Statistik zu verbessern kann die `timeit`-Funktion mit dem Argument `number=<xy>` angewiesen werden, den gewünschten Aufruf mehrfach auszuführen. Der übergebenen Funktionsaufruf wird nun `number` Mal ausgeführt und die dafür benötigte Zeit in Sekunden zurückgegeben. Um die Laufzeit pro Funktionsaufruf zu erhalten muss daher noch durch `number` (im Beispiel 100) geteilt werden. **Miss in deinem Skript die Laufzeiten am Besten zunächst mit `number=1` und passe die Variable nach Bedarf an.**

Aufgabe 12.2: C (3 Punkte)

- **12.2.1** Schreibe ein C-Programm `fib1`, das eine Ganzzahl n von der Standardeingabe einliest, die n-te Fibonacci-Zahl entsprechend Gleichung 1 berechnet und diese auf der Standardausgabe ausgibt. (2 Punkte)
- **12.2.2** Schreibe das C-Programm `fib1` zu `fib2` um, so dass es eine Ganzzahl n von der Standardeingabe einließt, die n-te Fibonacci-Zahl entsprechend Aufgabe 12.1.2 unter Verwendung einer Schleife berechnet und das Ergebnis auf der Standardausgabe ausgibt. (1 Punkt)