Hochschule
Albstadt-Sigmaringen
Albstadt-Sigmaringen University

Studiengänge ITS,TI,WIN **Programmieren 1**
Vorlesung: **Prof.Dr.Derk Rembold**
Praktikum: Ralf Mispelhorn, Dominik Höpfl

Aufgabenblatt 5

Abgabe: Die Lösungen sollten zeitnah als IPYNB-Datei fertiggestellt werden. Der Code muss ausreichend kommentiert sein und die Variablen müssen sinnvoll benannt werden. Sie müssen die Aufgabe selbst programmiert haben. Sie können Fragen in Form von Kommentaren im Code stellen, falls etwas nicht funktioniert hat. Die Antwort erfolgt dann im Praktikum mündlich. Sie dürfen nie mehr als drei Aufgabenblätter im Rückstand sein.

Hilfsmittel: Kein Copy-Paste aus dem Internet, alles muss selbstständig programmiert sein. Sie dürfen die IPython-Notebook-Skripte aus der Vorlesung (liegen nach der jeweiligen Vorlesung auf Ilias) und die Python-Einführung auf Ilias verwenden. Außerdem ist die Hilfe-Funktion `help(...)` und die Methode `dir(...)` zur Auflistung der verfügbaren Funktionen zu empfehlen.

Anwesenheit: Grundsätzlich herrscht Anwesenheitspflicht. Ein Attest ist notwendig, wenn jemand nicht kommen kann. Bei unentschuldigtem Fehlen ist das Praktikum nicht bestanden. Von der Teilnahme an der Klausur wird in diesem Fall dringend abgeraten.

Aufgabe 5.0

Implementieren Sie eine Funktion `saveperson(n, v, a, g)`, bei der Sie Nachname, Vorname, Alter und Größe (in Zentimeter) angeben können. Alle Eingaben sollen nach Ihren Typen mit Hilfe der `assert()`-Funktion überprüft werden. Also der Nachname und Vorname müssen ein String sein, Alter und Größe ein Integer.

Nach der Eingabe wird ein *json*-File (`namensliste.json`) eingeladen und die Information im File in eine Liste von *Dict*-Variablen umgewandelt (anfangs ist das File leer). Die eingegebene Information wird hinten an die Liste als *Dict*-Variable angehängt und auf die Festplatte zurückgespeichert.

(a) Implementieren Sie eine weitere Funktion `findperson(n)` mit Name als Eingabe, die ein *json*-File (`namensliste.json`) einliest, und die Namen im *json*-File mit der Eingabe vergleicht. Falls ein Name im File dem Namen der Eingabe entspricht, dann soll die gesamte Information der Person ausgegeben werden. Überprüfen Sie mit der `assert()`-Funktion, ob die Eingabe tatsächlich ein String ist.

(b) Implementieren Sie eine weitere Funktion `removeperson(n)` mit Name als Eingabe, die ein *json*-File (`namensliste.json`) einliest und die Information im File in eine Liste von *Dict*-Variablen umgewandelt. Die Namen in der Liste soll mit der Eingabe verglichen werden. Falls ein Name im File dem Namen der Eingabe entspricht, dann soll die *Dict*-Variable aus der Liste entfernt werden, und die restlichen Daten in der Liste in das *json*-File zurück auf die Festplatte gespeichert werden.

Aufgabe 5.1

Schreiben Sie eine Funktion `findFile(d, n)`, die als Eingangsparameter ein Verzeichnis `d` und ein Filename `n` hat. Überprüfen Sie bei Aufruf, ob das Verzeichnis tatsächlich existiert. Falls nicht, dann soll eine *Exception* geworfen werden. Die Funktion soll im Verzeichnis `d` alle Files mit dem Namen `n` suchen und mit `print()` ausgeben. Es besteht auch die Möglichkeit, `n` mit einem *regulären Ausdruck* als String anzugeben. Mit Hilfe eines regulären Ausdrucks soll überprüft werden, ob die Eingabe einem korrekten Filenamen entspricht. Die Funktion `os.listdir()` der `os.path` Bibliothek liefert dafür eine Liste von Files. Alle Elemente der Liste sollen mit dem regulären Ausdruck überprüft werden. Entspricht ein File dem Suchkriterium, dann soll die Funktion den Filenamen ausgeben.

Folgende Files sollen Sie auf Ihrer Festplatte im Verzeichnis d finden. Stellen Sie sicher, dass sich diese Files aus Testzwecken in Ihrem Verzeichnis befinden:

(a) Ein File namens `test.txt`.

(b) Alle Files, deren Filename mit einem Punkt getrennt sind.

(c) Alle Files, die mit einem Punkt anfangen.

(d) Alle Files mit der Endung `*.txt`, `*.csv` oder `*.json`

(e) Alle Files, die mit *Aufgabe* anfangen und mit *pdf* aufhören.

(f) Alle Files, die keine Zahlen im Namen enthalten.

Aufgabe 5.2

Implementieren Sie eine (eigene) Klasse `MyCompl2`, die komplexe Zahlen repräsentieren soll mit passenden Attributen und Methoden – bitte ohne Zuhilfename der bestehenden `complex`-Klasse.

(a) Implementieren Sie eine sinnvolle Konstruktor-Methode.

```
>>> c1 = MyCompl2(1.0,2.0)
```

(b) Implementieren Sie die `__str__`-Methode, die eine String-Repräsentation eines `MyCompl2`-Objekts wie folgt zurückliefern soll:

```
>>> c1 = MyCompl2(1.0,2.0)
>>> print(c1)
1.0 + 2.0i
```

(c) Implementieren Sie eine Methode, mit dem zwei komplexe Zahlen addiert werden können. Es soll die interne Methode `__add__` dazu verwendet werden.

```
>>> c1 = MyCompl2(1,2)
>>> c2 = MyCompl2(4,4)
>>> c1,c2
(1 + 2i, 4 + 4i)
>>> c1 + c2
5 + 6i
```

(d) Implementieren Sie eine Methode, mit dem zwei komplexe Zahlen multipliziert werden können. Es soll die interne Methode `__mul__` dazu verwendet werden.

```
>>> c1 * c2
-4 + 12i
```