Klausur Einführung in die Programmierung I

Bjoern Stuetz (bstuetz@lehre.dhbw-stuttgart.de)

2022-12-14

Klausur

Übersicht

- Klausurdauer: **120 Minuten**
- Mögliche Gesamtpunktzahl: **120 Punkte**
- **Keine** Hilfsmittel neben der Klausur (siehe *Informationen zur Verwendung von Python*)
- Verwenden Sie die ausgeteilten **Aufgabenblätter**, lösen Sie bitte die Aufgaben innerhalb der vorgesehen Bereiche.
- Sollte der Platz nicht ausreichen, oder sollten Sie Aufgaben auf der Rückseite der Klausur oder auf Extrablättern lösen, vermerken Sie dies bitte auf dem Aufgabenblatt bei der entsprechenden Aufgabe.

Aufgabenteil	Aufgabe	Erreichbare Punkte	Erreichte Punkte	Kommentar
A	1	5	[ ]	
A	2	5	[ ]	
B	1	10	[ ]	
B	2	20	[ ]	
B	3	10	[ ]	
C	1	40	[ ]	
C	2	30	[ ]	
–	–	**Summe**	**Erreicht**	–
–	–	**120**	**[ ]**	–

Tabelle 1: Aufgabenteile und Punkte

Falls Ihnen nicht immer sofort die richtige Lösung einfällt, beschreiben Sie bitte Ihren Lösungsansatz kurz und knapp mit einer Schema-Zeichnung, in Pseudocode oder direkt in Worten.

Viel Erfolg!

Informationen zur 'Programmierung auf Papier'

Bitte nehmen Sie sich kurz Zeit für diesen Abschnitt. Dieser wurde auch bereits in der Vorlesung vorgestellt. Es sind neben diesen Hinweisen keine Hilfsmittel erlaubt. Programmieren Sie bitte in dieser Klausur konzeptionell (d.h. soweit wie möglich vollständig und sinnvoll, dabei zielorientiert vereinfacht). Vermeiden Sie unnötige Lösungsteile, nach denen nicht ausdrücklich gefragt wurde. Die Syntax sollte größtenteils korrekt sein; es ist allerdings wichtiger Ihre Idee auszudrücken, als vollständig übersetzbaren (also *compilierbaren*) Code zu schreiben.

Vereinfachte Darstellung

Folgende Dinge können vereinfacht dargestellt werden (außer es wird ausdrücklich in der Fragestellung auf die reguläre Darstellung verwiesen):

- Ausgabe von Text muss nicht formatiert werden (z.B. `print(loesung, 'km')` statt `print('loesung =', loesung, 'km')`).
- Input von Text wird nie über die Konsole verwendet, Input-Variablen werden vorab zugewiesen (z.B. `eingabe = "Hallo"`).
- Es wird angenommen, dass die `__str__()` *magic method* immer mit allen Attributen vorhanden ist, auch wenn diese nicht von Ihnen ausprogrammiert wurde (z.B. nicht `print(f'Person={self.vorname}, ...)`).
- Es wird angenommen, dass Module vorhanden und automatisch importiert sind, auch wenn dies nicht von Ihnen ausprogrammiert wurde (z.B. nicht `from math import pi`).
- Es wird angenommen, dass der Standard-Konstruktor, also die `__init__()` Methode, immer vorhanden ist, auch wenn diese nicht von Ihnen ausprogrammiert wurden.
- Das Konzept *type hinting* muss nicht verwendet werden (z.B. nicht `def f(x: int) -> int:`).

Diese Punkte müssen entsprechend **nicht** ausprogrammiert bzw. voll ausgeschrieben werden. Dies soll Ihnen die Möglichkeit geben, sich auf die wesentlichen Teile zu konzentrieren und die Klausur in angemessener Zeit zu bearbeiten.

Programmierstil

Es gelten folgende Hinweise zum Programmierstil: Beachten Sie die gängigen Regeln und Konventionen der Programmiersprache. Programmieren Sie bitte auf **deutsch** oder **englisch** (*engl.*); falls Ihnen nicht das richtige Wort in englisch einfällt, nutzen Sie einfach ein **deutsch**es Wort. Bitte denken Sie an die notwendigen Konventionen für Python (z.B. Variablen, Klassen, Dateinamen). Bitte rücken Sie den Programmcode so weit wie möglich ein. Bitte verwenden Sie sprechende und selbsterklärende Identifier (z.B. nicht `i` für Sequenzen).

Bitte schreiben Sie immer leserlich und verständlich.

Informationen zur Verwendung von Python

Folgende Informationen sind für die Klausur relevant, diese wurde auch bereits in der Vorlesung vorgestellt:

- Zuweisungen: Zuweisung mit `=`, z.B. `a = 1`, Vergleich mit `==`, Inkrement/Dekrement mit `+=`, `-=`
- Rechnen: Addition `+`, Subtraktion `-`, Multiplikation `*`, Division `/`, Ganzahldivision `//`, Potenzierung `**`, Restwert (Modulo) `%`
- Datentypen: String: 'String' "String" `str()`, Integer: 5 `int()`, Float: (Gleitkomma): 4.3 `float()`, Boolean `True` / `False` `bool()`
- Datenstrukturen: Listen `[]`, Tupel `()` und Dictionaries `{ }`
- String-Operationen: `*`, `'String' * 2`, `+`, `in`, `'S' in 'String'`
- String Auswahl: `'String'[3]`, `'String'[1:2]`
- String Methoden: hochstellen, `upper()`, tiefstellen, `lower()`, Zeichen zählen `count('c')`, Zeichen ersetzen `replace('a', 'b')`, Leerzeichen (*whitespace*) entfernen `strip()`
- Listen: `list = ['a', 'b', c, d]`
- Listen Auswahl, Zerteilen, Kopieren: Index 1, `list[1]`, Index n-3, `list[-3]`, Index 1 und 2 `list[1:3]`, nach Index 1, `list[1:]`, vor Index 3, `list[:3]`, Kopie, `list2 = list1[:]`
- Listen von Listen: `list = [[a, b], [c, d]]`, `list[1][0]`
- Listenoperationen: `+`, `*`, `>`
- Listenmethoden: Index `index()`, Anzahl `count()`, Anhängen `append(a)`, Einfügen an Position `insert(0, i)`, Entfernen `remove(r)`, `del(list[0:1])`, `pop(-1)`, Erweitern `extend(list2)`, Umkehren `reverse()`, Sortieren `sort()`
- Import: `import math`, `import math as m`, `from math import pi`
- Fallunterscheidung: `if`, `elif`, `else`
- Schleifen: `for`, `while`
- Funktionen: `def function1(arg1, arg2 = 'default')`, `return`, `function1('1')`
- Klassen: `class Class1(object)`, `def __init__(self)`, `def function1(self, arg1, arg2 = 'default')` `return`, `class1 = Class1()`, `class1.function1('1')`
- Ausnahmen: `try`, `except`, `else`, `finally`
- Keine Operation: `pass`
- Datentyp ermitteln `type()`, prüfen: `isinstance()`
- Ausgabe auf der Kommandozeile: `print()`, Eingabe: `input()`
- Type Hinting: `def function1(arg1: int, arg2: str = 'default') -> int:`

Aufgabenteil A - Grundlagen

Aufgabe A1 - Textausgabe

Aufgabenteil	Aufgabe	Erreichbare Punkte	Erreichte Punkte
A	1	05	[ ]

Frage:

Wie erstellen Sie ein Python-Programm das in der Datei `hello.py`, das

- den String `Klausur am`
- und die Zahlen `14`, `12` und `2022` (alle `integer`) durch Punkte (`.` als `string`) getrennt,

auf der Kommandozeile ausgibt?

Lösung:

- (A1.1) Programmieren Sie bitte das Python-Programm `hello.py` (*engl.* für Hallo):

```
# hello.py



# ausgabe mit print()


```

- (A1.2) Geben Sie das Ergebnis der Ausgabe für die gegebenen Werte auf der Kommandozeile mit `print()` an:

```
$ python3 hello.py
```

- (A1.3) Erläutern Sie bitte **kurz** den Unterschied zwischen `print('text' + 'text')` und `print('text', 'text')` in Python ?

Aufgabe A2 - Datentypen, Typisierung und *Casts*

Aufgabenteil	Aufgabe	Erreichbare Punkte	Erreichte Punkte
A	2	05	[ ]

Frage:

Wie erstellen Sie ein Python-Programm in der Datei `calculate.py`,

- das die Ganzzahl `1` in eine Gleitzahl (`float()`) umwandelt,
- die Gleitzahl `1.22` addiert,
- und dann wieder zu einer Ganzzahl (`int()`)

ohne Rundungsregeln oder Rechenregeln als *cast* (`int()`, `float()`, etc.) umwandelt und das Ergebnis auf der Kommandozeile ausgibt?

Lösung:

- (A2.1) Programmieren Sie bitte das Python-Programm `calculate.py` (*engl.* für Berechnen):

```
# calculate.py

# eingabe
input = 1

# int zu float



# float addieren



# float zu int



# ausgabe mit print()

```

- (A2.2) Geben Sie das Ergebnis der Ausgabe für die gegebenen Werte auf der Kommandozeile mit `print()` an:

```
$ python3 calculate.py
```

- (A2.3) Erläutern Sie bitte **kurz** den Unterschied zwischen den Datentypen `int` und `float` in Python :

Aufgabenteil B - Weiterführende Themen

Aufgabe B1 - Verzweigungen

Aufgabenteil	Aufgabe	Erreichbare Punkte	Erreichte Punkte
B	1	10	[ ]

Frage:

Sie haben Hunger und wollen sich etwas zu essen kaufen. Sie haben `10` Euro zur Verfügung und möchten wissen, was Sie sich leisten können. Dazu erstellen Sie sich ein Programm.

Wie erstellen Sie ein Python-Programm in der Datei `hungry.py`,

- das bei `hungry = False` und `money > 0` die Ausgabe `'nicht hungrig'` und `'kann kaufen'`,
- das bei `hungry = True` und `money = 0` die Ausgabe `'hungrig'` und `'kann nichts kaufen.'`,
- das bei `hungry = True` und `money < 10` die Ausgabe `'hungrig'` und `'kann Döner kaufen.'`,
- und das bei `hungry = True` und `money >= 10` die Ausgabe `hungrig` und `'kann Ramen-Suppe kaufen.'`,

auf der Kommandozeile ausgibt?

Lösung:

- (B1.1) Programmieren Sie bitte das Python-Programm `hungry.py` (*engl.* für Hungrigsein):

```
# hungry.py

# eingabe
hungry = True
money = 10

# verzweigungen
```

```
19
20
21
22
23
24
25
26 # ausgabe mit print()
27
28
```

- (B1.2) Geben Sie das Ergebnis der Ausgabe für die gegebenen Werte auf der Kommandozeile mit `print()` an:

```
$ python3 hungry.py
```

- (B1.3) Erläutern Sie bitte **kurz** den Sinn und Zweck von Verzweigungen:

- (B1.4) Erläutern Sie bitte **kurz**, was der Unterschied zwischen `==` und `=` ist:

- (B1.5) Erläutern Sie bitte **kurz** was der Datentype `bool` *ean* in Python ist, und wofür wir diesen benötigen:

Aufgabe B2 - Schleifen und Verzweigungen

Aufgabenteil	Aufgabe	Erreichbare Punkte	Erreichte Punkte
B	2	20	[ ]

Frage:

Sie möchten ein Würfelspiel programmieren. Sie würfeln mit einem Würfel und wollen wissen, wie oft Sie eine `6` würfeln müssen, bis Sie eine `1` würfeln.

Wie erstellen Sie ein Python-Programm in der Datei `roll.py`,

- das in einer Schleife die Zufallszahl zwischen `1` und `6` generiert,
- die Anzahl der Würfe zählt,
- dabei die Anzahle der gewürfelten `6`er zählt,
- und die Schleife beendet, wenn die Zufallszahl `1` gewürfelt wird,

und das Ergebnis auf der Kommandozeile ausgibt?

> Hinweis: Nutzen Sie `random.randint(1,6)` um eine Zufallszahl zwischen 1 und 6 zu erhalten. Das Modul `random` ist bereits importiert.

Lösung:

- (B2.1) Programmieren Sie bitte das Python-Programm `roll.py` (*engl.* für Würfelrollen):

```
import random
# roll.py

# zaehler durchläufe
count_rolls = 0
# zaehler sechsen
count_sixes = 0

# schleife
```

```
19
20
21
22
23
24
25
26
27
28 # ausgabe mit print()
29
30
```

- (B2.2) Geben Sie das Ergebnis der Ausgabe für die gegebenen Werte auf der Kommandozeile mit `print()` an, unter der Annahme, dass die Zufallszahlen
  - `1`,
  - `6`,
  - `6`,
  - `6`,
  - und `1` in genau dieser Reihenfolge gewürfelt werden:

```
$ python3 roll.py
```

- (B2.3) Erläutern Sie bitte **kurz** den Sinn und Zweck von Schleifen:

- (B2.4) Erläutern Sie bitte **kurz**, was der Unterschied zwischen `for` und `while` ist:

- (B2.5) Erläutern Sie bitte **kurz** für was `break` und `continue` in Python verwendet werden:

Aufgabe B3 - Funktionen und Verzweigungen

Aufgabenteil	Aufgabe	Erreichbare Punkte	Erreichte Punkte
B	3	10	[ ]

Frage:

Sie wissen nicht, welches Geschenk Sie Ihrem Freund oder Freundin schenken sollen. Sie haben eine Liste mit `3` Geschenkideen, und möchten nun eine Funktion schreiben, die Ihnen eine zufällige Geschenkidee ausgibt.

Wie erstellen Sie ein Python-Programm in der Datei `present.py` (*engl.* für Geschenk),

- das in der Funktion `present()` eine zufällige Geschenkidee ausgibt,
- die Funktion `present()` aufruft,
  - dort eine zufällige Zahl zwischen `1` und `3` generiert,
  - je nach Zufallszahl eine der drei (`1` - `3`) Geschenkideen als Rückgabewert mit `return` als `string` zurückgibt,
  - denken Sie sich dabei die Geschenkideen **selbst** aus,

und das Ergebnis auf der Kommandozeile ausgibt?

> Hinweis: Nutzen Sie `random.randint(1,3)` um eine Zufallszahl zwischen 1 und 3 zu erhalten. Das Modul `random` ist bereits importiert.

Lösung:

- (B3.1) Programmieren Sie bitte das Python-Programm `present.py` :

```
1  import random
2  # present.py
3
4  # funktion
5
6
7
8
9
10
11
12
13
14
15
```

```
16
17
18
19 # funktionsaufruf
20
21
22
23
24 # ausgabe mit print()
25
26
```

- (B3.2) Geben Sie das Ergebnis der Ausgabe für die gegebenen Werte auf der Kommandozeile mit `print()` an, unter der Annahme, dass die Zufallszahl `2` generiert wird:

```
$ python3 present.py
```

- (B3.3) Erläutern Sie bitte **kurz** den Sinn und Zweck von Funktionen:

- (B3.4) Erläutern Sie bitte **kurz**, für was `import` und `return` in Python verwendet werden:

Aufgabenteil C - Fallbeispiele mit Klassen und Objekten

Aufgabe C1 - Fallbeispiel 1: Listen von Objekten, Verbinden von Objekten

Aufgabenteil	Aufgabe	Erreichbare Punkte	Erreichte Punkte
C	1	40	[ ]

Frage (1/2):

Sie möchten für Ihren Minigolfclub ein Programm schreiben, das die Mitglieder verwaltet. Jedes Mitglied hat einen Namen, ein Alter und eine Mitgliedsnummer. Ebenso hat jedes Mitglied eine Liste von Spielpartner:innen (`partners`), mit denen er oder sie schon gespielt hat. Sie möchten nun eine Klasse `Member` erstellen, die die Daten eines Mitglieds verwaltet.

Wie erstellen Sie die Python-Klasse `Member` (*engl.* für Mitglied) in der Datei `member.py` ,

- wenn die Klasse `Member` die Attribute
  - `name` als `string` ,
  - `age` als `float` (*engl.* für Alter),
  - `memberId` als `int`
  - und `partners` als `list` e von `Member` -Objekten hat,
- wenn die Attribute in der `__init__()` -Funktion initialisiert werden,
- wenn die Klasse `Member` die Methoden
  - `info()` ohne Parameter,
    - die die Daten des Mitglieds **einfach** als `string` zurückgibt,
  - `addPartner()` mit Parameter vom Typ `Member` ,
    - die einen neuen Partner der `partners` -Liste hinzufügt,
  - `getPartners()` ohne Parameter,
    - die die Liste der Partner zurückgibt, mit dem Rückgabewert vom Typ `list` e von `Member` -Objekten hat,

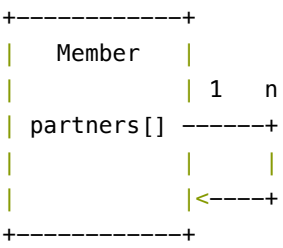
die also die Daten eines Mitglieds verwaltet?

Klassendiagramm:

```
+-----------------------------------------------+
| Member                                        |
+-----------------------------------------------+
| name (Typ: string)                            |
| age (Typ: float)                              |
| memberId (Typ: int)                           |
| partners (Typ: Liste von Member)              |
```

```
+-----------------------------------------------+
| __init__(self, name, age, memberId, partners) |
| info(self)                                    |
| addPartner(self, member Typ: Member)          |
| getPartners(self): Rückgabe Liste von Member  |
+-----------------------------------------------+
```

Übersicht:



Lösung (1/2):

- (C1.1) Programmieren Sie bitte die Python-Klasse `Member` in `member.py`:

```python
# member.py

# klasse
```

- (C1.2) Erläutern Sie bitte **kurz** den Sinn und Zweck von Klassen:

Frage (2/2):

Erweitern Sie nun die Klasse `Member` aus der Frage (1/2) um eine Methode `getPartnersByAgeGroup()` für die Ausgabe nach Alter gefilterten Spielpartner eines Mitglieds.

Wie erweitern Sie die Python-Klasse `Member` (*engl.* für Mitglied) in der Datei `member.py`,

- wenn die Klasse `Member` die **zusätzlichen** Methode
  - `getPartnersByAgeGroup()` mit Parameter `minimumAge` vom Typ `float`,
    - die die **nach Alter** gefilterte Liste der Partner zurückgibt,
    - mit dem Rückgabewert vom Typ `list`e von `Member`-Objekten hat,
      - wenn für die Methode die Regel `age >= minimumAge` gilt,

die also die Daten eines Mitglieds verwaltet?

> Hinweis: Programmieren Sie **nur** die zusätzliche Methode.

Lösung (2/2):

- (C1.3) Erweitern Sie bitte die Python-Klasse `Member` in `member.py`:

```
# member.py

# erweiterung klasse

class Member:

# ... bisheriger code aus C1.1 ...

# erweiterung methode

    def
```

Wie Testen Sie die Methode `getPartnersByAgeGroup()` in der Datei `test.py`,

- wenn Sie drei Objekte der Klasse `Member` anlegen,
  - `member1` mit den Attributen `name` = `"Maxine"`, `age` = `20.0`, `memberId` = `1` und `partners` = `[]`,
  - `member2` mit den Attributen `name` = `"Max"`, `age` = `30.0`, `memberId` = `2` und `partners` = `[ member1 ]`,
  - `member3` mit den Attributen `name` = `"Moritz"`, `age` = `40.0`, `memberId` = `3` und `partners` = `[ member1, member2 ]`,
- danach `member2` und `member3` als Spielpartner:in von `member1` hinzufügen,
- und schließlich die Methode `getPartnersByAgeGroup()` mit dem Parameter `minimumAge` = `40.0` aufrufen,

und anschliessend das Attribut `memberId` über die Methode `info()` des Ergebnis-Objekts bzw. der Ergebnis-Objekte auf der Kommandozeile ausgeben?

- (C1.4) Programmieren Sie bitte das Python-Programm `test.py`:

  Hinweis: Das Modul `member` ist bereits importiert.

```
from member import Member
# test.py

# eingabe
```

```
minimumAge = 30.0

# anlage der objekte
# member1



# member2



# member3



# hinzufuegen der spielpartner:innen



# aufruf der methode getPartnersByAgeGroup()



# ausgabe der von info() des ergebnis-objekts oder der ergebmis-objekte
```

- (C1.5) Geben Sie das Ergebnis der Ausgabe für die angegebenen Testdaten auf der Kommandozeile mit `print()` an:

```
$ python3 test.py
```

Aufgabe C2 - Fallbeispiel 2: Verbinden von Objekten, Listen von Objekten

Aufgabenteil	Aufgabe	Erreichbare Punkte	Erreichte Punkte
C	2	30	[ ]

Frage (1/3):

Sie planen eine Software für die Lagerverwaltung von Artikeln einer Schreibwaren-Firma. Jeder Artikel hat einen Namen, eine Artikelnummer, eine Menge und einen Preis.

Wie erstellen Sie die Python-Klasse `Item` (*engl.* für Artikel) in der Datei `item.py`,

- wenn die Klasse `Item` die Attribute
  - `name` als `string`,
  - `itemId` als `int`,
  - `amount` als `int` (*engl.* für Menge),
  - `price` als `float` (*engl.* für Preis),
- wenn die Attribute in der `__init__()`-Funktion initialisiert werden,
- wenn die Klasse `Item` die Methoden
  - `info()` ohne Parameter,
    - die die Daten des Artikel **einfach** als `string` zurückgibt

die also die Daten eines Artikels verwaltet?

Klassendiagramme:

```
+-----------------------------------------------+
| Item                                          |
+-----------------------------------------------+
| name (Typ: string)                            |
| itemId (Typ: int)                             |
| amount (Typ: int)                             |
| price (Typ: float)                            |
+-----------------------------------------------+
| __init__(self, name, itemId, amount, price)   |
| info(self)                                    |
+-----------------------------------------------+
```

Lösung (1/3):

- (C2.1) Programmieren Sie bitte die Python-Klasse `Item` in `item.py`:

```
# item.py

```

```
# klasse
```

- (C2.2) Erläutern Sie bitte **kurz** was Klassen, und was Objekte sind:

Frage (2/3):

Die Artikel sind in Artikelgruppen eingeteilt. Eine Artikelgruppe hat einen Namen und eine Artikelgruppennummer.

Wie erstellen Sie die Python-Klasse `ItemGroup` (*engl.* für Artikelgruppe) in der Datei `itemgroup.py`,

- wenn die Klasse `ItemGroup` die Attribute
  - `name` als `string`,
  - `itemGroupId` als `int`,
- wenn die Attribute in der `__init__()`-Funktion initialisiert werden,
- wenn die Klasse `ItemGroup` die Methoden

  - `info()` ohne Parameter,
    * die die Daten der Artikelgruppe **ohne** die Liste der Artikel **einfach** als `string` zurückgibt

die also die Daten einer Artikelgrupp verwaltet?

Klassendiagramme:

```
+-----------------------------------------------+
| ItemGroup                                     |
+-----------------------------------------------+
| name (Typ: string)                            |
| itemGroupId (Typ: int)                        |
| items (Typ: Liste von Item)                   |
+-----------------------------------------------+
| __init__(self, name, itemGroupId, items)      |
| info(self)                                    |
+-----------------------------------------------+
```

Übersicht:

```
+------------+       +------------+
| ItemGroup  |       |            |
|            | 1   n |            |
| items[]    -------->+    Item    |
|            |       |            |
|            |       |            |
+------------+       +-----+------+
```

Lösung (2/3):

- (C2.3) Programmieren Sie bitte die Python-Klasse `ItemGroup` in `itemgroup.py`:

  Hinweis: Das Modul `item` ist bereits importiert.

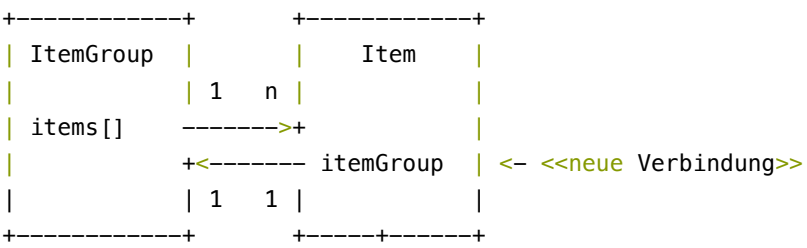
```
from item import Item
# itemgroup.py

# klasse
```

Frage (3/3):

Jeder Artikel gehört zu einer Artikelgruppe. Wie erweitern Sie die Python-Klasse `Item` in der Datei `item.py`, die dann auch die Verbindung zu einer Artikelgruppe beinhaltet?

Übersicht:



> Hinweis: Programmieren Sie **nur** die zusätzliche Verbindung **im Konstruktor bzw. der init-Methode**. Das Modul `itemgroup` ist bereits importiert.

Lösung (3/3):

- (C2.4) Erweitern Sie bitte die Python-Klasse `Item` in `item.py`:

```
from itemgroup import ItemGroup
# item.py

# erweiterung klasse

class Item:
```

```
7
8  # ... bisheriger code aus C2.1 ...
9
10     # erweiterung konstruktor
11
12     def __init__(self, name, itemId, amount, price,
13
14
15         self.name = name
16
17         self.itemId = itemId
18
19         self.amount = amount
20
21         self.price = price
22
23
24
25
26
27
28
29
30
```

- (C2.5) Skizzieren Sie bitte **kurz** in Stichworten und Pseudocode, wie Sie den Durchschnittspreis einer Artikelgruppe berechnen würden:

  Hinweis: Sie können Python-Code verwenden, müssen es aber nicht.

- (C2.6) Erläutern Sie bitte **kurz** was ein Modul in Python ist:

Ende