Informatik I: Einführung in die Programmierung

Prof. Dr. Peter Thiemann
Dr. Daniel Büscher, Hannes Saffrich
Wintersemester 2019

Universität Freiburg
Institut für Informatik

Übungsblatt 6

Abgabe: Montag, 2.12.2019, 9:00 Uhr morgens

Aufgabe 6.1 (Polynomauswertung; Datei: `poly_eval.py`; Punkte: 1+3+3+3)

In dieser Aufgabe sollen mehrere Funktionen zur Polynomauswertung implementiert werden, die sich nicht im Ergebnis, aber in der Laufzeit unterscheiden. Sie können die einzelnen Funktionen anhand der Beispiele der Vorlesung testen. Die Laufzeiten werden in der letzten Teilaufgabe ausgewertet.

(a) Implementieren Sie eine Funktion `poly_eval(p: list, x: float) -> float`, welche ein Polynom, dargestellt durch die Koeffizienten `p = [a0, a1, ..., an]`, an der Stelle `x` auswertet. Folgen Sie hierzu dem Beispiel der Vorlesung.

(b) Die Implementierung aus der Vorlesung ist nicht sehr effizient, da in jedem Schleifendurchlauf `x**i` ausgewertet werden muss. Außerdem ist `enumerate(p)` genutzt, welches etwas langsamer als eine einfache Iteration über `p` ist.

Implementieren Sie die alternative Funktion `poly_eval_faster(p: list, x: float) -> float`, welche mit zwei Produkten (statt der Potenzierung) pro Schleifendurchlauf auskommt. Nutzen Sie außerdem eine einfache Iteration über `p`.

(c) Implementieren Sie eine noch effizientere Funktion `poly_eval_horner(p: list, x:float) -> float`, welche das Horner-Schema[1] ausnutzt und mit einem Produkt pro Schleifendurchlauf und einfacher Iteration auskommt. Hinweis: Nutzen Sie Slicing von `p`, um eine geeignete Iteration zu erreichen.

(d) Implementieren Sie eine Funktion `time_poly_eval(name: str)`, welche die Laufzeit der Funktion mit Namen `name` auswertet und zusammen mit der Zeiteinheit und dem Funktionsnamen auf dem Bildschirm ausgibt. Beispiel:

```
>>> time_poly_eval('poly_eval')
Laufzeit von poly_eval : 1.3883828920079395 Sekunden
```

Es sollen hierbei 1000 Funktionsaufrufe mit `p = [0, 1, ..., 1999]` und `x = 2` getätigt werden. Nutzen Sie dafür das Modul `timeit`, siehe Dokumentation[2]. Beispiel:

[1] `https://de.wikipedia.org/wiki/Horner-Schema`
[2] `https://docs.python.org/3/library/timeit.html`

```
>>> timeit.timeit(
...     'poly_eval([1,2,3], 2)',
...     setup = 'from __main__ import poly_eval',
...     number = 1000)
0.6087981070159003
```

Verifizieren Sie die erwarteten Verhältnisse der Laufzeiten Ihrer Funktionen zur Polynomauswertung.

Hinweis. Die Funktion `timeit` nimmt als erstes Argument *einen String* der Pythoncode enthält. Sie können also `+` verwenden um `name` mit einem String für den Rest des Funktionsaufrufes zu verbinden.

Aufgabe 6.2 (Fibonacci-Folge; Datei: `fibonacci.py`; Punkte: 3)

Implementieren Sie eine Funktion `fib(n: int) -> int`, welche die `n`-te Fibonaccizahl berechnet und zurückgibt. Die Fibonacci-Zahlen werden wie folgt definiert:

$$\begin{aligned}
\texttt{fib}_0 &= 0 \\
\texttt{fib}_1 &= 1 \\
\texttt{fib}_2 &= \texttt{fib}_0 + \texttt{fib}_1 \\
\texttt{fib}_3 &= \texttt{fib}_1 + \texttt{fib}_2 \\
&\vdots \\
\texttt{fib}_n &= \texttt{fib}_{n-2} + \texttt{fib}_{n-1} \\
&\vdots
\end{aligned}$$

In Ihrer Funktion soll eine Liste der ersten n Fibonaccizahlen aufgebaut werden. Beginnen Sie hierzu mit der Liste `[0, 1]`, welche die nullte und erste Fibonaccizahl enthält. Berechnen sie anschließend aus den beiden vorhandenen Werten die zweite Fibonaccizahl, dann die Dritte, und so weiter. Verwenden Sie in Ihrer Lösung keine `while`-Schleifen und keine Rekursion (noch nicht in der Vorlesung behandelt).

Aufgabe 6.3 (Palindromische Zahlen; Datei: `palindromic.py`; Punkte: 3+2)

Eine palindromische Zahl liest sich in beide Richtungen gleich. Das größte Palindrom aus dem Produkt zweier zweistelliger Zahlen ist $9009 = 91 \cdot 99$. Finden Sie das größte Palindrom, das aus dem Produkt von zwei dreistelligen Zahlen besteht. Verwenden Sie in Ihrer Lösung keine `while`-Schleifen. Gehen Sie wie folgt vor:

(a) Implementieren Sie eine Funktion `is_palindromic(n: int) -> bool`, welche für eine ganze Zahl n zurückgibt, ob diese palindromisch ist. Zum Beispiel:

```
>>> is_palindromic(9009)
True
>>> is_palindromic(101)
True
>>> is_palindromic(35)
False
```

Hinweis: Eine Sequenz `xs` kann mittels *Slicing* umgedreht werden: `xs[::-1]`.

(b) Implementieren Sie eine Funktion `max_palindrome() -> int`, welche die größte palindromische Zahl, die aus dem Produkt zweier dreistelliger Zahlen besteht, berechnet und zurückgibt.

Aufgabe 6.4 (Erfahrungen; Datei: `erfahrungen.txt`; Punkte: 2)

Legen Sie im Unterverzeichnis `sheet06` eine Textdatei `erfahrungen.txt` an. Notieren Sie in dieser Datei kurz Ihre Erfahrungen beim Bearbeiten der Übungsaufgaben (Probleme, Bezug zur Vorlesung, Interessantes, benötigter Zeitaufwand, etc.).