Wiederholung

Philipp Hanisch, Valentin Roland

14. Juni 2018

Python-Grundlagen

Basics

Kontrollstrukturen in Python

Python kennt:

- Ein- / Zweiseitige Verzweigung: `if .. : / if .. : .. else: ..`
- Zählschleife: `for .. in .. :`
- Kopfgesteuerte Schleife: `while .. :`

Was sind Listen?

Wie alltägliche Listen. In Python:

- `[]`, bzw. `[1,2,3]`
- Methoden: `insert()`, `append()`, `pop()`, `remove()`, `index()`
- siehe auch `help([])`

Wie verwende ich Listen

```
1    a = [1,5,3,2] # create new list
2    a.sort() # a is now sorted
3    print (a[3]) # 4th element of sorted list -> 5
4    highest = a.pop() # removes last element (5) and returns it
5    highest += 1
6    a.insert(0, highest) # add 6 at the start
7    print (a) # -> [6,1,2,3]
8
```

For-Schleife

Klassische Zählschleife gibt es nicht in Python

↪ `for` durchläuft Listen

```
1    a = [1,2,3]
2    for elem in a:
3        print (elem)
4
```

Objektorientierung

„Objektorientierung“

- Software als interagierende „Objekte“
- Objekte haben
  - Eigenschaften (**Attribute**)
  - Verhalten (**Methoden**)

Klassen

- Oft mehrere Objekte einer Art gleich $\rightarrow$ Klassen
- Attribute, Methoden bei allen Objekten einer Klasse gleich
- Objekt ist **Instanz** einer Klasse

Eigene Klassen

```
class Wuerfel:
    # VORSICHT: keine (Instanz-)Attribute hier!

    # spezielle Methode: Konstruktor.
    # wird beim Erstellen einer Instanz aufgerufen
    def __init__(self, seiten):

        # neues Attribut "seiten" wird erzeugt.
        self.seiten = seiten

    # Methoden haben immer "self" als ersten Parameter
    # -> können so auf eigene Attribute zugreifen
    def wuerfeln(self):
        return random.randInt(1, self.seiten)

```

Eigene Klassen

```
1          d6 = Wuerfel(6)
2          d20 = Wuerfel(20)
3
4          wert = d20.wuerfeln()
5          print ("ich habe eine {} gewürfelt!".format(wert))
6
```

Module

Module einbinden

```
1  from random import randint
2  print(randint(1, 10))
3
4  from wuerfel import *
5  d20 = Wuerfel(20)
6  print(d20.wuerfeln())
7
8  import spieler
9  spieler = spieler.Spieler("Arndt", d20)
10 print(spieler.name)
11
```

Boilerplate

```
class Wuerfel:
    def __init__(self, seiten):
        self.seiten = seiten

    # weitere Methoden

if __name__ == '__main__': # Boilerplate
    # Code wird beim Importieren nicht ausgeführt
    d20 = Wuerfel(20)
    print(d20.wuerfeln())

```

`__name__` enthält den Namen des Scriptes beim Importieren oder `'__main__'`, wenn das Script direkt ausgeführt wird.

Aufgaben

Aufgaben

1. Erweitere die Spielerklasse, sodass ein Spieler mehrere Würfel besitzen kann. Wenn er würfelt, so soll er mit allen Würfeln nacheinander würfeln und eine Ergebnisliste zurückbekommen.
2. Modelliert das Würfelspiel als eigene Klasse.
   - 2.1 Das Spiel sollte (min.) zwei Spieler umfassen, die in Runden gegeneinander spielen.
   - 2.2 Wie ist der jeweilige Zwischenstand?
   - 2.3 Wie sieht eine Runde aus? Würfeln die Spieler mit einem Würfel? Mit mehreren? Gewinnt die höchste Summe? Oder der höchste Wert?
   - 2.4 Strukturiert euren Code so, dass die Änderung, was eine Runde ist, möglichst wenig Änderungen im Code bedeutet.

Aufgaben

4. Modifiziert eure Spieler, um folgende Sachverhalte zu modellieren.
   - 4.1 Wenn ein Spieler mit *Vorteil* würfelt, so würfelt er zweimal und der höhere Wert zählt. Gleichermaßen kann er einen *Nachteil* haben (niedrigerer Wert). Vorteil und Nachteil gleichzeitig führen zu einem normalen Würfelwurf.
   - 4.2 Ein Spieler verfügt über einen *Bonus*, der zu jedem Wurf dazu addiert wird.
   - 4.3 Ein Spieler kann seinen Bonus erhöhen, Vorteil oder Nachteil erhalten oder verlieren.
5. Wie viel Bonus braucht ein Spieler, damit das Spiel ausgeglichen ist, wenn sein Gegner Vorteil auf jeden Wurf hat? Wie viel, wenn unser Spieler gleichzeitig Nachteil hat?
6. Überlegt euch weitere (interessante) Würfelspiele.