יחידה 8

טאפל ומילון

שלושה פרקים, 24 דקות. tuple, השמה מרובה, תבנית מחרוזת, ו-dict.

מפת היחידה

18 חלקים ▲

פרק	דקה	מה קורה שם
8.1	0:33	טאפל ריק, וסוגריים עגולים
8.1	1:05	טאפל בלי סוגריים
8.1	**1:36**	**טאפל עם איבר אחד: הפסיק**
8.1	2:38	מה משותף לטאפל ולרשימה
8.1	**4:25**	**טאפל לא ניתן לשינוי**
8.1	5:26	למה בכלל טאפל
8.1	6:32	מתי טאפל ומתי רשימה
8.2	0:30	השמה לכמה משתנים בבת אחת
8.2	2:10	פונקציה שמחזירה כמה ערכים
8.2	**4:15**	**תבנית מחרוזת עם `%`**
8.2	6:17	`%d` , ולמה זה עדיף על `+`
8.3	0:32	מפתח וערך
8.3	1:02	`{}` ו- `dict()`
8.3	2:40	שליפה לפי מפתח
8.3	3:14	`KeyError`
8.3	**4:16**	**הוספה ועדכון**
8.3	6:05	`del` ו- `pop`
8.3	6:38	`keys` , `values` , `items`

8.1 · טאפל

טאפל (`tuple`) מחזיק כמה איברים, גם מטיפוסים שונים, בדיוק כמו רשימה. ההבדל: **אחרי שנוצר, אי אפשר לשנות אותו.**

```
>>> my_first_tuple = ()
>>> type(my_first_tuple)
<class 'tuple'>
>>> lunch = (3.5, 2, "sandwich")
>>> lunch = 3.5, 2, "sandwich"
>>> lunch
(3.5, 2, 'sandwich')
```

סוגריים עגולים, ופסיקים. ובלי סוגריים בכלל, פייתון אורזת את הערכים לטאפל לבד.

איבר אחד: הפסיק

```
>>> type((20))
<class 'int'>
>>> type((20,))
<class 'tuple'>
```

`(20)` הוא רק `20` בסוגריים, כמו בחשבון. **הפסיק הוא מה שהופך את זה לטאפל.**

מה עובד כמו ברשימה

טאפל הוא רצף, ולכן גישה, חיתוך, `len` , `in` ולולאת `for` עובדים:

```
>>> lunch[0]
3.5
>>> lunch[1:]
(2, 'sandwich')
>>> len(lunch)
3
>>> for item in lunch:
...     print(item)
...
3.5
2
sandwich
```

בסרטון רצות כאן עוד כמה פקודות על הטאפל, שמוצגות על המסך בלי הסבר בקול, ולכן הן לא בסיכום.

מה לא עובד

```
>>> salad = ("cucumber", "tomato", "tuna", "lemon")
>>> salad[2] = "spinach"
Traceback (most recent call last):
  ...
TypeError: 'tuple' object does not support item assignment
```

	רשימה	טאפל	מחרוזת
סוגריים	[ ]	( )	מרכאות
שינוי	**mutable**	immutable	immutable
הוספה ומחיקה	כן	לא	לא

אז למה טאפל

1. **בטוח.** מה שלא אמור להשתנות לא ישתנה בטעות
2. **חסכוני בזיכרון.** לרשימה מוקצה מקום נוסף כדי שתוכל לגדול. לטאפל בדיוק כמה שצריך

מתי טאפל

כששני תנאים מתקיימים: **מספר האיברים קבוע**, ו**למיקום יש משמעות**. הדוגמה הקלאסית: נקודה בגרף.
תמיד שני איברים, והראשון הוא x והשני y.

```
>>> point = (5, 2)
```

8.2 · שימושים בטאפל

השמה מרובה

```
>>> food, kind, calories = "bread", "whole wheat", 80
>>> food
'bread'
>>> calories
80
```

טאפל של שמות משמאל, טאפל של ערכים מימין, וכל שם מקבל את הערך שבמקום שלו. **וזה מה שקרה בהחלפת הערכים ביחידה 2:**

```
>>> my_basket, your_basket = "bread", "fruit"
>>> my_basket, your_basket = your_basket, my_basket
>>> my_basket
'fruit'
```

פונקציה שמחזירה כמה ערכים

ביחידה 5 ראינו `return a, b` . מה שחוזר הוא טאפל:

```
>>> def picnic_shopping(prices):
...     total = sum(prices)
...     return total, total <= 150
...
>>> result = picnic_shopping([12.5, 25, 60])
>>> type(result)
<class 'tuple'>
>>> money, is_budget_possible = picnic_shopping([12.5, 25, 60])
>>> money
97.5
>>> is_budget_possible
True
```

(`sum` מחברת את כל האיברים ברשימה.)

תבנית מחרוזת

שרשור עם `+` עובד, אבל ארוך ומבולגן, ונופל על מספרים:

```
>>> fruit = "grapes"
>>> cheese = "brie"
>>> print("In the basket: " + fruit + " and " + cheese)
In the basket: grapes and brie
>>> print("In the basket: %s and %s" % (fruit, cheese))
In the basket: grapes and brie
```

`%s`	מקום לערך (s מ-string)
`%d`	מקום למספר שלם
`% (a, b)`	אחרי המחרוזת: טאפל של הערכים, **לפי הסדר**

היתרון: הטקסט נפרד מהמשתנים, וקל לשנות כל אחד מהם. ואפשר לשבץ מספרים בלי הסבה:

```
>>> drivers = 1
>>> role = "designated driver"
>>> print("We need %d %s" % (drivers, role))
We need 1 designated driver
>>> print("We need " + drivers + " " + role)
Traceback (most recent call last):
  ...
TypeError: can only concatenate str (not "int") to str
```

אפשר גם לשים את הטאפל במשתנה מראש: `basket_content = (drivers, role)` , ואז `"..." %` `basket_content`.

זו השיטה הישנה. היא נפוצה מאוד, ותראו אותה בהרבה קוד. השיטה החדשה היא המתודה `format` , והסרטון מפנה לקישור עליה.

8.3 · מילון

ברשימה ניגשים לערך לפי **מיקום**, מספר שלא אומר כלום. ב**מילון** (`dict`) ניגשים לפי **מפתח** (key), שיש לו משמעות. כל איבר הוא **זוג**: מפתח וערך (value).

```
>>> empty = {}
>>> type(empty)
<class 'dict'>
>>> empty = dict()
>>> keyboard_shortcuts = {"copy": "Ctrl+C",
...                       "paste": "Ctrl+V",
...                       "undo": "Ctrl+Z"}
```

סוגריים מסולסלים, נקודתיים בין מפתח לערך, ופסיקים בין זוגות.

	סוגריים	גישה לפי
רשימה	[ ]	מיקום
טאפל	( )	מיקום
מילון	{ }	מפתח

שליפה

```
>>> keyboard_shortcuts["paste"]
'Ctrl+V'
>>> keyboard_shortcuts["cut"]
Traceback (most recent call last):
  ...
KeyError: 'cut'
```

- שולפים **לפי מפתח בלבד**. אין שליפה הפוכה, מערך למפתח. גם במילון של מילים מחפשים את המילה, ולא את ההסבר
- מפתח שלא קיים: `KeyError` . בודקים לפני עם `in` :

```
>>> "cut" in keyboard_shortcuts
False
```

הוספה ועדכון

מילון הוא **mutable**. אותה צורה מוסיפה ומעדכנת:

```
>>> keyboard_shortcuts["cut"] = "Ctrl+X"
>>> keyboard_shortcuts["paste"] = "Shift+Insert"
>>> keyboard_shortcuts["paste"]
'Shift+Insert'
```

כל מפתח ייחודי. השמה למפתח שכבר קיים לא יוצרת זוג נוסף ולא זורקת שגיאה. היא **מחליפה** את הערך.

מחיקה

```
>>> del keyboard_shortcuts["copy"]
>>> keyboard_shortcuts.pop("paste")
'Shift+Insert'
```

```
>>> keyboard_shortcuts
{'undo': 'Ctrl+Z', 'cut': 'Ctrl+X'}
```

`del` מוחק. `pop` מוחק **ומחזיר את הערך**.

לעבור על מילון

```
>>> keyboard_shortcuts.keys()
dict_keys(['undo', 'cut'])
>>> keyboard_shortcuts.values()
dict_values(['Ctrl+Z', 'Ctrl+X'])
>>> keyboard_shortcuts.items()
dict_items([('undo', 'Ctrl+Z'), ('cut', 'Ctrl+X')])
>>> for info in keyboard_shortcuts.items():
...     print("%s: %s" % info)
...
undo: Ctrl+Z
cut: Ctrl+X
```

`items` מחזיר כל זוג **כטאפל**, ולכן אפשר להעביר אותו ישר לתבנית המחרוזת. מה שחוזר מ- `keys` , `values` ו- `items` הוא **view object**: אפשר לעבור עליו בלולאה, והוא לא העתק של המילון.

בדיקה עצמית

1. מה הטיפוס של `("a")` ? ושל `("a",)` ?
2. מה ידפיס `"%s is %d" % ("pi", 3)` ?
3. מה קורה ב- `d = {"a": 1}` ואז `d["a"] = 2` ?
4. במה `pop` שונה מ- `del` ?
5. נקודה במרחב, `(x, y, z)` : טאפל או רשימה? ורשימת קניות?

תשובות. 1: `str` , ו- `tuple` . 2: `pi is 3` . 3: הערך של `"a"` מתעדכן ל-2. מפתח ייחודי, אז אין זוג שני. 4: `pop` גם מחזיר את הערך שנמחק. 5: נקודה: טאפל, כי מספר האיברים קבוע ולמיקום יש משמעות. רשימת קניות: רשימה, כי מוסיפים ומוחקים ממנה.

סיכומים אישיים מתוך סרטוני הקורס Self.py בקמפוס IL, ולא חומר רשמי של הקורס.