Objektorientierung

Claas de Boer, Tilman Hinnerichs

25. November 2020

Python-Grundlagen

Aufgaben 4.0

Schreibe ein Programm, welches

1. alle geraden Zahlen einer Zahlenliste in einer zweiten Liste speichert und stoppt, sobald eine 237 vorkommt
2. alle Elemente einer Liste von Strings zusammenfügt und ausgibt
3. als Input den Namen und Heimatplaneten des Nutzers abfragt und diesen nett grüßt
4. die Seitenlänge h und zugehörige Höhe h_c eines Dreiecks einliest und den Flächeninhalt ausgibt
5. die Quersumme einer Zahl berechnet

Nachtrag zu Listen: List comprehensions

```
1    L = []
2    for num in list_of_numbers:
3        if num%2 == 0:
4            L.append(num+1)
```

```
1    L = [num+1 for num in list_of_numbers if num%2==0]
```

Gliederung

1. Was ist „Objektorientierung“?

2. Objektorientierung in Python

3. Aufgaben

Was ist „Objektorientierung“?

„Objektorientierung“

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

Wozu?

- „intuitiver“Entwurf
- Gruppierung von Daten und Verhalten $\rightarrow$ Klasse
- Organisation großer Softwaresysteme
- $\rightarrow$ In Python grundlegend verankert

Beispiel

`random.randInt(min, max)` + Variablen → Objekt „Würfel“

- Attribute: `seiten`
- Methoden: `wuerfeln() -> int`

Klassen

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

Objektorientierung in Python

Nutzen von Objekten

- Wir kennen bereits Klassen und Objekte:
  - `"Hello World"` ist **Instanz** von `str`
  - `[1,2]` ist **Instanz** von `list`
- Funktionen `sum([1,2,3])` $\leftrightarrow$ Methoden `[1,2,3].append(4)`!

Nutzen von Objekten

```
1  # Erstellen eines neuen Objekts der Klasse list
2  l = [1,2,3]
3  # l ist tatsächlich Instanz von list
4  print(isinstance(l, list))
5
6  # Aufrufen der Methode append
7  l.append(4)
8  print (l) # Daten in der Liste wurden geaendert
9
10 print(len(l)) # len ist (eingebaute) Funktion, nicht Methode ->
       NICHT l.len!
11
```

Eigene Klassen

```
1  class Wuerfel:
2      # VORSICHT: keine (Instanz-)Attribute hier!
3
4      # spezielle Methode: Konstruktor.
5      # wird beim Erstellen einer Instanz aufgerufen
6      def __init__(self, seiten):
7
8          # neues Attribut "seiten" wird erzeugt.
9          self.seiten = seiten
10
11     # Methoden haben immer "self" als ersten Parameter
12     # -> koennen so auf eigene Attribute zugreifen
13     def wuerfeln(self):
14         return random.randInt(1, self.seiten)
15
```

Eigene Klassen

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

Aufgaben

Aufgaben

1. Implementiere die Würfelklasse, erzeuge einen 20-seitigen Würfel und würfle. Falls du eine 20 würfelst, rufe laut "20"!
2. Implementiere ein Würfelspiel, bei dem du und der Computer einmal würfeln. Wer die höhere Zahl würfelt, gewinnt!
3. Schreibe eine Klasse „Spieler“, die einen Spieler bei diesem Würfelspiel darstellt. Welche Attribute und Methoden hat ein Spieler?
4. Erweitere dein Würfelspiel, sodass zwei Spielerobjekte gegeneinander spielen.
5. Zwei Spieler sind zu langweilig? Mit der Spielerklasse kannst du einfach mehr Spielerobjekte erzeugen. Erweitere dein Spiel, so dass beliebig viele Spieler spielen können (s. auch listen (letzte Stunde))
6. Auch abstrakte Konzepte können als Klasse modelliert werden. Wie könnte man ein Würfelspiel mit mehreren Runden als Klasse darstellen?