יחידה 6

רשימות

שלושה פרקים, 16 דקות. יצירה, גישה, רצפים, ושינוי של רשימה.

מפת היחידה

16 חלקים ▲

פרק	דקה	מה קורה שם
6.1	0:30	רשימה ריקה, ו- `type`
6.1	1:00	איברים מטיפוסים שונים, וגישה במיקום
6.1	**1:39**	**רשימה בתוך רשימה**
6.1	3:18	הפונקציה `list`
6.2	0:00	רשימה ומחרוזת הן רצפים
6.2	1:01	גישה וחיתוך
6.2	2:05	שרשור, ו- `TypeError`
6.2	3:11	הכפלה ו- `len`
6.3	**0:31**	**רשימה ניתנת לשינוי**
6.3	1:02	`append` ו- `+=`
6.3	1:33	שינוי איבר, ו- `IndexError`
6.3	3:11	`remove`, ו- `ValueError`
6.3	3:42	`in`
6.3	4:46	`count` ו- `sort`
6.3	**5:16**	**`sort` מול `sorted`**
6.3	6:22	מיון עם `key`

6.1 · מהי רשימה

רשימה (`list`) מחזיקה כמה אובייקטים במקום אחד, ושומרת על הסדר שלהם. סוגריים מרובעים, ופסיקים בין האיברים:

```
>>> movies = []
>>> type(movies)
<class 'list'>
>>> movies = ["Titanic", "Up", "Inception",
...           "Coco", "Amelie"]
```

האיברים לא חייבים להיות מאותו טיפוס:

```
>>> titanic = [1997, "Leonardo DiCaprio",
...            "Kate Winslet", 194, True]
>>> titanic[0]
1997
>>> titanic[1]
'Leonardo DiCaprio'
>>> titanic[3]
194
```

שנה, שני שחקנים, אורך בדקות, וצפיתי או לא. גישה היא לפי מיקום, **מאפס**, בדיוק כמו במחרוזת.

רשימה בתוך רשימה

איבר ברשימה יכול להיות רשימה. לאיבר הפנימי ניגשים עם **זוג סוגריים נוסף**:

```
>>> unique = [["a", "list"], ["of", "lists"]]
>>> unique[0]
['a', 'list']
>>> unique[0][1]
'list'
>>> unique[1][0]
'of'
```

הסוגריים הראשונים בוחרים רשימה פנימית, והשניים בוחרים איבר בתוכה. זה שימושי לטבלה: כל רשימה פנימית היא שורה, למשל אדם, שנת לידה ומקצוע.

`list()`

```
>>> empty_list = list()
>>> empty_list
[]
```

```
>>> list("Letters")
['L', 'e', 't', 't', 'e', 'r', 's']
```

`list` על מחרוזת מפרק אותה לתווים.

6.2 · רשימות ומחרוזות הן רצפים

רצף הוא אוסף של פריטים בסדר מסוים. מחרוזת היא רצף של תווים, ורשימה היא רצף של איברים. ולכן מה שלמדנו ביחידה 3 עובד גם כאן:

```
>>> my_subject = "chemistry"
>>> chemical_elements = ["hydrogen", "helium",
...                      "lithium", "beryllium"]
```

פעולה	מחרוזת	רשימה
גישה	`my_subject[3]` הוא `'m'`	`chemical_elements[3]` הוא `'beryllium'`
מהסוף	`my_subject[-1]` הוא `'y'`	`chemical_elements[-1]` הוא `'beryllium'`
חיתוך	`my_subject[1:3]` הוא `'he'`	`chemical_elements[1:3]` הוא `['helium', 'lithium']`
אורך	`len(my_subject)` הוא 9	`len(chemical_elements)` הוא 4

```
>>> chemical_elements[1:3]
['helium', 'lithium']
>>> len(chemical_elements)
4
```

שרשור: רק רשימה לרשימה

```
>>> chemical_elements + "boron"
Traceback (most recent call last):
  ...
TypeError: can only concatenate list (not "str") to list
>>> chemical_elements + ["boron"]
['hydrogen', 'helium', 'lithium', 'beryllium', 'boron']
```

כמו שאי אפשר לשרשר מספר למחרוזת. העטיפה בסוגריים מרובעים הופכת את `"boron"` לרשימה עם איבר אחד.

הכפלה

```
>>> ["H", "O"] * 3
['H', 'O', 'H', 'O', 'H', 'O']
```

6.3 · פעולות על רשימות

ההבדל הגדול ממחרוזת

מחרוזת היא immutable. רשימה היא mutable: אפשר להוסיף, לשנות ולמחוק אחרי שהיא נוצרה.

```
>>> solar_system = ["Mercury", "Venus", "Earth",
...                 "Mars", "Jupiter", "Saturn",
...                 "Uranus", "Neptune"]
>>> len(solar_system)
8
```

להוסיף

```
>>> solar_system.append("Novo")
>>> solar_system[-1]
'Novo'
>>> solar_system += ["Novo"]
>>> solar_system[-2:]
['Novo', 'Novo']
```

שתי דרכים: `append` לאיבר אחד, ו- `+=` עם רשימה (שיכולה להכיל כמה איברים). שימו לב: `append` **לא מחזירה כלום**. היא משנה את הרשימה עצמה.

לשנות

```
>>> solar_system[8] = "De Novo"
>>> solar_system[9] = "Sun"
>>> solar_system[10] = "Sun"
Traceback (most recent call last):
```

```
  ...
IndexError: list assignment index out of range
```

משנים איבר במיקום **שקיים**. מיקום שלא קיים זו שגיאה, וכדי להוסיף בסוף משתמשים ב- append .

למחוק

```
>>> solar_system.remove("De Novo")
>>> solar_system.remove("De Novo")
Traceback (most recent call last):
  ...
ValueError: list.remove(x): x not in list
```

- remove מקבלת **ערך**, לא מיקום, ומוחקת **רק את המופע הראשון** שלו
- ערך שלא קיים ברשימה: ValueError

לכן בודקים לפני:

```
>>> "Earth" in solar_system
True
>>> "Pluto" in solar_system
False
```

עוד מתודות ופונקציות

(הטמפרטורות כאן לדוגמה בלבד.)

```
>>> temperature = [59, -180, 867, -180, 333]
>>> temperature.count(-180)
2
>>> max(temperature)
867
>>> sorted(temperature)
[-180, -180, 59, 333, 867]
>>> temperature
[59, -180, 867, -180, 333]
>>> temperature.sort()
>>> temperature
[-180, -180, 59, 333, 867]
```

	מה מחזיר	משנה את הרשימה?
`sorted(lst)`	**רשימה חדשה** ממוינת	לא
`lst.sort()`	None	**כן**

זה ההבדל ממה שהכרנו במחרוזות: `"abc".upper()` מחזירה מחרוזת חדשה ולא נוגעת במקורית. מתודות של רשימה כמו `sort` ו- `append` **משנות** את הרשימה עצמה.

מיון עם key

```
>>> astro_objects = ["moon", "asteroid", "galaxy"]
>>> sorted(astro_objects)
['asteroid', 'galaxy', 'moon']
>>> sorted(astro_objects, key=len)
['moon', 'galaxy', 'asteroid']
```

`key` הוא פרמטר אופציונלי שמקבל **פונקציה**. היא רצה על כל איבר, והמיון נעשה לפי מה שהיא החזירה. כאן `len`, ולכן לפי אורך. (שימו לב: `len` בלי סוגריים. מעבירים את הפונקציה עצמה, כמו ביחידה 5.)

אפשר להעביר גם פונקציה שכתבתם. אילו פונקציות מקבלות `key`, ומה עוד הן מקבלות, מוצאים בתיעוד של פייתון.

בדיקה עצמית

1. עם `grid = [[1, 2], [3, 4]]`, מה זה `grid[1][0]`?
2. מה ההבדל בין `lst.append([5])` לבין `lst += [5]`?
3. מה יחזיר `x = [3, 1, 2].sort()`, ועל מה מצביע `x`?
4. למה `"hi"[0] = "H"` נכשל, ו- `["h", "i"][0] = "H"` לא?
5. איך ממיינים רשימת מילים מהארוכה לקצרה? (רמז: `help(sorted)`)

תשובות. 1: 3. 2: `append([5])` מוסיף **רשימה** כאיבר אחד, `[...] += [5]` מוסיף את 5. 3: `None`. `sort` משנה את הרשימה ולא מחזירה אותה. צריך `sorted`. 4: מחרוזת היא immutable, ורשימה היא mutable. 5: `sorted(words, key=len, reverse=True)`.

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