Vorsemesterkurs
Informatik
Übungsaufgaben

Übungszettel 6b - Python OOP

Falls du heute lieber noch Aufgaben von den vorherigen Übungszetteln bearbeiten möchtest, darfst du das natürlich auch gerne tun.

Aufgabe 1: Objektorientierte Programmierung

Betrachte folgende Klasse "Student".

```
1  class Student:
2      'contains students, email, name ...'
3
4      def __init__(self, name, kurse=None):
5          self.name = name
6          if kurse == None:
7              self.kurse = []
8          else:
9              self.kurse = kurse
10         print("Klasseninstanz angelegt für: ", name)
11         return
12
13     def printDetails(self):
14         print("Name:",self.name)
15         print("Kurse: ", self.kurse)
16         return
17
18
19     def einschreiben(self, kurs):
20         self.kurse.append(kurs)
21         return
```

(a) Füge weitere Attribute hinzu (Telefonnummer, e-mail Adresse,...). Du kannst default-Werte für Parameter angeben, indem du ein Gleicheitszeichen (=) benutzt, wie es in der init-Methode der Klasse "Student" beim Parameter "Kurse" gemacht wurde.

(b) Schreibe ein Programm, welches eine Studentin namen "Marie" anlegt. Marie ist bereits in den Kurs mit Kursnummer "K374" eingeschrieben, möchte aber noch weitere Kurse besuchen. Gib den Nutzer*innen also die Möglichkeit weitere Kurse einzutragen. Wenn alle Kurse eingetragen sind, soll das Programm alle Daten von Marie auf dem Bildschirm ausgeben.

(c) Lege eine Liste von Studierenden an. Die Nutzer*innen sollen neue Studierende zur Liste hinzufügen können. Das Programm soll die Nutzer*innen nach dem Namen einer*s Studierenden fragen, und dann die Daten der*s Studierenden ausgeben.

(d) Füge eine Methode creditPoints hinzu, die die Creditpoints eines Studierenden berechnet, unter der Annahme, dass jeder Kurs 3CP hat.

(e) Programmiere eine Klasse "Angestellte", die Informationen über Namen, Alter und Position enthält. Welche Methoden und weitere Attribute könnten für diese Klasse nützlich sein?

Solution:

(a)
```python
class Student:
    'contains students, email, name ...'

    def __init__(self, name, kurse = None, phone = 'none given', email=
        'no email'):
        self.name = name
        if kurse == None:
            self.kurse = []
        else:
            self.kurse = kurse
        self.phone = phone
        self.email = email
        print("Klasseninstanz angelegt für: ", name)
        return

    def printDetails(self):
        print("Name:",self.name)
        print("Kurse: ", self.kurse)
        print("Phone: ", self.phone)
        print("Email: ", self.email)
        return


    def einschreiben(self, kurs):
        self.kurse.append(kurs)
        return

    def creditPoints(self):
        return(len(self.courses)*3)
```

(b)
```python
from student import Student

student1 = Student("Marie", ["K374"])

print("Geben Sie die Kurse ein in die ", student1.name, "
    eingeschrieben ist. ")
newcourse = input("Geben Sie eine Kursnummer oder 'stop' ein: ")

while newcourse != 'stop':
    student1.einschreiben(newcourse)
    newcourse = input("Geben Sie eine Kursnummer oder 'stop' ein: ")

student1.printDetails()
```

(c)
```python
from student import Student

students = []
again = "yes"

while again != "nein":
    name = input("Geben Sie den Namen des Studierenden ein: ")
    phone = input("Geben Sie die Telefonnummer des Studierenden ein: ")
    email = input("Geben Sie die E-mail Adresse des Studierenden ein: "
        )
    student = Student(name, [], phone, email)
    students.append(student)

    print("Geben Sie die Kurse ein, die ", student.name, "belegt.")
    newcourse = input("Geben Sie eine Kursnummer oder 'stop' ein: ")

```

```
    while newcourse != "stop":
        student.einschreiben(newcourse)
        newcourse = input("Geben Sie eine Kursnummer oder 'stop' ein: "
            )

    again = input("Möchten Sie noch mehr Studierende eingeben? ")

for student in students:
    student.printDetails()
```

(d) Ist oben schon drin

(e) Sieht bei jedem anders aus.

Aufgabe 2: OOP-Komposition

(a) Implementiere eine Klasse "Author", die die drei privaten Attribute `name, geschlecht` und `email`, welche im Konstruktor initialisiert werden, besitzt. Es soll auch möglich sein, Autor*innen ohne E-mail-Adresse anzulegen (Tipp: default-Wert setzen). Ferner soll die Klasse die Methoden `getName(), getEmail(), setEmail(emailadresse)` und `setGender()`, sowie eine Methode `info()` haben, die den String "AuthorName (Geschlecht) at AuthorEmail" zurückliefert, besitzen.

(b) Schreibe ein Programm "testAuthor.py" das den Konstruktor und die öffentlichen (public) Methoden der Klasse testet; E-mail Adresse ändern, Namen ausgeben, etc.

(c) Implementiere eine Klasse "Buch"(die die Authorklasse verwendet), welche die privaten Attribute `titel, author, preis` besitzt. Alle drei Attribute sollen im Konstruktor initialisiert werden. `titel` und `author` sind unveränderbar, der Preis soll sich aber ändern können. Die Klasse soll folgende öffentliche Methoden besitzen: `getName(), getAutor(), getPreis(), setPreis(), info()`. Die `info()`-Methode soll einen String "buchName von autorName (geschlecht) at email" zurückgeben. (Beachte, dass die `info()`-Methode der Klasse "Autor" "AutorName (geschlecht) at AutorEmail" liefert).

(d) Schreibe ein Programm "testBuch.py" um den Konstruktor und die öffentlichen Methoden der Klasse "Buch" zu testen. Beachte, dass du zunächst eine Instanz der Klasse "Autor" anlegen musst.

(e) Beachte, dass sowohl "Buch" als auch "Author" ein Attribut `name` besitzt. Sie können durch die referenzierende Instanz unterschieden werden. Für eine Buchinstanz `buch1` z.B. liefert der Aufruf `buch1.name()` den Namen des Buches, während für die Authorinstanz `author1`, der Aufruf `author1.name()` den Namen der*s Author liefert.
Versuche die Werte `name` und `email` des Author von der Buchinstanz aus aufzurufen.
(Hinweis: nutze das Attribut `autor`)

(f) Erweitere die Klasse "Buch" um die Methoden `getAuthorName(), getAuthorEmail(), getAuthorGender()`, um `name, email` und `geschlecht` des Authors des Buches zurückzugeben.

Solution:

(a)
```
#Author und Buch Klassen

class Author:
    'Author mit Name, e-mail und Geschlecht'

    def __init__(self, name, gender, email='no email'):
        self.__name = name
        self.__gender = gender
        self.__email = email
        print("Author angelegt: ", name)
```

```
        return

    def setEmail(self,email):
        'speicher e-mail-Adresse'
        self.__email = email
        return

    def getName(self):
        return(self.__name)

    def getEmail(self):
        return(self.__email)

    def getGender(self):
        return(self.__gender)

    def info(self):
        return(self.__name+' (' +self.__gender + ') at ' + self.__email
            )

class Buch:
    'Buch mit Titel, Author, Preis'

    def __init__(self, name, author, preis):
        self.__name = name
        self.__author = author
        self.__preis = preis
        print('Buch angelegt: ', name)
        return

    def getName(self):
        return(self.__name)

    def getAuthor(self):
        return(self.__author)

    def getPreis(self):
        return(self.__preis)

    def setPreis(self, preis):
        self.__preis = preis
        return

    def info(self):
        return(self.__name + ' von ' + self.__author.info())

    def getAuthorName(self):
        return(self.__author.getName())

    def getAuthorEmail(self):
        return(self.__author.getEmail())

    def getAuthorGender(self):
        return(self.__author.getGender())
```

(b)

```
from author import Author

a1 = Author('Swaroop', 'm')

print(a1.info())
```

```
a1.setEmail("swaroop@me.com")

print(a1.info())
print("Der Name der*s Authors ist: ", a1.getName())
print("Author ist: ", a1.getGender())
print("Die Email-Adresse lautet: ", a1.getEmail())
```

(c) oben schon drin

(d)
```
from author import Author, Buch

a1 = Author('Swaroop', 'm')

print(a1.info())
a1.setEmail("swaroop@me.com")

print(a1.info())
print("Der Name der*s Autors ist: ", a1.getName())
print("Author ist: ", a1.getGender())
print("Die Email-Adresse lautet: ", a1.getEmail())

b1 = Buch("A byte of Python", a1, 17.90)

print(b1.info())
b1.setPreis(12.95)
print(b1.info())
print("Der Titel des Buches lautet: ", b1.getName())
print("Author des Buches ist: ", b1.getAuthor().info()) # Zugriff auf
    Autor über Buch
print("Das Buch kostet: ", b1.getPreis())
print("Der Name der*s Author lautet: ", b1.getAuthorName())
print("Die E-Mail-Adresse der*s Author lautet: ", b1.getAuthorEmail())
print("Author ist: ", b1.getAuthorGender())
```

Aufgabe 3: Rock, Paper, Scissors schwierig

"Rock, Paper, Scissors" ist ein bekanntes rundenbasiertes Knobelspiel.

Jede*r Spielende wählt zu Beginn einer Runde eines der folgenden Objekte Rock, Paper, Scissors. Wählen beide Spielende das gleiche Objekt, gilt die Runde als unentschieden. Sonst wird nach folgenden Regeln bestimmt, wer gewonnen hat:

- Scissors cuts paper
- Paper covers rock
- Rock crushes scissors

(a) Setze dieses Spiel in Python um. Die Nutzer*innen sollen über eine textuelle Benutzungsschnittstelle gegen den Computer knobeln können.

(b) Erweitere das Spiel um die Objekte Lizard, Spock.
Die Gewinnregeln müssen um die folgenden Regeln erweitert werden:

- Lizards posions spock
- Spock smashes scissors
- Scissors decapitates lizard
- Lizard eats paper
- Paper disproves Spock
- Spock vaporizes rock
- Rock crushes lizard

Solution:

(a) Hier eine beispielhafte Implementierung des Programms für Aufgabe 2:

```
#!/usr/bin/env python
# encoding: utf-8
"""
RockPaperScissors.py

Created by Johannes Seitz on 2010-07-30.
"""

import random

VALID_OPTIONS = ("scissors", "paper", "rock", "lizard", "spock")
RULES = ["scissors cuts paper",
         "paper covers rock",
         "rock crushes lizard",
         "lizard poisons spock",
         "spock smashes scissors",
         "scissors decapitates lizard",
         "lizard eats paper",
         "paper disproves spock",
         "spock vaporizes rock",
         "rock crushes scissors"]

def main():
    """Mainloop"""
    user_score = computer_score = 0
    while True:
        user_choice = input("Choose your weapon: ").lower().strip()
        if user_choice in VALID_OPTIONS:
            user_score, computer_score = \
            play_game(user_choice, user_score, computer_score)
        elif user_choice == "exit":
            exit()
        elif user_choice == "rules":
            for rule in RULES:
                print("\t%s" %rule.capitalize())
        else:
            print("\tType one of the following:\n \
            \t%s, %s, %s, %s, %s, rules or exit." %VALID_OPTIONS)

def play_game(users_choice, user_score, computer_score):
    """The actual game"""
    computers_choice = random.choice(VALID_OPTIONS)
    if computers_choice == users_choice:
        print("\t%s ties with %s".capitalize() \
              %(users_choice, computers_choice))
    else:
        rule = rule_for_combination(users_choice, computers_choice)
        print("\t%s" %rule.capitalize())
        if rule.startswith(computers_choice):
            computer_score += 1
            print("\tYOU LOSE!"),
        else:
            user_score += 1
            print("\tYOU WIN!"),
        print("You: %s, Computer: %s" %(user_score, computer_score))
    return (user_score, computer_score)

```

```
58 def rule_for_combination(choice1, choice2):
59     """Looks up the rule for a given combination"""
60     for rule in RULES:
61         if choice1 in rule and choice2 in rule:
62             return rule
63 
64 if __name__ == '__main__':
65     main()
```

Viel Erfolg!