יחידה 3

מחרוזות

ארבעה פרקים, 24 דקות. מרכאות, תווים מיוחדים, חיתוך, ומתודות.

מפת היחידה

22 חלקים ▲

פרק	דקה	מה קורה שם
3.1	0:23	מה זו מחרוזת
3.1	**1:01**	**השגיאה הראשונה: מרכאות שלא תואמות**
3.1	2:50	גרש באמצע מחרוזת
3.1	4:16	הלוכסן ההפוך, `\`
3.1	4:47	שרשור עם `+` והכפלה עם `*`
3.1	5:26	`str()`
3.2	0:46	`\n` ו- `\t`
3.2	1:27	נתיב קובץ שנשבר
3.2	2:37	מרכאות משולשות
3.3	1:01	אינדקסים, מאפס
3.3	1:41	אינדקס שלילי
3.3	2:06	`IndexError`
3.3	**2:52**	**חיתוך (slicing)**
3.3	3:42	להשמיט התחלה או סוף
3.3	5:18	חיתוך בקפיצות
3.3	**6:04**	**מחרוזת לא ניתנת לשינוי**
3.4	0:58	למה פונקציות
3.4	1:48	`math.sqrt`
3.4	2:25	פונקציה מול מתודה
3.4	3:11	`upper` ו- `zfill`
3.4	4:40	`dir`
3.4	5:04	לכתוב טוב: רווחים

3.1 · יצירת מחרוזות

מחרוזת (`str`) היא רצף של תווים, אפילו תו אחד, בתוך מרכאות. המרכאות אומרות לפייתון שזה טקסט, ולא שם של פקודה או של משתנה.

השגיאה הראשונה

```
>>> 'I really hope defining this string will work"
Traceback (most recent call last):
  ...
SyntaxError: unterminated string literal (detected at line 1)
```

איך קוראים את ההודעה. ה-Traceback אומר באיזו שורה פייתון נתקעה, סמן קטן מצביע על המקום, והשורה האחרונה אומרת מה סוג השגיאה.

כאן זו **שגיאת תחביר**, `SyntaxError` : הקוד לא בנוי לפי חוקי השפה. המחרוזת נפתחה במרכאה יחידה, ופייתון הגיעה לסוף השורה בלי למצוא את המרכאה הסוגרת.

בסרטון ההודעה מופיעה כ- `EOL while scanning string literal` , ניסוח של גרסאות ישנות יותר. בגרסה עדכנית תופיע ההודעה שלמעלה. המשמעות זהה.

הכלל: כפולות או יחידות, שתיהן בסדר, כל עוד אותה מחרוזת נפתחת ונסגרת באותו סוג.

גרש באמצע

```
>>> 'What's wrong here?'
Traceback (most recent call last):
  ...
SyntaxError: unterminated string literal (detected at line 1)
```

פייתון קוראת את המחרוזת `'What'` , ונתקעת באות `s` שאחריה. ההודעה בסרטון הייתה `invalid syntax`, ושוב, זה ניסוח של גרסה אחרת.

שלושה פתרונות:

```
>>> "What's wrong here?"
"What's wrong here?"
>>> 'She said "maybe" and left'
'She said "maybe" and left'
>>> print("She said \"don't\" twice")
She said "don't" twice
```

1. גרש בפנים: עוטפים במרכאות כפולות

2. מרכאות כפולות בפנים: עוטפים ביחידות
3. גם וגם: **לוכסן הפוך** (\ , backslash) לפני התו הבעייתי. השם שלו הוא **escape character**, כי הוא "מבריח" את התו שאחריו מהמשמעות הרגילה שלו

שרשור והכפלה

```
>>> first = "We "
>>> second = "belong "
>>> third = "together"
>>> first + second + third
'We belong together'
>>> duplicate_me = ":) "
>>> duplicate_me * 8
':) :) :) :) :) :) :) :) '
```

הטיפ: רווח בסוף כל חלק. בלעדיו המילים נדבקות.

`str()`

```
>>> x = 3
>>> type(x)
<class 'int'>
>>> x + x
6
>>> x = str(x)
>>> type(x)
<class 'str'>
>>> x + x
'33'
```

אותו `+` , שתי תוצאות. על מספרים זה חיבור, ועל מחרוזות זה שרשור.

3.2 · תווים מיוחדים

תו	מה עושה
`\n`	שורה חדשה
`\t`	הזחה (טאב)
`\\`	לוכסן הפוך אחד, כמו שהוא

טבלת תוצאות במחרוזת אחת

```
>>> rounds_and_winners = "Round\tWinner\n1\tA\n2\tB\n3\tA"
>>> print(rounds_and_winners)
Round   Winner
1   A
2   B
3   A
```

שימו לב: רק `print` מציג את השורות והטאבים. בלי `print` האינטרפרטר מראה את המחרוזת עם `\n` ו- `\t` כמו שהם.

כשזה מפריע: נתיב של קובץ

```
>>> print("C:\tournaments\new_scores.txt")
C:  ournaments
ew_scores.txt
```

פייתון ראתה `\t` ו- `\n` והדפיסה טאב ושורה חדשה. **הפתרון:** לוכסן הפוך נוסף, שמנטרל את המיוחד:

```
>>> print("C:\\tournaments\\new_scores.txt")
C:\tournaments\new_scores.txt
```

מרכאות משולשות

שלוש מרכאות בהתחלה ושלוש בסוף. אפשר לשבור שורות ב-Enter, בלי `\n` , ואפשר לכתוב גרש בפנים בלי בעיה:

```
>>> how_to_win = """Three steps to win at chess:
... Don't lose your queen
... Control the center
... Checkmate"""
>>> print(how_to_win)
Three steps to win at chess:
Don't lose your queen
Control the center
Checkmate
```

`...` הוא האינטרפרטר שמחכה להמשך. לא מקלידים אותו.

שימוש נוסף, שיגיע בהמשך הקורס: **תיעוד** של קוד, הסבר מפורט על מה שקטע קוד עושה.

3.3 · גישה למחרוזת

אינדקס

גישה לתו היא **לפי המיקום שלו, ולא לפי הערך.** סופרים **מאפס**, ומבקשים בסוגריים מרובעים:

```
>>> card_str = "joker"
>>> card_str[0]
'j'
>>> card_str[1]
'o'
```

תו	j	o	k	e	r
מההתחלה	0	1	2	3	4
מהסוף	-5	-4	-3	-2	-1

```
>>> card_str[-1]
'r'
>>> card_str[-5]
'j'
```

מעבר לגבול

```
>>> card_str[5]
Traceback (most recent call last):
  ...
IndexError: string index out of range
```

ל- `"joker"` יש חמישה תווים, ולכן המיקום האחרון הוא 4. גם `card_str[-6]` ייתן את אותה שגיאה.
הטיפ: לבדוק את האורך לפני. איך, יגיע בהמשך.

חיתוך

`[התחלה:סוף]` . ההתחלה נכללת, **הסוף לא**:

```
>>> game = "gaming"
>>> game[0:2]
'ga'
```

```
>>> game[2:5]
'min'
```

סוף פחות התחלה הוא אורך המקטע. `2 - 5` הם שלושה תווים.

להשמיט: בלי התחלה, פייתון מתחילה מ-0. בלי סוף, היא ממשיכה עד הסוף. בלי שניהם, מתקבלת כל המחרוזת:

```
>>> game[:2]
'ga'
>>> game[2:]
'ming'
>>> game[:]
'gaming'
>>> game[:-1]
'gamin'
```

`[:-1]` הוא הכל חוץ מהתו האחרון.

בקפיצות

נקודתיים שלישיות, וגודל הקפיצה:

```
>>> game[0:5:2]
'gmn'
>>> game[1::2]
'aig'
```

כך מקבלים את כל המיקומים הזוגיים, או האי-זוגיים, **בלי לולאה**.

אי אפשר לשנות מחרוזת

```
>>> my_string = "Change me"
>>> my_string[0] = "S"
Traceback (most recent call last):
  ...
TypeError: 'str' object does not support item assignment
```

מחרוזת היא **immutable**, לא ניתנת לשינוי. המונח הזה יחזור.

מה כן עושים: יוצרים מחרוזת חדשה עם השינוי, ומעבירים את המשתנה להצביע עליה:

```
>>> my_string = "S" + my_string[1:]
>>> my_string
'Shange me'
```

זה בדיוק מה שקורה בכל "שינוי" של משתנה מחרוזת: ההצבעה עוברת למחרוזת חדשה, והישנה לא נגעה.

3.4 · פונקציות ומתודות

למה בכלל. ביטוי שיש בו חמישה שורשים ריבועיים פירושו לכתוב חמש פעמים את הקוד שמחשב שורש. ואם יש בו טעות, מתקנים חמש פעמים. פונקציה נכתבת פעם אחת, ונקראת כמה פעמים שצריך.

פונקציה

שם, סוגריים, ובתוכם **הפרמטרים**, מופרדים בפסיקים:

```
>>> import math
>>> math.sqrt(9)
3.0
>>> math.sqrt(16) + math.sqrt(25)
9.0
```

`import math` טוען את הספרייה, ואז `math.sqrt` קורא לפונקציה מתוכה. שימו לב שהתוצאה היא `3.0` ולא `3` : `sqrt` מחזירה תמיד `float` .

פונקציות משלכם מגיעות ביחידה 5.

מתודה

פונקציה ששייכת לטיפוס מסוים. **קודם האובייקט**, אחר כך נקודה, השם וסוגריים:

```
>>> "print me in upper case".upper()
'PRINT ME IN UPPER CASE'
>>> "1".zfill(5)
'00001'
>>> "234".zfill(5)
'00234'
```

- `upper` בלי פרמטרים. הסוגריים נכתבים בכל זאת, ריקים
- `zfill(רוחב)` מרפדת באפסים משמאל עד הרוחב שביקשתם

הנקודה מטעה. ב- `math.sqrt` היא אומרת "מתוך הספרייה". ב- `"1".zfill` היא אומרת "המתודה של האובייקט הזה". ומאחורי הקלעים, האובייקט עצמו נשלח למתודה בתור הפרמטר הראשון.

איך מוצאים מתודות

```
dir(str)
```

מחזיר רשימה ארוכה של כל מה שאפשר לעשות עם מחרוזת. `upper` ו- `zfill` נמצאות בסוף.

לכתוב טוב: רווחים

כלל	נכון	לא
רווח אחד **אחרי** כל פסיק	`f(a, b, c)`	`f(a,b,c)`
פרמטר אחד: בלי רווחים בכלל	`f(a)`	`f( a )`
רווח **משני** צדי אופרטור	`x = 1 + 2`	`x=1+2`

בדיקה עצמית

1. מה ההבדל בין `print("a\tb")` לבין הקלדת `"a\tb"` באינטרפרטר?
2. עם `s = "python"` : מה זה `s[-3:]` ?
3. מה זה `s[::-1]` ? (לא הופיע בסרטון. נסו לנחש לפי הכלל.)
4. איך הופכים את `"cat"` ל- `"bat"` ?
5. מה ההבדל בין `math.sqrt(x)` לבין `x.upper()` ?

תשובות. 1: `print` מציג טאב אמיתי. האינטרפרטר מציג את המחרוזת כמו שהיא, עם `\t` . 2: `'hon'` . 3: `'nohtyp'` . קפיצה שלילית עוברת על המחרוזת מהסוף להתחלה. 4: `"cat"[1:] + "b"` . אי אפשר לשנות תו בתוך מחרוזת, אז בונים חדשה. 5: `sqrt` היא פונקציה מתוך הספרייה `math` . `upper` היא מתודה של מחרוזות, והאובייקט שלפני הנקודה הוא מה שהיא פועלת עליו.

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