Python - Übungsblatt 5

Gerald Senarclens de Grancy

March 3, 2007

Übung 1 - Wiederholung Rekursion

Grösster gemeinsamer Teiler

Der Algorithmus von Euklid ist definiert durch `gcd(0,n)=n` und `gcd(m,n)=gcd(n % m,m)`. Definiere eine Funktion in Python, die den "greatest common devisor" (gcd) zurückgibt.

Fibonacci Zahlen

Jede Fibonacci Zahl ist definiert als Summe der zwei vorherigen Fibonacci Zahlen. Die ersten zwei Zahlen sind 0 und 1. Definiere eine Funktion, die die n-te Fibonacci Zahl berechnet. Nachdem diese Funktion korrekt arbeitet schreibe eine weitere Funktion `def fibList(n):`, die unter Verwendung einer for-Schleife die ersten n Fibonacci Zahlen als eine Liste zurückgibt. Zur Lösung dieser Funktion kannst du eine `if` Bedingung nach folgender Form verwenden:

```
if condition:
    if-Block
elif condition:
    elif-Block
else:
    else-Block
```

Übung 2 - Ein statistisches Modul

Zur Vertiefung des Stoffes soll ein Modul für statistische Funktionen geschrieben werden. Die hier angegebenen Funktionen bauen aufeinander auf und sollen möglichst selbständig erstellt werden.

Erstelle nocheinmal eine Funktion, die die Faktorielle einer gegebenen Zahl zurückgibt, da dies die Basis vieler statistischer Funktionen ist. Erstelle weiters eine Funktion, die den Binomialkoeffizienten $\binom{n}{k} = \frac{n!}{k!*(n-k)!}$ berechnet. Danach erstelle eine Funktion, die die Binomialverteilung (probability mass function, pmf) berechnet ($P(k) = \binom{n}{k} * p^k * (1-p)^{(n-k)}$). Insgesamt sollen folgende Funktionen mit den zugehörigen Docstrings definiert werden:

```
def fact(num):
def binom(n,k):
def pmf(n,k,p):
```

Erstelle weiters Funktionen mit einer unbestimmten Anzahl an Argumenten `def function(*arg):`, wobei `*arg` eine Liste von Argumenten ist. Zur Übung erstelle Funktionen zur Berechnung des arithmetischen Mittelwertes $\overline{x}$[1], des geometrischen Mittelwertes ${x_{geo}}^2$ und der Standardabweichung σ[3]. Die Kopfzeilen der Funktionen sollen wie folgt aussehen:

```
def mean(*values):
def meanGeo(*values):
def stddev(*values):
```

Übung 3 - Eine erste Klasse

Objektorientierte Programmierung hat, wenn sie richtig angewendet wird, einige Vorteile gegenüber funktionaler Programmierung. Am wichtigsten ist dabei, dass man als Mensch kognitiv am ehesten ein Vorstellungsvermögen für Objekte hat. Im folgenden definieren wir nun eine erste Klasse, wobei Klassen in der objektorientierten am besten als "Objektfabriken" zu verstehen sind. Um bei der Klassendefinition auf das jeweilig erstellte Objekt zu referenzieren, verwendet Python das `self` Schlüsselwort - es ist jeder Klassenfunktion als erstes Argument "zu übergeben", die etwas an einem Objekt verändern bzw. darüber in Erfahrung bringen soll. Bei anderen Objektorientierten Programmiersprachen ist ein etwas anderer Mechanismus vorgesehen, der im Grunde aber sehr ähnlich funktioniert.

Definiton eines Autos

Ein Auto ist zum Beispiel ein Objekt aus dem täglichen Leben. Um eine Klasse zu definieren, die Autoobjekte erzeugt, verwende folgenden Code:

```
class Car():
   def setspeed(self, speed):
      self.speed = speed
   def getspeed(self):
      return self.speed
```

[1] $$\overline{x} = \frac{1}{n} * \left(\sum_{i=1}^{n} x_i \right)$$

[2] $$x_{geo} = \left(\Pi_{i=1}^{n} x_i\right)^{1/n}$$

[3] $$\sigma = \sqrt{\frac{1}{n} * \sum_{i=1}^{n} (x_i - \overline{x})^2}$$

Erzeugen von Objekten

Nun kannst du deine neue Klasse benützen, indem du Objekte durch sie erstellen lässt. Ein Autoobjekt wird durch `myCar = Car()` hergestellt. Weitere Autos kann man zum Beispiel mittels `anotherCar = Car()` herstellen. Nun kann man die Autos auch fahren lassen:
`myCar.setspeed(30)`
`anotherCar.setspeed(50)`
Wenn man nun an eine Ampel kommt, sollte man früh genug zu bremsen beginnen. Dazu möchte man zuerst die Geschwindigkeit seines Autos wissen
`myCar.getspeed()`
bevor man auf die Bremse steigt `myCar.setspeed(0)` Danach kann man wieder losfahren usw.