Prescriptive Analytics, Seminar 2

Henry Haustein

Aufgabe 1: Podcast-Management

(a) Klasse `Podcast` gleich mit privaten Attributen:

```
1  class Podcast():
2    # nicht nötig, aber finde ich besser lesbar, insb. bei größeren Klassen
3    __episode = None
4    __length = None
5    __moderator = None
6    __adverts = None
7    __title = None
8
9    def __init__(self, e, t, l, m):
10     self.__episode =int(e)
11     self.__length = int(l)
12     self.__moderator = str(m)
13     self.__title = str(t)
14     self.__adverts = 0
15
16   def display(self):
17     print("Podcast Folge Nr.: {} mit ModeratorIn: {}".format(self.__episode, self.__moderator))
18     print("Thema der Podcastfolge: {}".format(self.__title))
19     print("Länge der Podcastfolge: {}".format(self.__length))
20     print("Aktuell verkaufte Werbeblöcke: {}".format(self.__adverts))
21
22   def setAdverts(self, werbebloecke):
23     self.__length = self.__length + 3*werbebloecke
24     self.__adverts = self.__adverts + werbebloecke
25
26   def getAdverts(self):
27     print(self.__adverts)
28
29   def cut(self, minuten):
30     if self.__length - minuten <= 30:
31       print("Keine Kürzung möglich")
32     else:
33       self.__length = self.__length - minuten
```

```
34
35 newpodcast = Podcast(23, "Traveling Salesman Problems", 70, "O
       . Peratio")
36 newpodcast.display()
```

(b) siehe (a)

(c) Klasse `SpecialPodcast`

```
1  class SpecialPodcast(Podcast):
2    __specialGuest = None
3
4    def __init__(self, e, t, l, m, s):
5      # Alternative 1
6      self.__episode = int(e)
7      self.__length = int(l)
8      self.__moderator = str(m)
9      self.__title = str(t)
10     self.__adverts = 0
11     # Alternative 2 (funktioniert nicht, weil Attribute von
           Podcast privat sind)
12     # Podcast(e, t, l, m)
13     self.__specialGuest = s
14
15   def changeGuest(self, newGuest):
16     self.__specialGuest = newGuest
17
18   def display(self):
19     print("Podcast Folge Nr.: {} mit ModeratorIn: {}".format(
           self.__episode, self.__moderator))
20     print("Thema der Podcastfolge: {}".format(self.__title))
21     print("Länge der Podcastfolge: {}".format(self.__length))
22     print("Aktuell verkaufte Werbeblöcke: {}".format(self.
           __adverts))
23     print("SpecialGuest der Podcastfolge ist: {}".format(self.
           __specialGuest))
24
25 sp = SpecialPodcast(1, "Vehicle Routing Problem", 65, "M.
       Ipler", "Dr. Best")
26 sp.changeGuest("Dr. Secondbest")
27 sp.display()
```

Aufgabe 2: Einführung des Planungsproblems

(a) mittels `with` (https://preshing.com/20110920/the-python-with-statement-by-example/)

```
1 import json
2
3 with open('InputFlowshopSIST.json') as json_file:
4   data = json.load(json_file)
5   print(data)
```

Das `with`-Statement wird gerne bei der Arbeit mit Dateien genommen, da man nach dem Öffnen einer Datei diese auch wieder schließen müsste. Aber das wird häufig vergessen und `with` macht das automatisch im Hintergrund. Man hätte also auch schreiben können:

```
import json

json_file =  open('InputFlowshopSIST.json')
data = json.load(json_file)
print(data)
json_file.close()
```

(b) Ich habe mich hier für die Klassen `Job`, `Maschine` und `Flowshop` entschieden:

```
class Job:
  id = None
  setupTimes = None
  processingTimes = None
  dueDate = None
  tardCosts = None

  def __init__(self, id, setup, processing, due, costs):
    self.id = int(id)
    self.setupTimes = setup
    self.processingTimes = processing
    self.dueDate = due
    self.tardCosts = costs

  def __str__(self):
    return f"Job({self.id = }, {self.setupTimes = }, {self.
        processingTimes = }, {self.dueDate = }, {self.
        tardCosts = })"

class Maschine:
  id = None

  def __init__(self, id):
    self.id = id

  def __str__(self):
    return f"Maschine({self.id = })"

class Flowshop:
  name = None
  nMaschines = None
  nJobs = None
  jobList = []

  def __init__(self, name, anzahlMaschinen,
      anzahlJobsProAuftrag, jobs):
    self.name = name
    self.nMaschines = int(anzahlMaschinen)
    self.nJobs = int(anzahlJobsProAuftrag)
```

```
37      self.jobList = jobs
38
39    def __str__(self):
40      return f"Flowshop({self.name = }, {self.nMaschines = }, {
            self.nJobs = }, {self.jobList = })"
```

(c) Warum ich jetzt hier eine Klasse erstellen soll, weiß ich nicht, eine Funktion hätte es auch getan

```
1  class InputData:
2    flowshop = None
3    filename = None
4
5    def __init__(self, filename):
6      self.filename = filename
7      with open(filename) as file:
8        data = json.load(file)
9        liste = []
10       for job in data["Jobs"]:
11         j = Job(job["Id"], job["SetupTimes"], job["
               ProcessingTimes"], job["DueDate"], job["TardCosts"
               ])
12         liste.append(j)
13       self.flowshop = Flowshop(data["Name"], data["nMachines"
             ], data["nJobs"], liste)
14
15 input = InputData("InputFlowshopSIST.json")
```

(d) Diese Klasse ist noch unnötiger...

```
1  class OutputJob:
2    job = None
3
4    def __init__(self, job):
5      self.job = job
6
7    def display(self):
8      print(self.job)
9
10 OutputJob(input.flowshop.jobList[0]).display()
```