יחידה 2

משתנים

ארבעה פרקים, 26 דקות. טיפוסים, ביטויים לוגיים, קלט, ושמות.

מפת היחידה

19 חלקים ▲

פרק	דקה	מה קורה שם
2.1	0:55	מה זה משתנה, והשמה עם `=`
2.1	1:40	**משתנה הוא מצביע**, לא קופסה
2.1	2:22	הטיפוסים `str` , `int` , `float`
2.1	4:31	לכתוב טוב: לא להחליף טיפוס של משתנה
2.1	5:35	חישוב ממשתנים אחרים
2.2	0:21	הטיפוס הבוליאני
2.2	1:08	ביטוי לוגי כערך
2.2	2:41	`==` , `!=` , `not` , `and` , `or`
2.2	4:43	לכתוב טוב: רווחים וסוגריים בביטוי ארוך
2.2	5:56	האופרטור `in`
2.2	**6:23**	**`help()` , ואיך יוצאים ממנו**
2.3	0:44	קלט מהמשתמש עם `input`
2.3	**2:43**	**`TypeError` : הקלט הוא תמיד מחרוזת**
2.3	3:17	הסבה בין טיפוסים
2.3	4:09	החלפת ערכים בשורה אחת
2.4	0:33	קונבנציות כתיבה ו-PEP 8
2.4	2:20	פייתון רגישה לגודל אות
2.4	2:57	אותיות קטנות, קו תחתון, וקבועים
2.4	3:59	שמות משמעותיים

2.1 · משתנה הוא מצביע

משתנה הוא שם שמצביע על ערך. כותבים את השם, סימן שווה, ואת הערך:

```
>>> course = "Python"
>>> unit_name = 'variables'
>>> unit_number = 2
>>> chapter_length = 7.25
>>> course
'Python'
```

כדי לראות ערך של משתנה באינטרפרטר, מקלידים את השם ולוחצים Enter.

מה קורה בזיכרון. `course = "Python"` יוצר קודם אובייקט שמכיל את הטקסט, ורק אחר כך משתנה בשם `course` שמצביע עליו. זו ההבחנה שהפרק חוזר עליה פעמיים: המשתנה אינו הערך. הוא מצביע אליו.

ארבעה טיפוסים

כל אובייקט בפייתון הוא מטיפוס מסוים:

ערך	טיפוס	מה זה
`"Python"` , `'variables'`	`str`	מחרוזת. מרכאות כפולות או יחידות, שתיהן תקינות
`2`	`int`	מספר שלם
`7.25` , `7.0`	`float`	מספר עשרוני. גם `7.0` הוא `float`
`True` , `False`	`bool`	בוליאני. מגיע בפרק 2.2

לא מצהירים על הטיפוס. פייתון מזהה אותו לבד מהערך, וזה מה שנקרא **dynamically typed**.

שינוי ערך

"לשנות" משתנה זה להעביר אותו להצביע על אובייקט אחר:

```
>>> chapter_length = 7.25
>>> chapter_length = "7.25 minutes"
>>> chapter_length
'7.25 minutes'
```

לכתוב טוב: הדוגמה הזו עובדת, אבל היא **bad practice**. המשתנה התחיל כ- `float` ועבר להצביע על `str`. פעולה שמתאימה לטיפוס אחד לא בהכרח מתאימה לשני, וכך נוצרות שגיאות שקשה למצוא. משתנה נשאר באותו טיפוס.

חישוב ממשתנים

```
>>> current_unit = unit_number
>>> next_unit = current_unit + 1
>>> next_unit
3
>>> print(current_unit – 1)
1
```

אחרי `current_unit = unit_number` שני המשתנים מצביעים **על אותו אובייקט**, `2` . ההדפסה בסוף היא כמה יחידות כבר הסתיימו: היחידה הנוכחית עוד לא נגמרה, ולכן מפחיתים אחת.

2.2 · בוליאני וביטויים לוגיים

לטיפוס `bool` יש שני ערכים בלבד: `True` ו- `False` . שימו לב לאות הגדולה, `true` בלי אות גדולה הוא שגיאה.

```
>>> i_am_hungry = True
```

ביטוי לוגי כערך

במקום לכתוב `True` או `False` ישירות, אפשר להשים תוצאה של ביטוי:

```
>>> number_of_sandwiches = 2
>>> i_am_hungry = number_of_sandwiches < 4
>>> i_am_hungry
True
```

כל מספר שאינו אפס נחשב `True` , ואפס הוא תמיד `False` . זה ייראה שימושי כשיגיעו הלולאות.

```
>>> bool(5)
True
>>> bool(0)
False
```

האופרטורים

```
>>> number_of_sandwiches == 5
False
>>> number_of_sandwiches != 5
```

```
True
>>> not number_of_sandwiches != 5
False
```

`==` הוא השוואה, ו- `=` הוא השמה. מי שמבלבל ביניהם יבלבל עוד הרבה פעמים, ולכן כדאי לעצור על זה עכשיו.

`and` דורש ששני הצדדים יהיו נכונים, ו- `or` מסתפק באחד:

```
>>> cups_of_coffee = 3
>>> number_of_sandwiches < 4 and cups_of_coffee > 5
False
>>> number_of_sandwiches < 4 or cups_of_coffee > 5
True
```

לכתוב טוב: ביטוי ארוך

ביטוי שבנוי משלושה תנאים: רעב אם אכלתי לפחות כריך אחד וגם פחות מארבעה, **או** בדיוק שתי קערות מרק. שני שינויים הופכים אותו לקריא: רווח משני צדי כל אופרטור, וסוגריים סביב כל תנאי.

```
>>> bowls_of_soup = 0
>>> i_am_hungry = (((number_of_sandwiches >= 1)
...                 and (number_of_sandwiches < 4))
...                or (bowls_of_soup == 2))
>>> i_am_hungry
True
```

`in`

בודק אם מחרוזת אחת נמצאת בתוך אחרת:

```
>>> "break" in "breakfast"
True
>>> "b" in "food"
False
```

בקשת עזרה

אף אחד לא זוכר את כל הפקודות. מה שצריך לזכור הוא איפה מחפשים.

- `help()` פותח את ספריית העזרה של פייתון. שם מקלידים, למשל, `in` ומקבלים הסבר מלא.
- **רווח** מדפדף, ו- **`q`** יוצא. בלי זה אפשר להיתקע שם.

- `help("in")` הולך ישר לנושא, בלי להיכנס לממשק.
- באינטרנט יש תיעוד רב, אבל **רק לפייתון 3**. זו הגרסה של הקורס, וחלק מההסברים ברשת עוד נכתבו לגרסה 2.

2.3 · קלט, הסבה והחלפה

`input`

`input` עוצר את התוכנית ומחכה שהמשתמש יקליד ויקיש Enter. מה שבסוגריים מוצג לו על המסך:

```
name = input("What is your name? ")
print(name)
```

הקיצור `_` : באינטרפרטר, המשתנה `_` מחזיק את התוצאה האחרונה. אם הרצתם `input(...)` בלי לשמור, `_` עדיין מכיל את מה שהוקלד. זה עובד רק באינטרפרטר, לא בקובץ.

התקלה המרכזית של הפרק

```
>>> number = "5"
>>> print(number ** 2)
Traceback (most recent call last):
  ...
TypeError: unsupported operand type(s) for ** or pow(): 'str' and 'int'
```

בשורה הראשונה `"5"` במקום `input()` , כי זה בדיוק מה ש- `input` מחזיר: **`input` מחזיר תמיד מחרוזת**, גם כשהוקלד מספר. ואי אפשר להעלות מחרוזת בחזקה.

הסבה (casting)

כותבים את הטיפוס הרצוי, ובסוגריים את הערך:

```
>>> int(number) ** 2
25
>>> float(number)
5.0
```

החלפת ערכים בשורה אחת

```
>>> peter_pan_home = "Neverland"
>>> snow_white_home = "the forest"
```

```
>>> peter_pan_home, snow_white_home = snow_white_home, peter_pan_home
>>> peter_pan_home
'the forest'
>>> snow_white_home
'Neverland'
```

בשפות כמו C ו-Java החלפה דורשת משתנה זמני ושלוש שורות.

2.4 · שמות למשתנים

קונבנציות כתיבה הן מוסכמות של הקהילה על איך קוד נראה. פייתון לא אוכפת אותן, והקוד ירוץ גם בלעדיהן. המסמך שמרכז אותן הוא **PEP 8**, והקורס מצפה לעבוד לפיו.

פייתון רגישה לגודל אות

```
>>> nose_length = 4
>>> Nose_length = 6
>>> Nose_length - nose_length
2
```

אלה **שני משתנים שונים** (case-sensitive), והקוד תקין. אבל מי שקורא אותו לא יודע מה ההבדל ביניהם. כאן הקונבנציות נכנסות.

הכללים

כלל	דוגמה
אותיות קטנות	`character = "Pinocchio"`
יותר ממילה אחת: קו תחתון ביניהן	`nose_length`
קבוע, ערך שלא משתנה בתוכנית: אותיות גדולות	`PI = 3.14`
שם שאומר מה יש בו	`fairy_tale_file_with_extension` , **ולא** `file`

קבוע הוא קונבנציה בלבד. אם משנים את `PI` באמצע התוכנית, פייתון לא תזרוק שגיאה. האחריות על המתכנת.

שמות כלליים כמו `number` או `name` עובדים, אבל לא אומרים כלום. שם ארוך ומפורט עדיף, כי הוא חוסך לקורא את הניחוש.

בדיקה עצמית

1. אחרי `a = 5` ואז `b = a` , על מה מצביע `b` ?
2. מה הטיפוס של `"2" + "2"` , ומה הערך?
3. מה התוצאה של `not 0` ?
4. המשתמש הקליד `7` ב- `x = input()` . מה יחזיר `x * 2` ?
5. איזה שם עדיף לפי PEP 8: `Total` , `total_price` או `tp` ?

תשובות. 1: על אותו אובייקט, `5` . שני שמות, ערך אחד. 2: `str` , והערך `'22'` . 3: `True` , כי `0` נחשב `False` . 4: `'77'` . הקלט הוא מחרוזת, וכפל מחרוזת במספר משכפל אותה. כדי לקבל `14` צריך `int(x) * 2`. 5: `total_price` . אותיות קטנות, קו תחתון, ושם שאומר מה יש בו.

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