scope של משתנים

מה תבצע התוכנית הבאה?

```
def speak():
    word = "hi"
    print(word)


speak()
print(word)
```

כפי שאולי שמתם לב, PyCharm מסמן את המילה word באדום. אם ננסה להריץ את הקוד, יודפס הפלט הבא:

```
hi
Traceback (most recent call last):
    print word
NameError: global name 'word' is not defined
```

אבל, איך זה יכול להיות שישנה שגיאת הרצה? הרי הפונקציה main קראה לפונקציה speak, אשר בה הוגדר המשתנה word ואף הודפס למסך. מדוע main לא מכירה את המשתנה word?

הסיבה היא שהמשתנה word הוגדר בפונקציה speak והוא קיים רק בה. ברגע שיצאנו מהפונקציה speak, המשתנה word פשוט נמחק ואינו קיים יותר. מכאן שה-scope של המשתנה word הוא אך ורק בתוך הפונקציה speak.

נחדד את ההסבר ותוך כדי נבין את ההבדל בין משתנה גלובלי למשתנה לוקלי, מקומי. נניח שכתבנו את הקוד הבא, שימו לב למשתנה החדש name:

```
def speak():
    word = "hi"
    print("{} {}".format(word, name))

name = "Shooki"
speak()
```

התוכנית תדפיס hi Shooki. מדוע? משום ש-name הוא משתנה גלובלי – משתנה שמוגדר מחוץ לכל הפונקציות, ולכן הוא מוכר בכולן, כלומר ה-scope של name הוא כל הסקריפט שלנו.

נתקדם עוד שלב – כעת אנחנו מגדירים את המשתנה word פעמיים. אחת כגלובלי ואחת כלוקלי... מה יודפס הקוד הבא?

```
def speak():
    word = "hi"
    print(word)

word = "bye"
speak()
```

ובכן, יודפס hi. מדוע? הרי אמרנו שמשתנה גלובלי מוכר גם בתוך פונקציה? נכון, אלא שבפונקציה speak אנחנו "דורסים" את המשתנה הגלובלי word עם משתנה לוקלי בעל אותו שם. כעת כאשר נפנה בתוך speak למשתנה word, הוא כבר לא יכיר את המשתנה הגלובלי, אלא רק את מה שהוגדר לוקלית.

ניסיון נוסף... מה יודפס הקוד כעת?

```
def speak():
    word = "hi"
    print(word)

word = "bye"
speak()
print(word)
```

יודפס:

```
hi
bye
```

מדוע? את ההדפסה של hi כבר הבנו. כאשר speak מסיימת את הריצה שלה, המשתנה הלוקלי word נמחק, וכעת קורה משהו מעניין – מסתבר שהמשתנה הגלובלי word לא נמחק, אלא פשוט שמר בצד. לפיתון יש מידע של עדיפויות: כאשר פונים למשתנה בתוך פונקציה, קודם כל פייתון מחפש אם קיים משתנה לוקלי כזה, ולאחר מכן אם קיים משתנה גלובלי. ההדפסה השניה מתרחשת מתוך הפונקציה main, שמכירה רק את המשתנה הגלובלי word.

ניסיון אחרון... מה יודפס הקוד הבא?

```
def speak():
    word += " you"
    print(word)

word = "love"
speak()
print(word)
```

שגיאה!

```
Traceback (most recent call last):
    word += ' you'
UnboundLocalError: local variable 'word' referenced before assignment
```

המשתנה word אינו מוגדר. אבל מדוע? הרי הגדרנו משתנה גלובלי בשם זה? הסיבה היא, שכאשר אנחנו מבצעים פעולה שמשנה את ערכו של משתנה בתוך פונקציה, כמו פעולת חיבור, פייתון מניח שלמשתנה יש עותק מקומי והוא מנסה לפעול עליו.

כעת נסקור שתי שיטות לתקן את השגיאה בקוד. אפשרות א', והיא הפחות טובה, היא להשתמש במילה global בתוך הפונקציה, כך:

```
def speak():
    global word
    word += " you"
    print(word)

word = "love"
speak()
print(word)
```

בעקבות הריצה יודפס:

```
love you
love you
```

הפקודה global word אומרת לפייתון – 'ראה, אנו עומדים לעבוד בתוך הפונקציה עם המשתנה הגלובלי word. אם ננסה לשנות את ערכו, עשה זאת בלי להכריז שהוא אינו מוכר לך'.

האפשרות השניה, היא להעביר לפונקציה speak את word בתור פרמטר, כך:

```
def speak(word):
    word += " you"
    print(word)
    return word

word = "love"

speak(word)
print(word)
```

בעקבות הריצה יודפס:

```
love you
love
```

כעת, מדוע האפשרות הראשונה אינה מומלצת? כפי ששמתם לב, האפשרות הראשונה משנה את ערכו של המשתנה word גם מחוץ לפונקציה, הודפס פעמיים love you. כלומר, הפונקציה שינתה את ערכו של המשתנה. דמיינו שאתם כותבים את הפונקציה main ומתכנת אחר כותב את הפונקציה speak. כעת תארו לעצמכם את ההפתעה, שהפונקציה שינתה ערך של משתנה בלי שהיה לכם מושג שהיא עשתה זאת! אם הייתם רוצים לאפשר לפונקציה לשנות את הערך של המשתנה, הייתם מעבירים לה אותו כפרמטר ולוחצים להציב את ערך החזרה של הפונקציה במשתנה, כך:

```
word = speak(word)
```

זו הדרך הנכונה לשנות משתנה על ידי פונקציה – לקבל אותו כפרמטר ולהחזיר אותו עם return לקוד שקרא לו, כך:

```
def speak(word):
    word += " you"
    print(word)
    return word

word = "love"
word = speak(word)
print(word)
```

תרגילים

- כיתבו פונקציה בשם factorial שמחזירה את התוצאה של !5 (5 עצרת). אין צורך להשתמש ברקורסיה.