Algorithmen und Programmierung 2, SS 2016 — 6. Übungsblatt

Abgabe bis Freitag, 27. Mai 2016, 12:00 Uhr

Liebe Studierende! Das KVV-System bietet die Möglichkeit, dass Sie *die Lösungen Ihrer Mitstudierenden bewerten und durch das Lesen von anderen Lösungen und den Vergleich mit Ihrer eigenen Lösung ein andersartiges Lernerlebnis erfahren. Wir wollen das bei Aufgabe 28 zum ersten Mal ausprobieren. Ihre Teilname am Bewertungsverfahren wird mit Übungspunkten honoriert. In der Vorlesung am Mittwoch, den 25. Mai erfahren Sie genaueres zum Ablauf, und es wird auch schriftliche Anleitungen und Hilfestellungen geben.*

27. Objekte und Klassen, 8 Punkte

    Definieren Sie folgende Begriffe in 1–2 Sätzen, und illustrieren Sie sie jeweils mit einem kurzes Programmbeispiel in Python.

    (a) Objekt (Exemplar) einer Klasse
    (b) Vererbung
    (c) Überschreiben von Attributen und Methoden
    (c) Aufruf einer Methode
    (d) Unterklasse und Oberklasse

28. Wettrennen, 15 Punkte

    (a) Definieren Sie eine Unterklasse `Verfolger` von `Turtle`, die bei der Konstruktion eine *Zielschildkröte* als Parameter erhält: `t=Verfolger(ziel)`. Mit der Methode `t.verfolge(Schrittweite)` soll `t` einen Schritt der Länge `Schrittweite` auf das *Ziel* zu machen.

    (b) Definieren Sie eine Klasse `Rechteck` mit dem Konstruktionsaufruf

    ```
    Rechteck(l,r,u,o)
    ```

    und einer Methode `zeichne`, die das Rechteck zeichnet.

    (c) Verändern Sie die Klasse `Wegläufer` aus der Vorlesung, sodass sie bei der Konstruktion ein Rechteck als zusätzlichen optionalen Parameter *Begrenzung* akzeptiert:

    ```
    def __init__(self, wovor, Begrenzung=None):
    ```

    Die Methode `lauf_weg` soll so modifiziert werden, dass die Schildkröte das Rechteck nicht verlässt, sich aber dabei innerhalb der `Schrittweite` möglichst weit weg vom gefürchteten Verfolger bewegt.
    Sie müssen den letzen Satz dieser Vorgabe nicht buchstäblich umsetzen; aber die Schildkröte darf nicht einfach stehenbleiben, wenn sie am Rand der Begrenzung ist. *Spezifizieren Sie genau*, was Ihre `Wegläufer`-Schildkröte in jedem Fall macht. Schreiben Sie die Spezifikation als Kommentar in Ihr Programm.

    Vorerst sollen Sie bei der Bearbeitung dieser Aufgabe bloß folgendes beachten: Die Bewertung soll anonym erfolgen; schreiben also keine Namen in die Dateien, und nennen Sie Ihr Programm bloß `Aufgabe27.py`*, abweichend von den generellen Vorgaben. Kommentieren Sie besonders gut, damit Ihr Programm angenehm zu begutachten ist.*

29. Knobelaufgabe, 0 Punkte

    Stellen Sie mit den Zahlen $1, 5, 6, 7$ und den Grundrechnungsarten $+, -, \times, /$ die Zahl 21 dar. Jede Zahl muss genau einmal verwendet werden. Klammern sind erlaubt, aber Tricks, wie zum Beispiel 15 aus 1 und 5 zu bilden, sind verboten.[1]

[1] Die meisten von Ihnen werden das Problem schneller mit dem Computer durch Probieren aller Möglichkeiten lösen als per Hand.

30. (a) Jäger und Beute, 0 Punkte
Ein Adler A mit Geschwindigkeit 2 möchte möglichst schnell zwei Spatzen S_1 und S_2 fangen. Die Spatzen fliegen stets mit konstanter Geschwindigkeit 1 von der aktuellen Position des Adlers weg. Der Adler könnte zunächst gradlinig den näheren der beiden Spatzen fangen und von dort aus geradlinig auf den anderen zusteuern. Schreiben Sie ein Programm, dass dieses Verhalten mit kleinen Zeitschritten simuliert und graphisch visualisiert.

(b) (Forschungsaufgabe, Antwort unbekannt) Ist diese Strategie immer optimal? Können Sie Ausgangspositionen finden, wo es eine schnellere Strategie gibt?

31. Objekte und Referenzen, 7 Punkte

Was ist die Ausgabe von untenstehendem Programm? Erklären Sie, was passiert. Wie kann man am Ende auf das Element '88' zugreifen?

```
def f1(a):
  a = a + [a]
def f2(a):
  b = a
  b.append(7)
def f3(a):
  b = a + ['88']
  a.append([a,b])
a=[4,5,6]
f1(a)
print(a)
f2(a)
print(a)
f3(a);print(a)
```

32. Elimination von `goto`-Anweisungen, 0 Punkte

Das nebenstehende Programm wurde 1962 in der Programmiersprache ALGOL 60 veröffentlicht[2] und von mir fast eins-zu-eins nach PYTHON übersetzt. Das Programm funktioniert so:

> Each call of *PERM* changes the order of the first n components of x, and $n!$ successive calls will generate all $n!$ permutations. The parameter '*first*' must be *True* when *PERM* is first called, to cause proper initialization. Successive calls must leave *first* = *False*. On exit from the $(n!)$-th call of *PERM*, the function returns *True*.

Obwohl es damals schon `for`- und `while`-Schleifen gab, sehen Sie noch jede Menge `goto`-Anweisungen.

```
def PERM(x,n,first=False):
   global p,d
   done = False
   if first then
initialize:
      p = (n+1)*[0]
      d = (n+1)*[1]
   k=0
INDEX:
   p[n] = q = p[n] + d[n]
   if q == n:
      d[n] = -1
      go to LOOP
   if q != 0:
      go to TRANSPOSE;
   d[n] = 1
   k = k + 1
LOOP:
   if n > 2:
      n = n - 1
      go to INDEX
Final exit:
   q = 1
   done = True
TRANSPOSE:
   q = q + k
   x[q-1],x[q] = x[q],x[q-1]
   return done
"end PERM;"
```

(a) Bringen Sie das Programm in eine vernünftige Struktur. Den Bezug auf globale Variablen und die Aufrufkonventionen können Sie unverändert lassen.

(b) Verstehen Sie, was das Programm macht.

[2] H. F. Trotter, Algorithm 115: PERM, *Communications of the ACM* **5** (1962), 434–435. doi:10.1145/368637.368660, siehe `http://www.inf.fu-berlin.de/lehre/SS16/ALP2/PERM.py`