Danny Rehl, Programmieren I - Python

Übungsblatt 05

Abgabefrist: Montag, den 18.11.2024, um 23:59 Uhr

Sie finden für dieses Übungsblatt eine Vorlage für Ihre Abgabe zum Herunterladen im Moodle.

Für die Bewertung Ihrer Abgabe zählen die von Ihnen vervollständigten Funktionen im oberen Teil der Vorlage. Schreiben Sie hier nur Ihre Lösungen hin, keine weiteren Befehle außerhalb der Funktionen.

Im unteren Teil finden Sie einen `if __name__ == "__main__"`-Abschnitt, welcher bereits einige Beispielaufrufe zu den Funktionen beinhaltet. Hier können Sie die Befehle beliebig ergänzen oder ersetzen, um Ihre Funktionen zu testen.

Hinweis: Trotz aller Tests muss die hochgeladene `.py`-Datei natürlich immer noch ausführbar bleiben. Beispielsweise wird ein `SyntaxError` im `if __name__ == "__main__"`-Teil in jedem Fall zur Pytest-Bewertung von 0 % führen.

Aufgabe 1 (1,5 Punkte)

Sie finden in der Vorlage die Funktion `add_snake(snake, skill, template)`, welche zwei String-Parameter `snake` und `skill` erhält und diese als Tuple in einer Liste zurückgibt. Diese Liste wird vorher um die bereits vorhandenen Einträge in `template` ergänzt. Dabei soll die per `template` übergebene Liste nicht geändert werden.

Beispiel:

```
>>> SNAKES = [("Cobra", "Venom"), ("Boa", "Constriction")]
>>> my_snakes = add_snake("Rattlesnake", "Venom", SNAKES)
>>> print(my_snakes)
[('Cobra', 'Venom'), ('Boa', 'Constriction'), ('Rattlesnake', 'Venom')]
>>> print(SNAKES)
[('Cobra', 'Venom'), ('Boa', 'Constriction')]
```

Aufgabe 2 (2,5 Punkte)

Vervollständigen Sie eine Funktion `is_perfect_number`, welche eine Ganzzahl `number` als Argument erhält. Der Rückgabewert vom Typ `bool` gibt an, ob die übergebene natürliche, positive Zahl eine vollkommene Zahl ist.

Eine natürliche Zahl wird perfekte oder vollkommene Zahl genannt, wenn sie gleich der Summe aller ihrer Teiler außer sich selbst ist. Ein Beispiel ist 28, denn 28 ist teilbar ohne Rest mit 1, 2, 4, 7, 14 und 28. Deren Summe (außer der Zahl selbst) ist $1 + 2 + 4 + 7 + 14 = 28$.

Tipp: Machen Sie es sich erst klar wie Sie hierbei generell vorgehen müssen, bevor Sie anfangen, Code zu schreiben. Das heißt, überlegen Sie sich zunächst einen Algorithmus (zum Beispiel auf Papier oder als Pseudo-Code) und implementieren Sie diesen im zweiten Schritt.

Aufgabe 3 (1,5 Punkte)

Es geht bei dieser Aufgabe um eine Funktion `append_square`, welche das Quadrat einer Zahl an eine Liste anhängt.

Der Funktionskopf sieht zunächst so aus:

```
def append_square(number, numbers)
```

Und die Funktion lässt sich beispielsweise so verwenden:

```
>>> append_square(2, numbers=[1, 16])
[1, 16, 4]
```

Wir haben in diesem Beispiel das Quadrat von `2` (also `4`) an eine bereits bestehende Liste `[1, 16]` angehängt und wie erwartet `[1, 16, 4]` erhalten.

Für `3` wollen wir mit einer leeren Liste starten:

```
>>> append_square(3, numbers=[])
[9]
```

Das funktioniert zwar, ist aber etwas lästig, denn irgendwie könnten wir uns die Angabe `number=[]` sparen, wenn wir das als Standard für die Funktion festlegen würden. Wir ändern also den Funktionskopf zu der Form, wie Sie die Funktion auch in der Vorlage finden werden:

```
def append_square(number, numbers=[])
```

Es gibt immer noch zwei Parameter, wobei der zweite Parameter jetzt optional ist, denn für das `numbers` Argument ist ein Standardwert ("default value") von `[]` angegeben.

Wenn wir jetzt also mit `3` eine neue Liste anfangen möchten, dann können wir den zweiten Parameter einfach weglassen:

```
>>> append_square(3)
[9]
```

Soweit scheint alles zu funktioniert. Wenn wir jetzt allerdings eine neue Liste mit beispielsweise `5` starten wollen, erhalten wir folgendes Ergebnis:

```
>>> append_square(5)
[9, 25]
```

Irgendwie war die Liste nicht leer und es ist noch das Ergebnis vom vorherigen Funktionsaufruf `append_square(3)` gespeichert.

Wenn wir jedoch explizit eine leere Liste für den `numbers` Funktionsparameter übergeben, dann erhalten wir das gewünschte Ergebnis:

```
>>> append_square(5, numbers=[])
[25]
```

Warum funktioniert der zweite Funktionsparameter nicht wie erwartet? Wie lässen sich die beobachteten Rückgabewerte erklären? Sie haben diese Besonderheit von Funktionsparametern, deren Standardwert ein mutabler Datentyp ist, bereits in der Vorlesung kennengelernt.

Verbessern Sie die Funktion, sodass der Aufruf sowohl mit leeren, als auch mit bestehenden Listen funktioniert. Des Weiteren soll der zweite Funktionsparameter optional bleiben und bei Weglassen wie ein Aufruf mit leerer Liste interpretiert werden.

Nach dem Verbessern soll die Funktion also beispielsweise so funktionieren:

```
>>> append_square(7, numbers=[9, 4])
[9, 4, 49]
>>> append_square(1, numbers=[])
[1]
>>> append_square(2)
[4]
>>> append_square(3)
[9]
>>> NUMBERS = []
>>> append_square(4, NUMBERS)
[16]
>>> append_square(5, NUMBERS)
[16, 25]
```

Aufgabe 4 (1,5 Punkte)

Wir möchten mithilfe einer rekursiven Funktion die Fakultät einer Zahl berechnen. Allerdings bekommen wir von Python beim Aufruf der Funktion einen komischen Fehler und kein Ergebnis. Erwartet hätten wir zum Beispiel:

```
Fakultät von 3 ist: 6
Fakultät von 5 ist: 120
```

Welcher häufige Fehler ist hier unterlaufen? Verbessern Sie die Funktion `factorial` und dokumentieren Sie Ihre Begründung als Kommentar im Code.

Aufgabe 5 (1,5 Punkte)

Die Beträge einer Rechnung sind in Euro gespeichert, sollen allerdings sowohl in Euro als auch anschließend in Dollar ausgegeben werden. (Als Umrechnungskurs nehmen wir einfach mal `1.5` an.) Die passenden Dollar-Werte wurden erfolgreich berechnet, aber irgendetwas Seltsames ist mit den Euro-Werten passiert. Es soll eigentlich angezeigt werden:

```
Rechnung in Währung Euro:
{'Charles': [27, 12], 'Jacques': [31, 38]}
Rechnung in Währung Dollar:
{'Charles': [40.5, 18.0], 'Jacques': [46.5, 57.0]}
```

Welcher häufige Fehler ist hier unterlaufen? Verbessern Sie die Funktion `euros_to_dollars` und dokumentieren Sie Ihre Begründung als Kommentar im Code.

Hinweis: `enumerate` ist eine praktische built-in function von Python. Sie gibt Ihnen beim Iterieren über ein Objekt zusätzlich die derzeitige Iterationsanzahl als Zahl zurück. Diese entspricht dann praktischerweise der aktuellen Indexposition des Objekts, da hier mit 0 begonnen wird zu zählen. Die Rückgabe von `enumerate` ist dann ein Tupel mit genannter Zahl als erstes Element und das eigentliche Objekt als zweites Element.

Aufgabe 6 (1,5 Punkte)

Es soll eine zufällig Position auf dem Schachbrett ausgegeben werden. Für die Ausgabe erwarten wir zum Beispiel:

```
Horizontal: D
Vertikal: 2
```

Leider funktioniert anscheinend jedoch nur die Ausgabe der ersten Zeile. Die zweite Zeile macht Probleme.

Welcher häufige Fehler ist hier unterlaufen? Verbessern Sie die Funktion `print_random_chess_position` und dokumentieren Ihre Begründung als Kommentar im Code.
