Institut für Informatik
Dr. Tobias Neckel
Alfredo Parra
Paul Sârbu

Technische Universität München
WS 2016/2017

Einführung in die wissenschaftliche Programmierung

Übungsblatt 9

1.) Taschenrechner

In dieser Aufgabe soll ein Taschenrechner erstellt werden. Als Operationen sind nur + und ∗ zugelassen, dafür soll er mit Klammerausdrücken umgehen können.

Grundsätzliches Vorgehen: Das Programm soll zuerst die Eingabe (in Infix-Notation), z.B.

```
( 5 * ( ( ( 9 + 8 ) * ( 4 * 6 ) ) + 7 ) )
```

in die Postfix-Notation umwandeln, d.h.

```
5 9 8 + 4 6 * * 7 + *
```

In der Postfix-Notation steht der Operator nach den Operanden, d.h. `( 4 + 3 )` wird zu `4 3 +`. Der Vorteil ist, dass keine Klammern mehr benötigt werden und sich dann der Ausdruck in Postfix-Notation einfacher auswerten lässt.

Eine nützliche Datenstruktur für die Auswertung dieser Ausdrücke ist ein *Stack* oder *Stapel*. In Python kann ein Stack mit einer Liste dargestellt werden. Dabei entspricht die Operation `push` der Operation `append`. Die Operation `pop` ist bereits eingebaut.

Implementierung: Schreiben Sie eine Funktion `infix2postfix` welche einen Ausdruck (als String) in Infix-Notation in die Postfix-Notation umwandelt. Verwenden Sie einen Stack, und durchlaufen Sie den Ausdruck. Ist das aktuelle Zeichen

i) ein Operator, legen Sie “+” oder “*” auf den Stack,

ii) eine rechte Klammer “)”, dann nehmen Sie ein Element vom Stapel und hängen es an den Postfix-Ausdruck an,

iii) eine Zahl, so hängen Sie diese (als String!) an den Postfix-Ausdruck an,

Leerzeichen und linke Klammern werden ignoriert. Sie können davon ausgehen, dass Sie einen gültigen Ausdruck bekommen, dass Leerzeichen zwischen Operator, Operanden und Klammern sind, und dass nur natürliche Zahlen kleiner 10 vorkommen.

Die Auswertung des Postfix-Ausdrucks funktioniert ganz ähnlich. Verwenden Sie wieder einen Stack und durchlaufen Sie den String Element für Element. Ist das aktuelle Zeichen

i) ein Operator, holen Sie die beiden letzten Elemente vom Stapel (zweimal `pop`), wenden Sie den Operator an und legen Sie das Ergebnis wieder auf den Stapel,

ii) eine Zahl, legen Sie diese auf den Stapel.

Nach diesem Durchlauf, liegt das Ergebnis als einziges Element am Stack.

2.) Partikelsimulation

Im Folgenden wollen wir die Interaktion zweier Partikel simulieren. Wir beschränken uns hierbei auf eindimensionale Betrachtungen. Zwischen den beiden Partikeln wirken Lennard-Jones-basierte Kräfte. Sind die Positionen der Partikel gegeben durch x_1 und x_2, so wirken auf die Partikel die Kräfte

$$\begin{aligned} F_1 &= \frac{24}{(x_2-x_1)^7} \cdot \left(1 - \frac{2}{(x_2-x_1)^6}\right), \\ F_2 &= -F_1. \end{aligned} \tag{1}$$

Als Zeitschrittverfahren kennen wir bereits aus der Vorlesung das explizite Euler-Verfahren. Um die Physik der Partikelinteraktion akkurater zu beschreiben (bspw. hinsichtlich Energieerhaltung), verwenden wir diesmal das sog. Velocity-Störmer-Verlet-Verfahren. Hierbei berechnen wir die Positionen und Geschwindigkeiten $x_i(t + \Delta t)$, $v_i(t + \Delta t)$, $i = 1, 2$, wie folgt aus den Werten des Zeitpunkts t:

$$\begin{aligned} x_i(t + \Delta t) &= x_i(t) + \Delta t v_i(t) + \frac{\Delta t^2}{2} F_i(t) \\ v_i(t + \Delta t) &= v_i(t) + \frac{\Delta t}{2}(F_i(t) + F_i(t + \Delta t)) \end{aligned} \tag{2}$$

i) Erstellen Sie eine Klasse `Particle`. Die Klasse soll als Attribute die Position, Geschwindigkeit und Kraft eines Partikels halten. Diese Klasse enthält keine Methoden.

ii) Schreiben Sie eine Funktion `computeForce(p1,p2)`, welche als Argumente zwei Partikel-Objekte erhält und als Ergebnis die Kraft F_1, siehe Gl. 1, zurückgibt.

iii) Schreiben Sie eine Funktion `timestep(p1,p2,dt)`, die einen Zeitschritt der Länge `dt` für beide übergebenen Partikel ausführt (siehe Gl. 2).

iv) Schreiben Sie eine Funktion `mysim(posX,velX,dt,timesteps,fixP1)`, die die gesamte Simulation ausführt. Ein Partikel p1 soll immer mit Nullgeschwindigkeit bei $x_1 = 0.0$ initialisiert werden, der andere Partikel p2 an der Stelle `posX` mit Geschwindigkeit `velX`. Initialisieren Sie auch die Krafteinträge der Partikel entsprechend. Die Simulation soll `timesteps` Zeitschritte ausführen, jeweils mit Zeitschrittweite `dt`. Der Parameter `fixP1` soll die Werte `True` oder `False` annehmen dürfen. Gilt `fixP1=True`, dann soll Partikel p1 im Ursprung über den Simulationsverlauf fixiert sein.

v) Visualisieren Sie die Geschwindigkeit und Position von Partikel p2 über die Zeit mit Hilfe der Funktion `plot()` des Moduls Matplotlib.

vi) Testen Sie Ihre Implementierung für die folgenden Eingabewerte:

(a) posX=3.0, velX=0.1, dt=0.001, timesteps=1 000 000, fixP1=True

(b) posX=2.0, velX=0.1, dt=0.001, timesteps=100 000, fixP1=True

(c) posX=2.0, velX=1.0, dt=0.001, timesteps=100 000, fixP1=True

(d) posX=2.0, velX=0.1, dt=0.001, timesteps=100 000, fixP1=False

Was beobachten Sie? Welches Verhalten würden Sie unter Verwendung des expliziten Euler-Verfahrens erwarten?