Objektorientierung (II) und Modularisierung

Philipp Hanisch, Valentin Roland

6. Oktober 2022

Python-Grundlagen

Gliederung

1. Rückblick: Objektorientierung
2. Modularisierung
3. Aufgaben

Rückblick: Objektorientierung

Objekte und Klassen

- **Objekte** haben Eigenschaften (**Attribute**) und Verhalten (**Methoden**)
- Objekte als **Instanzen** (Ausprägungen) einer **Klasse**
- Klassen als „Schablone“ oder „Bauplan“ für Objekte
- Objekte einer Klasse haben gleiche Attribute, die einen unterschiedlichen Wert haben können

Aufgabenstellung

1. Implementiere die Würfelklasse und erzeuge einige Würfel und würfel!
2. Schreibe eine Klasse „Spieler“!
3. Erweitere dein Würfelspiel, sodass zwei Spielerobjekte gegeneinander spielen.
4. Auch abstrakte Konzepte können als Klasse modelliert werden. Wie könnte man ein Würfelspiel mit mehreren Runden als Klasse darstellen?

Lösung

```
1  from random import randint
2
3  class Wuerfel:
4      def __init__(self, seiten):
5          self.seiten = seiten
6
7      def wuerfeln(self):
8          return randint(1, self.seiten)
9
10
11 d6 = Wuerfel(6)
12 d20 = Wuerfel(20)
13
14 print("Du hast eine {} gewürfelt!".format(d20.wuerfeln()))
15
16 d20.seiten = 12
17 print(d20.seiten)
18
```

Lösung

```
1  class Spieler:
2      def __init__(self, name, wuerfel):
3          self.name = name
4          self.wuerfel = wuerfel
5
6      def wuerfel_wecheln(self, wuerfel):
7          self.wuerfel = wuerfel
8
9      def wuerfeln(self):
10         return self.wuerfel.wuerfeln()
11
12
13 d20 = Wuerfel(20)
14 spieler = Spieler("Laucian", d20)
15
16 wurf = spieler.wuerfeln()
17 print("{} hat eine {} gewürfelt!".format(spieler.name, wurf))
18 spieler.wuerfel = Wuerfel(12)
19 ^^I
```

Modularisierung

Motivation

- Struktur und Übersichtlichkeit
- Wiederverwendung (z.B. mathematische Funktionen)
- Nutzung von Bibliotheken (z.B. random)
- (einheitliche) Definition von Konstanten, Klassen, etc.

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 ^^I
```

Boilerplate

```
class Wuerfel:
    def __init__(self, seiten):
        self.seiten = seiten
^^I
    # weitere Methoden

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

`__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.