פרק 7 פונקציות

תוכן העניינים :

פרק 7 : פונקציות

7.1 יצירה/הגדרה של פונקציה	2
7.2 ארגומנטים	3
7.3 ארגומנטים ופרמטרים	3
7.4 הגדרת פונקציה כאשר לא יודעים כמה ארגומנטים מועברים אל הפונקציה (*args)	4
7.5 ארגומנטים של מילות מפתח	6
7.6 ארגומנטים שרירותיים של מילות מפתח, **kwargs	6
7.7 ברירת מחדל של ערך פרמטר	8
7.8 העברה של list כארגומנט	8
7.9 ערכים מוחזרים - return values	9
7.9.1 החזרה של יותר מערך אחד	9
7.10 ההצהרה pass	10
7.11 משתנים לוקאליים וגלובאליים	11
7.12 הפונקציה lambda	14
7.13 השימוש בפונקציה ( )id לדעת כתובת של אובייקט	15
7.14 כיצד מועברים ארגומנטים אל הפרמטרים בפונקציה ?	15
7.15 שימוש במילה is להשוואת כתובות	16
7.16 רקורסיה	17
7.17 תרגילים	18

בפרק 7 של תוכנית הלימודים להנדסאים במקצוע שפת פייתון מופיע:

פרק 7 : פונקציות

יעדים

היכרות עם עולם הפונקציות שבו מצרפים יחד מספר הוראות, במטרה לבצע אלגוריתם מוגדר.

תכנים

1. תחום קיום של משתנים (Scope)
2. משמעות הפקודה "def"
3. כיצד ניגשים למשתנה גלובלי ?
4. קינון (Nested) פונקציות
5. סוגי משתנים וערכים מוחזרים
6. מתי יש להשתמש בכל אחד מהם
7. קלט מהמקלדת
8. עבודה עם עיבוד מידע : map , reduce , lambda , filter , zip
9. עבודה עם מחרוזות מפורמטות
10. פונקציית מחוללת אקראיות – random
11. העברת פרמטרים לפונקציה, תחביר הפונקציות :

- func(name)
- func(name = value)
- func(*name)
- func(**name)

פרק 7 : פונקציות

פונקציה היא בלוק של קוד (בלוק של פקודות) הפועל רק כאשר קוראים לו . הפונקציה יכולה לקבל נתונים, לעבד אותם ולהחזיר את תוצאות העיבוד (נקרא לזה "ערך מוחזר") לשורה שקראה לפונקציה.

בעבודה עם פונקציה יש את השלבים הבאים :

* יצירה/הגדרה של הפונקציה – כתיבת בלוק הקוד של הפונקציה כולל הכנה לקבלת נתונים הנשלחים אליה ועיבוד הנתונים. הנתונים המתקבלים בפונקציה נקראים **פרמטרים**.

* קריאה לפונקציה משורה מסוימת בתוכנית עם אפשרות לשלוח לה נתונים הנקראים **ארגומנטים**.

* בשורה הקוראת יש לעשות "הכנה" לקבלת ערך חוזר מהפונקציה שהוא תוצאה של עיבוד הנתונים שנשלח אל הפונקציה.

בהמשך נסביר את השימוש בפונקציות והייתרון של המודולריות והאחזקה הקלה יותר של תוכניות עם פונקציות.

7.1 יצירה/הגדרה של פונקציה

פונקציה בפייתון מוגדרת בעזרת מילת המפתח def . פונקציה גם נקראת פעולה.

תחביר:

def שם הפונקציה ( ) : פרמטרים שהפונקציה מקבלת (אופציונלי)

משפטי הפונקציה

return ((ערך מוחזר (אופציונלי

def – מילה שמורה המתארת הגדרה של פונקציה.

שם הפונקציה – שם כלשהו אבל בדומה לחוקיות כמו למשתנה. כדאי לתת שם המסביר מה הפונקציה עושה.

פרמטרים שהפונקציה מקבלת – אופציה. אם נשלחים לפונקציה ארגומנטים הם נכנסים לפרמטרים שרושמים בין הסוגריים .

משפטי הפונקציה – המשפטים של הפונקציה כאשר הם באיידנטציה (הזחה) יחסית לשורה הראשונה של ההגדרה.

Ret – אופציונלי . ערך חוזר אל השורה הקוראת.

דוגמה 1 : הגדרת פונקציה שלא מקבלת ערכים ולא מחזירה ערכים.

```
def func1( )
    print("a print in func1")
```

התוכנית הראשית בפייתון נקראת **main( ) .** בפונקציית ה main( ) קוראים (ניתן לומר גם **מזמנים**) פונקציות אחרות.

הקריאה לפונקציה : השורה הקוראת לפונקציה יכולה להיות בתוכנית ה main או בכל פונקציה שהיא והיא תהיה עם שם הפונקציה ולאחריה סוגריים קטנים. בדוגמה כאן : func1( ) .

כל הפונקציות חייבות להיות מוגדרות לפני הפונקציה הראשית main . סדר הפונקציות לפני main איננו חשוב. כל פונקציה יכולה לזמן פונקציות אחרות אבל פונקציה לא יכולה לזמן את main .

7.2 ארגומנטים

ניתן להעביר נתונים אל פונקציה והם נקראים ארגומנטים. הארגומנטים מצוינים לאחר שם הפונקציה, בתוך הסוגריים. יש אפשרות להוסיף ארגומנטים רבים ככל שנרצה ונפריד ביניהם באמצעות פסיק.

ארגומנטים נקראים לפעמים בקיצור בשם args .

דוגמה : נגדיר פונקציה עם 2 ארגומנטים אחד הוא fname – שם פרטי - והשני הוא lname – שם המשפחה. כאשר קוראים לפונקציה נעביר לה את השם הפרטי ושם המשפחה . הפונקציה תדפיס את השם המלא.

```
def func2(fname,lname) :                הגדרה של פונקציה המקבלת 2 ערכים ולא מחזירה ערך #
    print (fname + ' ' + lname)
```

השורה הקוראת :

```
func2("Avraham","Avinoo")
```

ההדפסה שנקבל : Avraham Avinoo

7.3 ארגומנטים ופרמטרים

המושגים ארגומנטים ופרמטרים משמשים עבור המידע שמועבר אל הפונקציה. ההבדל ביניהם הוא :

הפרמטרים הם המשתנים הרשומים בסוגריים בהגדרת הפונקציה.

הארגומנטים הם הערכים הנשלחים מהשורה הקוראת אל הפונקציה.

בדוגמה הקודמת fname ו lname הם הפרמטרים של הפונקציה. "Avraham" ו "Avinoo" הם הארגומנטים הנשלחים אל הפונקציה.

כמות הארגומנטים הנשלחים אל הפונקציה חייבת להיות שווה לכמות הפרמטרים שהפונקציה מקבלת !! אם כמות הארגומנטים איננה שווה לכמות הפרמטרים נקבל הודעת שגיאה.

דוגמה : הודעת שגיאה שנקבל אם שולחים ארגומנט אחד אבל בפונקציה הגדרנו 2 פרמטרים :

```
def func2(fname,lname) :
    print (fname + ' ' + lname)
```

הקריאה לפונקציה :

```
func2("Avraham")
```

הודעת השגיאה שנקבל :

```
Traceback (most recent call last):
  File "./prog.py", line 4, in <module>
TypeError: func2() missing 1 required positional argument: 'lname'
```

השגיאה היא שפונקציה 2 חסרה ארגומנט 1 הנדרש במיקום lname .

7.4 הגדרת פונקציה כאשר לא יודעים כמה ארגומנטים מועברים אל הפונקציה (*args)

בסעיפים הקודמים ראינו שכמות הארגומנטים ששולחים לפונקציה חייבת להיות שווה לכמות הפרמטרים שהפונקציה מקבלת. נראה דוגמה :

```
def add(a,b,c) :                    # הגדרת פונקציה המקבלת 3 פרמטרים
    print("The sum is : ",a+b+c)

add(5,6,10)                         # קריאה לפונקציה ושליחה של 3 ארגומנטים לפונקציה
```

וההדפסה שנקבל : The sum is : 21

אם היינו קוראים לפונקציה add ושולחים לה יותר מ 3 פרמטרים כמו לדוגמה : add(5,6,7,8,10) היינו מקבלים הודעת שגיאה :

TypeError : add( ) takes 3 positional arguments but 5 were given כלומר שבפונקציה add( ) יש מיקום ל 3 ארגומנטים אבל ניתנו 5 .

בשפת פייתון אנחנו יכולים להעביר מספר ארגומנטים לא קבוע באמצעות סמלים מיוחדים . קיימים 2 סמלים מיוחדים :

1. *** args** ארגומנטים שאינם מילות מפתח 2. ****kwargs** - ארגומנטים של מילות מפתח.

משתמשים ב 2 הסמלים המיוחדים האלו כאשר אנחנו לא בטוחים לגבי מספר הארגומנטים שיש להעביר אל הפונקציה.
בדרך זו הפונקציה תקבל tuple של ארגומנטים, ויכולה לגשת לפריטים בהתאמה:

דוגמה 1 לשימוש ב *args .

```
def func3_add(*num):
   sum = 0
   for n in num:
      sum = sum + n
   print("The sum is : ",sum)
```

נבצע מספר קריאות לפונקציה ונשלח אליה בכל פעם כמות שונה של ארגומנטים

```
func3_add(3,4)
func3_add(3,4,5)
func3_ add(3,4,5,6)
```

ההדפסות שנקבל :

```
The sum is :  7
The sum is :  12
The sum is :  18
```

דוגמה 2 לקליטת כמות ארגומנטים שלא ידועה והדפסת האיבר האחרון .

```
def func3(*brothers):
        for item in brothers:
              print("The brothers are : " + item)


func3("Reuven")
func3("Reuven", "Shimon", "Levi")
func3("Reuven", "Shimon", "Levi", "Josef", "Benjamin")
```

ההדפסות שנקבל :

```
The brothers are : Reuven
The brothers are : Reuven
The brothers are : Shimon
The brothers are : Levi
The brothers are : Reuven
The brothers are : Shimon
```

The brothers are : Levi

The brothers are : Josef

The brothers are : Benjamin

דוגמה 3 : קליטת 2 ארגומנטים אחד ידוע והשני איננו ידוע:

```
def func4(arg1, *arg2):
   print ("First argument :", arg1)
   for arg in arg2:
      print("Next argument through *arg2:", arg)

func4('Hello', 'Welcome', 'to', 'Class')
```

ההדפסות שנקבל :

First argument : Hello

Next argument through *arg2 : Welcome

Next argument through *arg2 : to

Next argument through *arg2 : Class

7.5 ארגומנטים של מילות מפתח

ניתן לשלוח ארגומנטים עם התחביר *key = value* syntax . בצורה זו סדר הארגומנטים לא משנה.

דוגמה :

```
def func5(child3, child2, child1,child5, child4):  # סדר הפרמטרים הוא לא סדר הארגומנטים בשורה הקוראת
  print("The youngest child is " + child5)
```

השורה הקוראת :

```
Func5(child1 = "Reuven", child2 = "Shimon", child3 = "Levi", child4="Josef", child5="Benjamin")
```

ההדפסה שנקבל : The youngest child is Benjamin

הערה : צירוף הארגומנטים של מילות המפתח נרשם בקצרה kwargs .

7.6 ארגומנטים שרירותיים של מילות מפתח **kwargs

אם אנחנו לא יודעים כמה ארגומנטים של **מילות מפתח** יועברו לפונקציה ניתן להוסיף שתי כוכביות ** לפני שם הפרמטר בהגדרת הפונקציה. בדרך זו הפונקציה תקבל מילון - dictionary - של ארגומנטים, ויכולה לגשת לפריטים בהתאמה .

דוגמה 1: הוספת 2 כוכביות ** לפרמטר כאשר מספר הארגומנטים של מילות המפתח אינו ידוע.

```
def func6(**kid):
```

```
    print("His last name is " + kid["lname"])
```

הקריאה לפונקציה:

```
Func6(fname = "Avraham", lname = "Avinoo")
```

ההדפסה : His last name is Avinoo

דוגמה 2:

```
def func7(**data):
    for key, value in data.items():
        print("{} is {}".format(key,value))
    print()

func7(Firstname="Avraam", Lastname="Avinoo", Age=5222, Phone=1234567890)
func7(Firstname="Itzhak", Lastname="Avinoo", Email="Itzhak@ganEden.com", Country="Eden
Garden", Age=5203, Phone=9876543210)
```

ההדפסות שנקבל:

```
Firstname is Avraam
Lastname is Avinoo
Age is 5222
Phone is 1234567890

Firstname is Itzhak
Lastname is Avinoo
Email is Itzhak@ganEden.com
Country is Eden Garden
Age is 5203
Phone is 9876543210
```

דוגמה 3 :

```
def func8(**kwargs):
        for key, value in kwargs.items():
                print ("%s == %s" %(key, value))
```

הקריאה לפונקציה :

```
func8(first ='Hello', mid ='my', last='Class')
```

ההדפסה שנקבל :

```
last == Class
mid == my
first == Hello
```

7.7 ברירת מחדל של ערך פרמטר

אם נקרא לפונקציה ולא נשלח לו ארגומנט אז הפונקציה תשתמש בערך ברירת המחדל.

דוגמה : כיצד להשתמש בערך פרמטר המהווה ברירת מחדל :

```
def func6(brother = "Levi"):          # אם הפונקציה לא תקבל ארגומנט אז יירשם "Levi"
  print("My brother is " + brother)
```

הקריאה לפונקציה :

```
func6("Reuven")
func6("Shimon")
func6()
func6("Josef")
```

ההדפסה שנקבל :

```
My brother is Reuven
My brother is Shimon
My brother is Levi
My brother is Josef
```

7.8 העברה של list כארגומנט

ניתן לשלוח כארגומנט לפונקציה את כל סוגי הנתונים (מחרוזת, מספר, רשימה, מילון וכו ') והארגומנט יטופל כסוג נתונים זהה בתוך הפונקציה. למשל, אם נשלח ארגומנט כמו רשימה הוא יהיה רשימה ויטופל כרשימה בפונקציה.

דוגמה :

```
def func7(food):
  for x in food:
    print(x)
fruits = ["apple", "cherry", "lemon", "banana"]   # הגדרה של רשימה
```

הקריאה לפונקציה :

```
func7(fruits)
```

ההדפסה שנקבל :

apple
cherry
lemon
banana

7.9 ערכים מוחזרים – return values

כדי להחזיר ערך מפונקציה משתמשים בהצהרה return .

דוגמה : רשום שורה שתקרא לפונקציה המחזירה את הריבוע של המספר ששולחים אליה.

```
def func8(x):
  return x * x
```

השורות הקוראות לפונקציה

```
print(func8(3))
print(func8(5))
print(func8(9))
```

ההדפסה שנקבל :

```
9
25
81
```

7.9.1 החזרה של יותר מערך אחד

בפייתון ניתן להחזיר יותר מערך אחד.

דוגמה:

```
def return2values(x,y):
    return x*y , x/y              #  החזרת 2 ערכים
```

הקריאה לפונקציה

```
multiply,divide = return2values(10,2)
print(multiply,divide)
```

ההדפסה שנקבל : 20 5.0

אם פונקציה איננה מחזירה ערך אז הערך החוזר ממנה הוא None .

דוגמה: נגדיר פונקציה stam( ) שמחברת סתם 2 מספרים **:**

```
>>> def stam( ):
 5+5
```

נבקש להדפיס את הערך החוזר מהפונקציה:

```
>>> print(stam())
None                                    # None  נקבל את ההדפסה
```

7.10 ההצהרה pass

הגדרות פונקציה אינן יכולות להיות ריקות אבל אם מסיבה כלשהי אנחנו מגדירים פונקציה ללא תוכן יש לרשום הכנס את משפט ה pass כדי להימנע מקבלת שגיאה.

דוגמה :

```
def func9( ):
  pass
```

אם היינו מגדירים פונקציה ללא משפטים בפונקציה נקבל הודעת שגיאה.

דוגמה : קבלת הודעת שגיאה בפונקציה ללא משפטים וללא pass .

```
def func10():
```

הודעות השגיאה שנקבל :

```
Traceback (most recent call last):
  File "/usr/lib/python3.8/py_compile.py", line 144, in compile
    code = loader.source_to_code(source_bytes, dfile or file,
  File "<frozen importlib._bootstrap_external>", line 846, in source_to_code
  File "<frozen importlib._bootstrap>", line 219, in _call_with_frames_removed
  File "./prog.py", line 2

    ^
SyntaxError: unexpected EOF while parsing

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "<string>", line 1, in <module>
  File "/usr/lib/python3.8/py_compile.py", line 150, in compile
    raise py_exc
py_compile.PyCompileError:   File "./prog.py", line 2

    ^
SyntaxError: unexpected EOF while parsing
```

7.11 משתנים לוקאליים וגלובאליים

ניתן להשתמש במשתנה בתוך האזור שבו הוא נוצר. מצב זה נקרא **scope** ובעברית טווח הכרה.

7.11.א טווח הכרה לוקאלי (מקומי, פנימי, אוטומטי) – Local Scope

משתנה הנוצר בתוך פונקציה שייך לטווח ההכרה הלוקלי (שנקרא גם מקומי או פנימי או אוטומטי) וניתן להשתמש בו רק בתוך הפונקציה.

דוגמה :

```
def myfunc( ):
    x = 30
    print(x)
```

x הוא משתנה לוקלי וההדפסה שנקבל היא 30 .

7.11.ב טווח הכרה גלובאלי – Global Scope

משתנה הנוצר בגוף הראשי של קוד הפייתון הוא משתנה גלובאלי והוא שייך לטווח ההכרה הגלובאלי.

משתנים שנוצרים מחוץ לפונקציה הם גלובאליים וניתן להשתמש בהם מהשורה שבה הם נוצרו (הוגדרו) בהמשך כל השורות בתוכנית ועל ידי כל אחד.

דוגמה :

```
x=30
def func1( ) :
   print(x)
```

גם כאן נקבל הדפסה של 30 כי x הוגדר כגלובלי ולכן הוא מוכר בכל התוכנית.

יש לזכור שאם היינו רושמים :

```
print (y)
y=40
```

נקבל את הודעת השגיאה :

```
Traceback (most recent call last):
  File "./prog.py", line 10, in <module>
NameError: name 'y' is not defined
```

הסיבה לכך היא שביקשנו הדפסה של המשתנה y אבל הוא נוצר מאוחר יותר.

7.11.ג מתן שמות למשתנים

אם יש משתנה גלובאלי בשם מסוים ויש משתנה לוקאלי באותו השם בתוך הפונקציה אז בתוך הפונקציה משתמשים במשתנה הלוקאלי ומחוץ לפונקציה במשתנה הגלובאלי.

דוגמה:

```
x=10
def func2( ):
   x=30
   print("x =  ",x)

print("x = ",x)
```

נקבל כאן 2 הדפסות : בפונקציה נקבל x = 30 כי זה ערכו של המשתנה הלוקאלי ובהדפסה האחרונה נקבל x = 10 כי מודפס הערך של המשתנה הגלובאלי.

7.11.ד מילת המפתח global

כאשר רוצים להשתמש במשתנה שהוגדר כגלובאלי בתוך פונקציה ניתן להצהיר עליו כגלובאלי בתוך הפונקציה . ניתן להשתמש בו בפונקציה אבל יש לזכור שאם משנים את ערכו בפונקציה הוא מקבל את הערך החדש שקיבל בפונקציה גם בהמשך התוכנית.

דוגמה :

```
x=50
def myfunc():
  global x
x = 30

myfunc()
print(x)
```

בשורה הראשונה הגדרנו משתנה גלובלי בשם x והתחלנו אותו בערך 50 . בפונקציה הצהרנו שמדברים על ה x הגלובלי ושינינו את ערכו ל 30 . בהמשך קראנו לפונקציה והרצנו אותה ולאחר מכן ביקשנו להדפיס את x . נקבל בהדפסה את הערך החדש מהפונקציה, כלומר 30 .

אם מוגדר משתנה גלובלי ורוצים להשתמש בו בפונקציה ללא הגדרת global - ניתן להשתמש בו אבל לא לשנות את ערכו.

דוגמה 1:

נגדיר 2 משתנים גלובליים x ו y

```
x = 10
y = 20
```

נגדיר פונקציה שמחברת את 2 המשתנים הגלובליים :

```
def add():
   z = x + y
```

```
    print(z)
```

הקריאה לפונקציה:

```
add()
```

ההדפסה שנקבל : 30 .

אם ננסה לשנות את הערך של אחד המשתנים הגלובליים שלא הוגדר בפונקציה כ global נקבל הודעת שגיאה.

דוגמה 2:

```
x = 10
y = 20
```

נגדיר פונקציה שמחברת ערך למשתנה גלובלי שלא הוגדר בפונקציה עם המילה global :

```
def add():
    x = x + 5
    print(z)
```

הקריאה לפונקציה:

```
add()
```

נקבל את הודעת השגיאה : UnboundLocalError: local variable 'x' referenced before assignment

אם נרצה לשנות את ערכו של x נרשום אותו בפונקציה כ global ונוכל לשנות את ערכו .

דוגמה 3:

```
x = 15
def change():
    global x
    x = x + 5
    print("Value of x inside a function :", x)
change()
print("Value of x outside a function :", x)
```

ההדפסות שנקבל :

```
Value of x inside a function : 20
Value of x outside a function : 20
```

7.11.ה משתנים גלובליים במודולים של פייתון

הדרך הטובה ביותר לשתף באותה התוכנית משתנים גלובליים במודולים שונים היא ליצור מודול תצורה שנקרא config.py או cfg.py ולייבא (import) אותו בכל המודולים שבתוכנית. לאחר מכן המודול נעשה נגיש (ניתן לגשת אליו) כשם של משתנה גלובלי. קיים instance (מופע) אחד בלבד של כל מודול ולכן כל שינוי שמבוצע באובייקט המודול משתקפים בכל המקומות.

דוגמה לשיתוף משתנים גלובליים בין מודולים.

א. ניצור קובץ **config.py** שבו נשמור 3 משתנים שיהיו משתנים גלובליים בכל התוכנית.

```
x = 0
y = 0
z = "Hello"
```

ב. ניצור קובץ **modify.py** לשינוי המשתנים הגלובליים.

```
import config
config.x = 1
config.y = 2
config.z = "Hello Friends"
```

ג. ניצור קובץ תוכנית main.py שבו נוכל לעבוד עם המשתנים הגלובליים.

```
import config
import modify
print(config.x)
print(config.y)
print(config.z)
```

ונקבל את ההדפסות:

```
1
2
Hello Friends
```

7.12 הפונקציה lambda

הפונקציה lambda היא פונקציה קטנה ללא שם.

פונקציית lambda יכולה לקבל כל מספר של ארגומנטים אבל יש לה הוראה אחת בלבד.

התחביר :

lambda ארגומנטים : ביטוי

דוגמה 1: נגדיר פונקציית lambda המחזירה את השורש של המספר ששולחים לה :

```
root = lambda a : a ** 0.5
print(root(25))
```

ההדפסה שנקבל : 5.0

לפונקציית lambda אפשר לשלוח כמה ארגומנטים שנרצה.

דוגמה 2: שליחת 2 ארגומנטים. נרצה לבצע את הפעולה a^b .

```
square = lambda a, b : a ** b
print(square(2, 6))
```

דוגמה 3: שליחה של 3 ארגומנטים : רוצים לשלוח את המקדמים a b c של משוואה ריבועית $ax^2+bx+c=0$ ולבדוק מהי הדלתה delta = b^2-4ac כדי לדעת האם יש שורשים למשוואה וכמה. נניח שהמשוואה היא $2x^2+5x-6=0$

```
delta = lambda a,b,c : b**2 – 4*a*c
delt=delta(2,5,-6)
if(delt>0):
   print("2 roots")
```

ההדפסה שנקבל : 2 roots כי הדלתה חיובית ושווה 73 .

7.13 השימוש בפונקציה ( ) id לדעת כתובות של אובייקטים

הפונקציה ( )id מקבלת אובייקט (משתנה או פונקציה) ומחזירה את הכתובת שלו בזיכרון.

דוגמה:

```
>>> x=5
>>> print("The address of x is : ",id(x))
The address of x is :  2173039176112
```

גם לפונקציה יש כתובת שבה היא נמצאת :

דוגמה :

```
>>> def func1():
print("The address of function func1 is : ",id(func1))


>>> func1()
The address of function func1 is :  2173080242352
>>>
```

7.14 כיצד מועברים ארגומנטים אל הפרמטרים בפונקציה ?

כאשר שולחים ארגומנטים אל פונקציה הם נכנסים אל הפרמטרים שבסוגריים הקטנים בהגדרת הפונקציה. שולחים אל הפונקציה את הכתובת של כל אחד מהארגומנטים בשיטה הנקראת **call by reference** (כמו בשפת C). בשפות אחרות כמו שפת C יש גם שיטה נוספת הנקראת **call by value** ששולחים ערך ולא כתובת . בשיטת call by value נוצר בפונקציה משתנה נוסף שמקבל את ערך הארגומנט שנשלח . אם בפונקציה משנים את הערך של המשתנה החדש המשתנה מהשורה הקוראת לא משנה את ערכו. כאשר שולחים בשיטת call by reference , הארגומנט הנשלח הוא כתובת ואז עובדים עם הכתובת שנשלחה . להבדיל משפה כמו C שבה שינוי ערכו של המשתנה בפונקציה יגרום לשינוי המשתנה בשורה המקורית, בשפת פייתון שינוי ערכו של המשתנה ייצור אובייקט חדש בכתובת חדשה (בתנאי שהוא immutable – לא ניתן לשינוי) ולא ישנה את הערך המקורי .

דוגמה :

```
>>> x1=5
>>> print("The address of x1 is : ",id(x))
```

```
The address of x1 is :  2173039176112                    # x1 הכתובת בזיכרון של
>>> def stam(x2):                                        # x2 כפרמטר stam המקבלת את הארגומנט הגדרה של פונקציה בשם
  print("x2 = ", x2,"and the address of x2 is : ",id(x2))    # x2 הוראה להדפיס את הערך והכתובת של
  x2+=1                                                  # 1 ב x2 הגדלת ערכו של
  print("x2 = ", x2,"and the address of x2 is : ",id(x2))  # x2 הוראה להדפיס שוב את הערך והכתובת של

  >>> stam(x1)                                          #( x1 (למעשה את הכתובת של x1 ושליחת הארגומנט stam קריאה לפונקציה
  x2 = 5 and the address of x2 is :  2173039176112       # x1 ל x2 ששווים הדפסת הערך והכתובת של
  x2 = 6 and the new address of x2 after adding 1 is : 2173039176144  #1 אחרי ההגדלה ב x2 הדפסת הערך והכתובת של
  >>> print("The address of x1 is : ",id(x1),"and his value is : ",x1)    # x1 המקורי בדיקה מה קורה עם
  The address of x1 is :  2173039176112 and his value is :  5          # 5 רואים ש x1 לא שינה את ערכו המקורי שהיה
```

גם אם נשלח ערך קבוע x2 ישנה את כתובתו כאשר נשנה את ערכו .

דוגמה:

```
>>> stam(5)
x2 =  5 and the address of x2 is :  2173039176112
x2 =  6 and the new address of x2 after adding 1 is :  2173039176144
>>>
```

7.15 שימוש במילה is להשוואת id

אם יש לשני אובייקטים אותו id זה אומר שהם מצביעים על אותה כתובת בזיכרון.

דוגמה: נגדיר 2 משתנים מטיפוס שלם (x1 ו x2) וניתן להם את אותו הערך.

```
>>> x1=5
>>> x2=5
```

נבדוק את הכתובת של כל אחד מהמשתנים:

```
>>> id(x1)
2173039176112
>>> id(x2)
2173039176112
```

רואים שהיות ויש להם את אותו האובייקט (הערך 2) אז יש להם את אותה הכתובת.

כדי לבדוק האם לשני אובייקטים יש את אותה הכתובת משתמשים במילה **is** .

דוגמה : נבדוק האם ל x1 ו x2 אותה הכתובת :

```
>>> x1 is x2
True
```

7.16 רקורסיה

רקורסיה היא מושג מתמטי ותכנותי נפוצים. רקורסיה קורית כאשר מגדירים פונקציה ובאחד ממשפטי הפונקציה קוראים לפונקציה עצמה . היתרון של רקורסיה הוא שניתן לעבור בלולאה על נתונים כדי להגיע לתוצאה.

יש לעבוד בזהירות רבה מאוד עם רקורסיה כי טעות קטנה יכולה לגרום לכניסה ללולאה אין סופית או שהמחשב ישתמש בכמות זיכרון עודפת וכוח מעבד מיותר. עם זאת, כאשר נכתוב נכון רקורסיה נוכל לקבל גישה יעילה מאוד מתמטית אלגנטית לתכנות.

בדוגמה שנראה מיד נגדיר פונקציה בשם my_recursion( ) . באחד ממשפטי הפונקציה נקרא ל my_recursion( ) פעם נוספת. נשתמש במשתנה בשם k כנתון אשר מורידים ממנו (1-) בכל לולאה של רקורסיה. הרקורסיה מסתיימת כאשר k מגיע ל 0 .

דוגמה 1: נרשום רקורסיה המחברת את המספרים מ 1 עד 5 . הרקורסיה תתבצע 5 פעמים כאשר בכל פעם נחבר את הערך ב k עם הערך של חיבור המספרים הקודמים שהיו ב k. נכנסים לרקורסיה כאשר k=5 ויוצאים ממנה רק כאשר k לא גדול מ 0 (כלומר k=0).

```
def my_recursion(k):       #   k הגדרה של פונקציה המקבלת את הערך 5 למשתנה
  if(k > 0):               #  k>0 כניסה רק כאשר
    result = k + my_recursion(k - 1)  # k-1 התוצאה result שווה לערך של k פלוס הערך החוזר מהפונקציה my_recursion ושולחים לה את
    print(result)
  else:
    result = 0
  return result
```

השורות שבו מדפיסים את תוצאות הרקורסיה

```
print("\n\nRecursion Example Results")
my_recursion(5)            # קריאה לפונקציה ושליחת הערך 5 אליה
```

ההדפסות שנקבל :

```
Recursion Example Results
1
3
6
10
15
```

דוגמה 2 : מציאת העצרת factorial של מספר :

```
def factorial(n):
  if n == 0:
    return 1
  else:
    return n * factorial(n - 1)
print(factorial(5))
```

ההדפסה שנקבל : 120

7.17 תרגילים

1 . יש לרשום תוכנית ראשית ופונקציה. בתוכנית הראשית קולטים 2 מספרים ופעולה (חיבור או חיסור או כפל או שארית). התוכנית הראשית תשלח לפונקציה את 2 המספרים והפעולה. בפונקציה נדפיס את תוצאת הפעולה.

2. מדוע בהרצת התוכנית הבאה נקבל את ההודעה :UnboundLocalError: local variable 'x' referenced before assignment

```
x = "Hi"
def func1( ):
    x=x*2
    print(x)
func1( )
```

3. מדוע קיבלנו את ההודעה : NameError: name 'y' is not defined בהרצת התוכנית הבאה:

```
def func( ):
    y="Hi"
func ( )
print(y)
```

4. יש לכתוב תוכנית המחשבת שכר נטו של 10 עובדים. התוכנית תהיה מורכבת מתוכנית ראשית הקולטת שכר של עובד וקוראת לפונקציה המחזירה את שכר הנטו על פי מס פרוגרסיבי לפי המפתח הבא :

עד 2000 ₪ - אין הורדת מס. עד 4000 ₪ - 10% מס. עד 5000 ₪ – 20% מס. עד 6000 ₪ - 30% מס, עד 7000 ₪ - 40% . מעל זה 50% מס .

5. כתוב תוכנית הקולטת מספר שלם וקוראת לפונקציה המדפיסה את כל המספרים הראשוניים עד למספר הנקלט. מספר ראשוני הוא מספר המתחלק רק ב 1 ובעצמו (לדוגמא : 1,2,3,5,7,11)

6. כתוב תוכנית הקולטת 3 מספרים מהמשתמש. התוכנית תקרא לפונקציה הבודקת האם 3 המספרים שווים ותחזיר ערך 0 אם השלושה לא שווים. התוכנית תוציא הדפסה מתאימה.

7. כתוב תוכנית שתקלוט 3 מספרים ותקרא לפונקציה שתבדוק מי המספר הגדול ומי הקטן.

הפונקציה תחזיר את הערכים הבאים :

123 אם c>b>a או 132 אם b>c>a או 231 אם a>c>b או 312 אם b>a>c או 213 אם c>a>b או 321 אם a>b>c .

8. יש לרשום תוכנית המבצעת את הדברים הבאים: א. מבקשת מהמשתמש להכניס שני מספרים שלמים. ב. מוציאה הדפסה של מסך תפריט הכולל חיבור, חיסור, כפל וחילוק. ג. עבור כל בחירה שהמשתמש יקיש יש לבצע את הפעולה המתבקשת ולהוציא הדפסה של התוצאה. התוכנית תכיל את הפונקציות הבאות: 1. פונקציית קלט של 2 מספרים. 2 . פונקציית הדפסת תפריט. 3. פונקציית קליטת מקש לבחירת הפעולה הרצויה. 4. פונקציית חיבור, 5. פונקציית חיסור, 6. פונקציית כפל 7. פונקציית חילוק.

9. יש לכתוב תוכנית המקבלת כקלט מטריצה של 10*10 . התוכנית תקרא לפונקציה אשר תאתחל את המערך במספרים רנדומליים מ 0 עד 10. לאחר מכן תקרא לפונקציה שתציג את מסך התפריט הבא :

1. איפוס שורה במטריצה
2. חישוב ממוצע של שורה במטריצה // המשתמש יכניס מאיזו שורה עד איזו שורה
3. חישוב ממוצע של שורות במטריצה
4. איפוס עמודה במטריצה
5. חישוב ממוצע של עמודה במטריצה
6. חישוב ממוצע של עמודות
7. הדפסת המטריצה

יש לחלק את התוכנית לפונקציות המבצעות את הנאמר בתפריט.

הערה : יש לרשום את השורה import random ולהיעזר במתודה random (1,50) שמכניסה מספרים אקראיים בין 1 ל 50 .

10. יש לכתוב פונקציה המקבלת את מספר היום בשנה ומחזירה את החודש והיום המתאימים. הנח שכמות הימים בחודש היא : 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 החל מינואר ועד דצמבר בהתאמה.

11. יש לרשום פונקציה שתקבל רשימה ותדפיס את כל הפריטים בעלי ערך זוגי.

12. מספר מושלם - perfect number - הוא מספר חיובי שלם השווה לסכום המחלקים החיוביים שלו ללא שארית (לא כולל המספר עצמו). לדוגמה המספר 6 מתחלק ב 1,2,3 ו 6 = 1+2+3 . גם המספר 28 הוא מושלם כי 28 מתחלק ב 1,2,4,7,14 ו 28= 1+2+4+7+14 . יש לרשום את התוכנית הבאה, להסביר כל שורה ולרשום מה 2 המספרים הבאים :

```
def perfect_number(n):
    sum = 0
    for y in range(1, n):
        if n % y == 0:
                sum += y
        return sum == n
    for x in range (10000):
        if(perfect_number(x)):
            print (x)
```

13. יש לרשום פונקציה המקבלת מחרוזת ובודקת האם המחרוזת היא פלינדרום . (פלינדרום היא מחרוזת שאם קוראים אותה מימין לשמאל או משמאל לימין מקבלים אותה מחרוזת (לדוגמה abba)) .

14. יש לרשום פונקציה המקבלת מחרוזת של משפטים , בודקת ומדפיסה האם במחרוזת יש לפחות 15 תווים שונים .

15. יש לרשום פונקציה המקבלת מספר ומדפיסה רשימה של ריבועי המספרים מ 1 ועד המספר . לדוגמה : המספר הנשלח הוא 5 אז הרשימה שתודפס היא : [1,4,9,16,25] .

16. מה עושה התוכנית הבאה ?

```
def string_reverse(str1):
    reverse_str1 = ''
    index = len(str1)
    while index > 0:
```

```
        reverse_str1 += str1[index-1]
        index = index-1
    return reverse_str1
```

הקריאה לפונקציה :

```
print(string_reverse('1234abcd'))
```

17. יש לרשום פונקציה המקבלת מספר שלם חיובי ועוד ספרה. הפונקציה תחזיר את כמות הפעמים שהספרה נמצאת במספר. לדוגמה :
עבור המספר 2345434567 והספרה 4 יוחזר הערך 3 כי הספרה 4 מופיעה 3 פעמים במספר.

18. מדוע קיבלנו את ההדפסה : UnboundLocalError: local variable 'c' referenced before assignment
עבור התוכנית הבאה :

```
c = 1 # global variable

def add():
   c = c + 2 # increment c by 2
   print(c)

add()
```

19. מה ההדפסה בסיום הקטע הבא ?

```
hezka = lambda x: x*x*x

print(hezka(5))
```

20. יש לרשום תוכנית שתקלוט מהמשתמש מחרוזת . התוכנית תקרא לפונקציה שתספור כמה פעמים מופיעה המילה "hi" ?