

Lab4

Functions and Modules

<https://www.tutorialspoint.com/python>

<http://marcuscode.com/lang/python/functions>

SECTION 1: FUNCTIONS

1. Defining a function

```
def printme( str ) :  
    "This prints a passed string into this function"  
    print (str)  
    return
```

<<Try to understand the function elements>>

Calling function

```
printme("This is first call")  
printme("Again second")
```

```
This is first call  
Again second
```

2. Pass by reference

```
# Function definition is here  
def changeme( mylist ) :  
    "This changes a passed list into this function"  
    print ("Values inside the function: ", mylist)  
  
    mylist[2]=50  
    print ("Values inside the function after change: ", mylist)  
    return
```

```
# Now you can call changeme function  
mylist = [10,20,30]  
changeme( mylist )  
print ("Values outside the function: ", mylist)
```

```
Values inside the function: [10, 20, 30]  
Values inside the function after change: [10, 20, 50]  
Values outside the function: [10, 20, 50]
```

One more example where argument is being passed by reference and the reference is being overwritten inside the called function.

```
# Function definition is here  
def changeme( mylist ) :
```

```

    "This changes a passed list into this function"
    # This would assign new reference in mylist
    mylist = [1,2,3,4]
    print ("Values inside the function: ", mylist)
    return

# Now you can call changeme function
mylist = [10,20,30]
changeme( mylist )
print ("Values outside the function: ", mylist)

```

```

Values inside the function:  [1, 2, 3, 4]
Values outside the function:  [10, 20, 30]

```

Exercise: Let's try to pass on of the primitive datatype (e.g. Integer, Float, String, Boolean) and see that it is passed by reference or pass by value

```

def changeme( myvar ):
    myvar = 100
    print ("Values inside the function: ", myvar)
    return

myvar = 99
changeme( myvar )
print ("Values outside the function: ", myvar)

```

```

Values inside the function:  100
Values outside the function:  99

```

3. Arguments

Required arguments

```

# Function definition is here
def printme( str ):
    "This prints a passed string into this function"
    print (str)
    return

# Now you can call printme function
printme()

```

```
# Function definition is here
def printme( str ):
    print (str)
    return

# Now you can call printme function
printme("sssss")
```

sssss

Keyword arguments

```
# Function definition is here
def printme( str ):
    "This prints a passed string into this function"
    print (str)
    return

# Now you can call printme function
printme( str = "My string")
```

My string

```
# Function definition is here
def printinfo( name, age ):
    "This prints a passed info into this function"
    print ("Name: ", name)
    print ("Age ", age)
    return

# Now you can call printinfo function
printinfo( age = 50, name = "miki" )
```

Name: miki

Age 50

Default arguments

```
# Function definition is here
def printinfo( name, age = 35 ):
    "This prints a passed info into this function"
    print ("Name: ", name)
    print ("Age ", age)
    return

# Now you can call printinfo function
printinfo( age = 50, name = "miki" )
printinfo( name = "miki" )
```

```
Name: miki
Age 50
Name: miki
Age 35
```

```
def show_info(name, salary = 84360, lang = "Python"):
    print('Name: %s' % name)
    print('Salary: %d' % salary)
    print('Language: %s' % lang)
    print()
```

```
# calling function
show_info('Mateo')
show_info('Mateo', 105000)
show_info('Danny', 120000, 'Java')
```

```
Name: Mateo
Salary: 84360
language: Python
```

```
Name: Mateo
Salary: 105000
language: Python
```

```
Name: Danny
Salary: 120000
language: Java
```

```
def create_button(id, color = '#ffffff', text = 'Button', size = 16):
    print('Button ID: %d' % id)
    print('Attributes:')
    print('Color: %s' % color)
    print('Text: %s' % text)
    print('Size: %d px' % size)
    print()
```

```
create_button(10)
create_button(11, color = '#4286f4', text = 'Sign up')
create_button(id = 12, color = '#323f54', size = 24)
create_button(color = '#1cb22b', text = 'Log in', size = 32, id = 13)
```

```
Button: 10
Attributes:
Color: #ffffff
Text: Button
Size: 16
```

```
Button: 11
Attributes:
Color: #4286f4
Text: Sign up
Size: 16
```

```
Button: 12
Attributes:
Color: #323f54
Text: Button
Size: 24
```

```
Button: 13
Attributes:
Color: #1cb22b
Text: Log in
Size: 32
```

Variable-length arguments

```
# Function definition is here
def printinfo( arg1, *vartuple ):
    "This prints a variable passed arguments"
    print ("Output is: ")
    print (arg1)

    for var in vartuple:
        print (var)
    return

# Now you can call printinfo function
printinfo( 10 )
printinfo( 70, 60, 50 )
```

```
Output is:
10
Output is:
70
60
50
```

<<what is the value of arg1 in printinfo(70, 60, 50)>>

70

<<what is the value of *vartuple in printinfo(70, 60, 50)>>

(60, 50)

4. Return statement

```
# Function definition is here
def sum( arg1, arg2 ):
    # Add both the parameters and return them."
    total = arg1 + arg2
    print ("Inside the function : ", total)
    return total

# Now you can call sum function
total = sum( 10, 20 )
print ("Outside the function : ", total )
```

```
Inside the function : 30
Outside the function : 30
```

5. Lambda function

```
# Function definition is here
sum = lambda arg1, arg2: arg1 + arg2

# Now you can call sum as a function
print ("Value of total : ", sum( 10, 20 ))
print ("Value of total : ", sum( 20, 20 ))
```

```
Value of total : 30
Value of total : 40
```

```
f = lambda x: x + 1
print(f(2))
print(f(8))
```

```
3
9
```

```
g = lambda a, b: (a + b) / 2
print(g(3, 5))
print(g(10, 33))
```

```
4.0
21.5
```

Built-in functions for lambda: filter() and map()

```
numbers = [2, 15, 5, 7, 10, 3, 28, 30]
print(list(filter(lambda x: x % 5 == 0, numbers)))
print(list(map(lambda x: x * 2, numbers)))
```

```
[15, 5, 10, 30]
[4, 30, 10, 14, 20, 56, 60]
```

True or False

```
is_even = lambda n : n % 2 == 0
```

```
numbers = [2, 15, 5, 7, 10, 28, 30]
is_even = lambda n : n % 2 == 0
print(list(map(is_even, numbers)))
```

```
[True, False, False, False, True, True, True]
```

Using Lambda with map (changing the sequence with new value)

```
numbers = [1, 2, 3, 4, 5, 7, 8]
doubled = map(lambda n: n * 2, numbers)
print(list(doubled))
```

```
[2, 4, 6, 8, 10, 12, 16]
```

Exercise: Describe the different between filter and map

Filter

Filter function takes 2 arguments, a function and a list. It returns a list of values that the given function returns true upon

Map

Map function takes 2 arguments, a function and a list. It returns a list of values that was passed through the given function

6. Global and local variables

```
total = 0    # This is global variable.
# Function definition is here
def sum( arg1, arg2 ):
    # Add both the parameters and return them."
    total = arg1 + arg2; # Here total is local variable.
    print ("Inside the function local total : ", total)
    return total

# Now you can call sum function
sum( 10, 20 )
print ("Outside the function global total : ", total )
```

```
Inside the function local total : 30
Outside the function global total : 0
```

7. Example

```
def count_vowel(str):
    vowel = 0
    for c in str:
        if c in ('A', 'E', 'I', 'O', 'U', 'a', 'e', 'i', 'o', 'u'):
            vowel = vowel + 1
    return vowel
```

```
print(count_vowel("Hello world, today is a good day"))
```

```
10
```

```
def area(width, height):
    c = width * height
    return c
```

```
print(area(6, 14.7))
print(area(9, 6.4))
```

```
88.19999999999999
```

```
57.6
```


SECTION 3: I/O

1. Open and close file

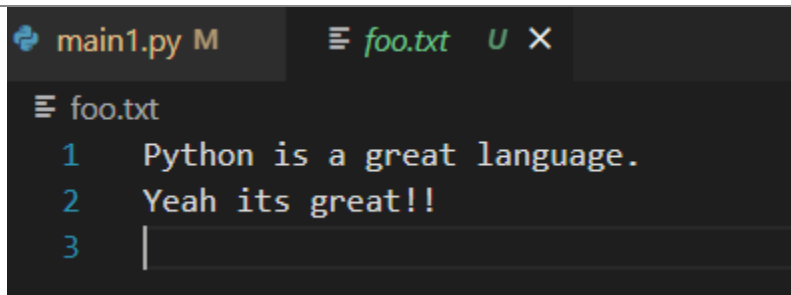
```
# Open a file
fo = open("foo.txt", "wb")
print ("Name of the file: ", fo.name)
print ("Closed or not : ", fo.closed)
print ("Opening mode : ", fo.mode)
fo.close()
```

```
Name of the file: foo.txt
Closed or not : False
Opening mode : wb
```

2. Write a file

```
# Open a file
fo = open("foo.txt", "w")
fo.write( "Python is a great language.\nYeah its great!!\n")

# Close opened file
fo.close()
```

A screenshot of a code editor window. The top bar shows 'main1.py M' and 'foo.txt U X'. The editor content shows the file 'foo.txt' with three lines of text: '1 Python is a great language.', '2 Yeah its great!!', and '3' followed by a cursor. The background is dark with light-colored text.

```
main1.py M    foo.txt U X
foo.txt
1 Python is a great language.
2 Yeah its great!!
3 |
```

3. Read a file

```
# Open a file
fo = open("foo.txt", "r+")
str = fo.read(10)
print ("Read String is : ", str)

# Close opened file
fo.close()
```

```
Read String is : Python is
```

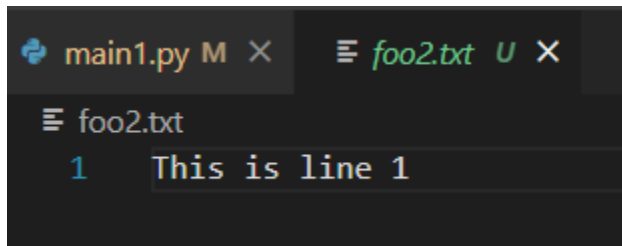
Exercise: Try all the mode to open the file. Use your own example to explain the different of each mode. You must display the result with explanation.

Mode "w"

Open the file in write mode. This mode is for writing only. It creates the file if the file does not already exist and it overwrites the file if the file already exists.

```
f = open("foo2.txt", "w")
f.write("This is line 1")
f.close()
```

Result

A screenshot of a code editor with a dark background. At the top, there are two tabs: 'main1.py M' and 'foo2.txt U'. The 'foo2.txt' tab is active, showing a single line of text: '1 This is line 1'. The line number '1' is on the left, and the text 'This is line 1' is to its right.

Mode "r"

Open the file in read mode. This mode is for reading the content within the file only. This is also the default mode.

```
f = open("foo2.txt", "r")
print(f.read(-1))
f.close()
```

Result


```
This is line 1
```


Mode "a"


Open the file in append mode. It puts the pointer at the end of the file and you will be able to start writing from there. If the file does not exist, it creates the file.

```
f = open("foo2.txt", "a")
f.write("\nAnd this is line 2")
f.close()
```

Result

 main1.py M

 foo2.txt U X

 foo2.txt

1 This is line 1

2 And this is line 2