Fingerübungen

Woche 05 – 8. Juni 2016

Sequentieller Datentyp `tuple([Iterable])`

Tupel: Unveränderliche Sequenzen; `len()`; Methoden `index()` u. `count()`

Die einzigen Methoden, die an Tupel-Objekten aufgerufen werden können, sind `index(Element)` und `count(Element)`. Die **`index`**-Methode gibt dabei die Position zurück, an dem ein bestimmtes Element vorzufinden ist, während die **`count`**-Methode zurückgibt, wie oft ein Element in dem Tupel enthalten ist.

Achtung: **Methoden != Funktionen**.

- *Methoden* werden mit *Punktnotation*, also in diesem Fall bspw. mit `(1, "2", 3.0).count(1)` aufgerufen. Die Schreibweise `o.m()` bedeutet, dass die Methode `m()` auf dem Objekt `o` aufgerufen wird. Es hängt immer vom *Typ* eines Objektes ab, welche Methoden „verfügbar“ sind. Auf Strings (*str*) hatten wir bereits die `replace`-Methode kennengelernt, die z. B. im Falle `"Hallo".replace('a', 'e')` `"Hello"` zurückgibt. Der String `"Hallo"` wird nicht verändert, da Strings (wie Tupel) unveränderlich sind.
- *Funktionen* sind bspw. `len()`, `str()`, `tuple()`, `help()`, `print()` und `input()`. Ihnen werden i. d. R. Parameter übergeben. Anders als Methoden werden sie unabhängig von einem Objekt aufgerufen (keine Punktnotation). Z. B. kann `len()` ein Tupel oder String übergeben werden und `len()` gibt Länge/Größe zurück. `len((1, 2, 3))` ist `3`, `len("Hallo")` ist `5`.

Sequentieller Datentyp `tuple([Iterable])`

Tupel: Unveränderliche Sequenzen; `len()`; Methoden `index()` u. `count()`

(1) Erste Tupel: Funktion `len()`, Methoden, Indizierung und Slicing

Erstellen Sie einige Tupel und experimentieren Sie mit den Methoden `index()` und `count()` sowie der Funktion `len()`. Üben Sie sich an Index-basiertem Zugriff auf einzelne Elemente (z. B. mittels `t[0]` oder `name[1]`) und „Teil-Tupel“ durch *Slicing*, also z. B. `(1, 2, 3, 4)[1:3:2]`

Wichtige Beispiele in Kürze (*Was erwarten Sie? Was passiert? Wieso?*):

a) `len((1, 2, 3))`
b) `len(("ab", "c"))`
c) `len((8, "Wort"))`
d) `len((17, ))`
e) `len(("Wort"))`
f) `len(("abc", ))`

g) `t = ("a", "b")`
Was ist `t[0]`*?*
h) `(1, 2, 3)[1]`
i) `(19, 7, 9)[-1]`
j) `(1, 2, 3, 4)[1:3]`
k) `(5, 9, 4)[:2]`

l) `(1, 1, 2).count(1)`
m) `(1, 1, 2).index(1)`
n) `(2, 3, 4).index(1)`
o) `range(3, 7)[2]`
p) `tuple(range(5, 9))`
q) `1 in (3, 2, 1, 5)`

Wie können Tupel mit nur einem Element erzeugt werden? Warum nur so?
Ist ein Element nicht vorhanden, liefert die Methode `index()` eine Fehlermeldung. *Wie könnte man die Fehlermeldung „umgehen“?*

Sequentieller Datentyp `tuple([Iterable])`

Tupel: Unveränderliche Sequenzen; Funktion `len()`

(2) Zufallsnamen

Erstellen Sie zwei Tupel (`vornamen` und `nachnamen`), die verschiedene Vor- bzw. Nachnamen enthalten (beliebig viele). Definieren Sie nun eine Funktion, die aus den beiden Tupeln jeweils ein Element zufällig auswählt, und diese beiden Elemente (Vor- und Nachname) zu einem Namen verknüpft. Der (gesamte) Name soll als ein (zusammenhängender) String (*str*) zurückgegeben werden.

Erinnerung: Sie können die Funktion `randint(a, b)` aus dem Modul `random` nutzen, um eine zufällige Zahl zwischen `a` und `b` zu erzeugen. Um auf Funktionen aus einem Modul zuzugreifen, müssen Sie dieses zunächst importieren (`import random`). Es folgt der Aufruf: `random.randint(…)`.

Schreiben Sie anschließend ein Programm, das eine bestimmte Anzahl von Zufallsnamen generiert (also die soeben definierte Funktion nutzt) und auf Konsole ausgibt. Die Anzahl an Zufallsnamen (Typ *int*) soll vom Benutzer eingelesen werden.

Sequentieller Datentyp `tuple([Iterable])`

Tupel: Unveränderliche Sequenzen; Methode `count()`

(3) [*Zusatzaufgabe*] **Vokale zählen**

Erweitern Sie Ihr Programm (nicht die Funktion!) aus Aufgabe (2) wie folgt: Es soll nun rechts neben dem (gesamten) Namen noch die Anzahl der Vokale ausgegeben werden!

Schreiben Sie dazu eine weitere Funktion, die als Parameter einen String (oder ein Tupel) übergeben bekommt, die Anzahl der darin enthaltenen Vokale (a, e, i, o, u) ermittelt und zurückgibt. Ihr Programm aus Aufgabe (2) soll nun – nachdem ein Zufallsname generiert wurde – diesen Zufallsnamen an die neue Funktion übergeben und das Ergebnis (also die Anzahl der darin enthaltenen Vokale) nach dem Zufallsnamen ausgeben.

Hinweis: Strings können ähnlich wie Tupel behandelt werden und besitzen auch die Methode `count()`!

Beispielausgabe (falls der Nutzer 3 eingibt):

```
Alois Edelmann (6)
Edmund Fischer (4)
Ursula Sattler (5)
```

Sequentieller Datentyp `list([Iterable])`

Listen: Veränderliche Sequenzen; Methoden auf Listen

(4) Erste Listen: Wichtige Methoden

Experimentieren Sie jetzt mit Listen (wie in Aufgabe (1)) und einigen weiteren Methoden auf Listen (*in-place → verändern das Objekt!*):

- **`append(x)`**: Fügt das Element `x` hinten an die Liste an.
- **`insert(i, x)`**: Fügt das Element `x` am Index `i` ein (andere aufgerückt).
- **`remove(x)`**: Entfernt das erste Vorkommen von Element `x`
- **`pop([i])`**: Entfernt das Element am Index `i` und gibt es zurück.

(5) Lottozahlen

Schreiben Sie eine Funktion (ohne Parameter), die eine Liste von Lottozahlen generiert und zurückgibt (→ `return`). Die Liste soll also 6 *verschiedene* Zahlen zwischen 1 und 49 enthalten! Hilfreich ist die Methode `append()`.

Als zusätzliche Bedingung sollen die Zahlen aufsteigend sortiert sein. Sie können hierfür die Methode **`sort()`** auf der Liste aufrufen, indem Sie z. B. für die Liste `myList = [7, 1, 3]` schreiben: `myList.sort()`

Anmerkung: Es können nur Listen sortiert werden, deren Elemente alle von demselben Typ sind!

Sequentieller Datentyp `list([Iterable])`

Listen: Veränderliche Sequenzen; Methoden auf Listen

(6) [*Zusatzaufgabe*] **Lotto *„6aus49“*** (Spiel)

Lassen Sie den Nutzer zunächst 6 Zahlen zwischen 1 und 49 raten, indem Sie diese nacheinander (per `input()`) abfragen. Fügen Sie die eingegebenen Zahlen dazu nacheinander einer Liste hinzu (→ `append`).
Achten Sie darauf, dass der Nutzer (1.) keine Zahlen doppelt eingibt und (2.) nur Zahlen zwischen 1 und 49 erlaubt sind. → Prüfen Sie die Eingaben und geben Sie bei einer ungültigen Eingabe eine entsprechende Meldung aus.

Nutzen Sie anschließend die Funktion aus Aufgabe (5), um eine Ziehung von Lottozahlen zu simulieren. Überprüfen Sie, wie viele Zahlen richtig geraten wurden und geben Sie das Ergebnis (auf der Konsole) aus.

Tipp: Nutzen Sie für diese Überprüfung den `in`-Operator.

Sequentieller Datentyp `list([Iterable])`

Listen, Tupel und Strings

(7) [*Zusatzaufgabe*] **Indices finden**

Die Methode **`index(`**`Element`**`)`** passt uns in zweierlei Hinsicht nicht. Einerseits liefert sie nur den ersten Index des jeweiligen Elements, andererseits endet sie in einer Fehlermeldung, falls das Element im Tupel bzw. in der Liste nicht existiert. Ihre Aufgabe ist es nun, eine Funktion zu schreiben, der als Parameter (1.) ein Tupel oder eine Liste (macht keinen Unterschied) und (2.) das gesuchte Element übergeben wird. Die Funktion soll dann eine Liste aller Indizes (!) zurückgeben, an denen das gesuchte Element steht.

Beispiele: Wird der Funktion das Tupel `(1, 2, 1, 3, 4, 1)` übergeben und das Element `1` gesucht, so soll die Funktion `[0, 2, 5]` zurückgeben. Wird nach dem Element `76` gesucht, so soll eine leere Liste `[]` zurückgegeben werden (→ keine Fehlermeldung!).

Hinweis: Sie können z. B. mit einer `for`-Schleife über alle Indizes der Liste bzw. des Tupels iterieren (d. h. `range` bis `len()`) und den jeweiligen Index an eine Rückgabe-Liste anhängen, falls das Element dort gefunden wurde.