1

PART: 1
FUNCTIONS

2

FUNCTIONS

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

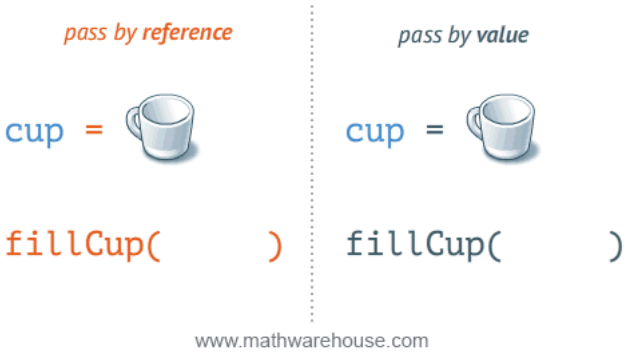
- A function is a block of organized, reusable code used to perform an action
- Simple rules to define a function in Python
  - Function blocks begin with '**def**' followed by the function name and parentheses ()
  - Any input parameter or argument should be placed within these parentheses
  - The first statement of a function can be an optional statement - the documentation string of the function or docstring
  - The code block within every function starts with a colon (:) and is indented
  - The statement return [expression] exits a function, optionally passing back a value to the caller. A return statement with no arguments is the same as return None

https://www.tutorialspoint.com/python/index.htm

3

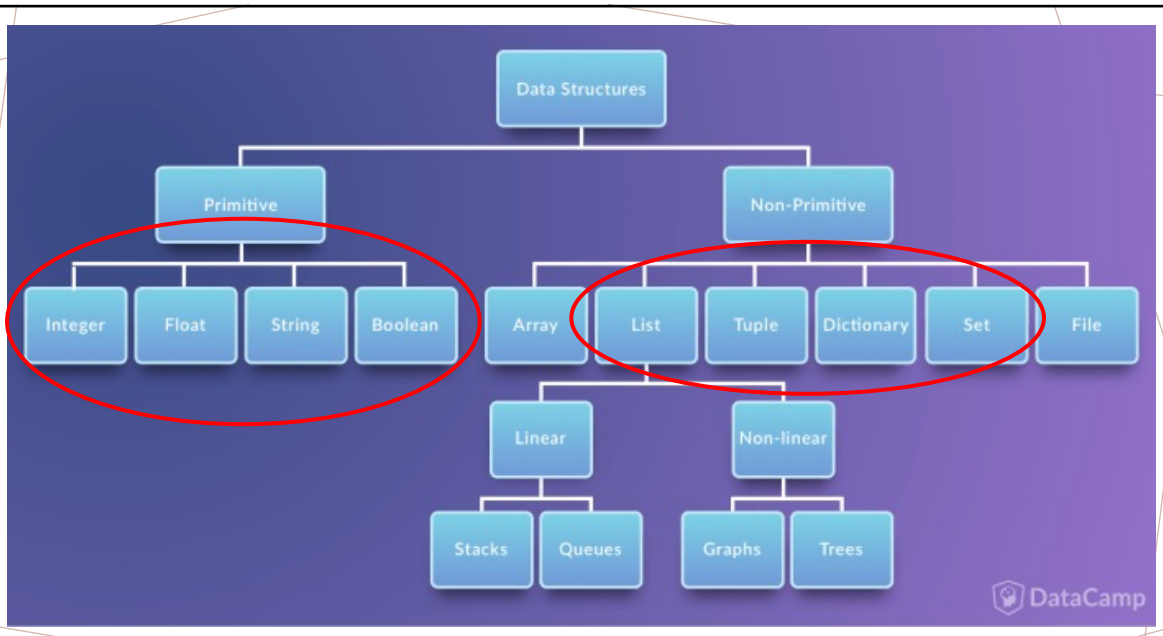
FUNCTIONS

- All parameters (arguments) in the Python language are passed by reference
- Only primitive data type in parameters are passed by value



http://www.mathwarehouse.com/programming/passing-by-value-vs-by-reference-visual-explanation.php

4



https://www.datacamp.com/community/tutorials/data-structures-python

5

FUNCTIONS

- All parameters (arguments) in the Python are passed by reference
- It means if you change what a parameter refers to within a function, the change also reflects back in the calling function

```
# Function definition is here
def changeme( mylist ):
   "This changes a passed list into this function"
   print ("Values inside the function before change: ", 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 before change:  [10, 20, 30]
Values inside the function after change:  [10, 20, 50]
Values outside the function:  [10, 20, 50]
```

https://www.tutorialspoint.com/python/index.htm

6

FUNCTIONS

- Local and Global variables

https://www.tutorialspoint.com/python/index.htm

7

FUNCTIONS

- Where argument is being passed by reference and the reference is being overwritten inside the function
- The parameter **mylist** is local to the function changeme
- Changing mylist within the function does not affect mylist
- The function accomplishes nothing and finally this would produce the following result

```
# Function definition is here
def changeme( mylist ):
   "This changes a passed list into this function"
   mylist = [1,2,3,4] # This would assi new reference in mylist
   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]
```

https://www.tutorialspoint.com/python/index.htm

8

FUNCTIONS

- Function Arguments
  - Required arguments
  - Keyword arguments
  - Default arguments
  - Variable-length arguments

```
def functionname( parameters ):
```

https://www.tutorialspoint.com/python/index.htm

9

FUNCTIONS

- 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()
```

When the above code is executed, it produces the following result −

```
Traceback (most recent call last):
   File "test.py", line 11, in <module>
      printme();
TypeError: printme() takes exactly 1 argument (0 given)
```

https://www.tutorialspoint.com/python/index.htm

10

FUNCTIONS

- Keyword Arguments
  - The caller identifies the arguments by the parameter name
  - This allows you to place them out of order

```
# 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")
```

When the above code is executed, it produces the following result −

```
My string
```

https://www.tutorialspoint.com/python/index.htm

11

FUNCTIONS

- Keyword Arguments
  - The following example gives clearer
  - Note that the order of parameters **does not** matter

```
# 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" )
```

When the above code is executed, it produces the following result −

```
Name:  miki
Age  50
```

https://www.tutorialspoint.com/python/index.htm

12

FUNCTIONS

- Default Arguments
  - It has a default value if a value is not provided.

```
# 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" )
```

When the above code is executed, it produces the following result −

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

https://www.tutorialspoint.com/python/index.htm

13

FUNCTIONS

- Variable-length Arguments
  - You may need a function for more arguments
  - These arguments are called **variable-length** arguments and are not named in the function definition, unlike required and default 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 )
```

When the above code is executed, it produces the following result −

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

https://www.tutorialspoint.com/python/index.htm

14

FUNCTIONS

- Return statement
  - Return [expression] and exit a function, optionally passing back an expression to the caller
  - A return statement with no arguments is the same as return None

```
# 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 )
```

When the above code is executed, it produces the following result −

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

15

SCOPE OF VARIABLES

16

SCOPE OF VARIABLES

- All variables in a program may not be accessible at all locations in that program
- This depends on where you have declared a variable
- The scope of a variable determines the portion of the program where you can access. There are two basic scopes of variables in Python
  - Global variables
  - Local variables

https://www.tutorialspoint.com/python/index.htm

17

SCOPE OF VARIABLES

- Global vs. Local variables
  - Variables that are defined inside a function have a local scope, and those defined outside have a global scope.
  - Local variables can be accessed only inside the function where they are declared
  - Global variables can be accessed throughout the program.

```
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 )
```

When the above code is executed, it produces the following result –

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

https://www.tutorialspoint.com/python/index.htm

18

ANONYMOUS FUNCTIONS

19

ANONYMOUS FUNCTIONS

- They are not declared in the standard manner using the **def** keyword
- You can use the **lambda** keyword to create small anonymous functions
  - Lambda forms can take any number of arguments but return just one value
  - They cannot contain commands or multiple expressions

```
lambda [arg1 [,arg2,.....argn]]:expression
```

https://www.tutorialspoint.com/python/index.htm

20

ANONYMOUS FUNCTIONS

```
lambda [arg1 [,arg2,.....argn]]:expression
```

```
# 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 ))
```

When the above code is executed, it produces the following result −

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

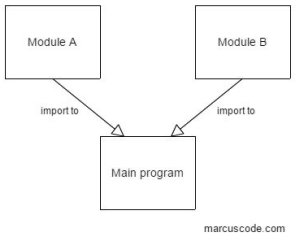
https://www.tutorialspoint.com/python/index.htm

21

SECTION:2 MODULES

22

MODULES



- A module allows you to organize your code.
- Grouping related code into a module makes the code easier to understand and use.
- A module is a Python object with arbitrarily named attributes that you can bind and reference.
- Simply, a module is a file consisting of Python code.
- A module can define **functions**, **classes** and **variables**.

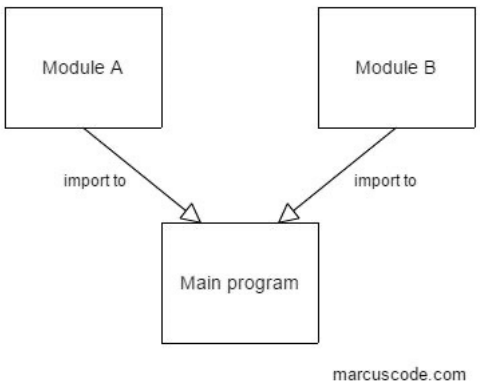
https://www.tutorialspoint.com/python/index.htm

23

MODULES

- Examples
  - Here is an example of a simple module, support.py

```
def print_func( par ):
   print "Hello : ", par
   return
```



https://www.tutorialspoint.com/python/index.htm

24

MODULES

- Examples
  - Before use, you must import the file

```
# Import module support
import support

# Now you can call defined function that module as follows
support.print_func("Zara")
```

https://www.tutorialspoint.com/python/index.htm

25

MODULES

- from…import
  - **from** statement lets you import specific attributes from a module into the current namespace.
  - For example, to import the function fibonacci from the module fib, use the following statement
  - Note: the file name is 'fib.py'

```
# Fibonacci numbers module

def fib(n): # return Fibonacci series up to n
   result = []
   a, b = 0, 1
   while b < n:
      result.append(b)
      a, b = b, a + b
   return result
>>> from fib import fib
>>> fib(100)
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
```

https://www.tutorialspoint.com/python/index.htm

26

MODULES

- The from...import * Statement
  - It is also possible to import all the names from a module into the current namespace by using the following import statement

```
from modname import *
```

https://www.tutorialspoint.com/python/index.htm

27

NAMESPACES AND SCOPING

- For example, we define a variable *Money* in the global namespace.
- Within the function addMoney, we assign Money a value, therefore Python assumes Money as a local variable.
- However, we accessed the value of the local variable *Money* before setting it, so an UnboundLocalError is the result. Uncommenting the global statement fixes the problem.

```
Money = 2000
def AddMoney():
   # Uncomment the following line to fix the code:
   # global Money
   Money = Money + 1

print (Money)
AddMoney()
print (Money)
```

https://www.tutorialspoint.com/python/index.htm

33

MODULES

- See built-in modules

```
# Import built-in module math
import math

content = dir(math)
print (content)
```

When the above code is executed, it produces the following result −

```
['__doc__', '__file__', '__name__', 'acos', 'asin', 'atan',
'atan2', 'ceil', 'cos', 'cosh', 'degrees', 'e', 'exp',
'fabs', 'floor', 'fmod', 'frexp', 'hypot', 'ldexp', 'log',
'log10', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh',
'sqrt', 'tan', 'tanh']
```

https://www.tutorialspoint.com/python/index.htm

34

SECTION:3 I/O

36

I/O

- Printing to the Screen

```
#!/usr/bin/python3

print ("Python is really a great language,", "isn't it?")
```

This produces the following result on your standard screen −

```
Python is really a great language, isn't it?
```

https://www.tutorialspoint.com/python/index.htm

37

I/O

- Reading Keyboard Input

https://www.tutorialspoint.com/python/index.htm

38

I/O

- The open Function
  - Python provides basic functions and methods necessary to manipulate files
  - Using a **file** object.

```
file object = open(file_name [, access_mode][, buffering])
```

  - **file_name** — The name of the file
  - **access_mode** — The mode the file is opened, i.e., read, write, append, etc. The default file access mode is read (r)
  - **buffering** — If the buffering value is set to 0, no buffering takes place. If the buffering value is 1, line buffering is performed while accessing a file. If you specify the buffering value as an integer greater than 1, then buffering action is performed with the indicated buffer size.

https://www.tutorialspoint.com/python/index.htm

39

I/O

Sr.No.	Mode & Description
1	**r** Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.
2	**rb** Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.
3	**r+** Opens a file for both reading and writing. The file pointer placed at the beginning of the file.
4	**rb+** Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.
5	**w** Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.

https://www.tutorialspoint.com/python/index.htm

40

I/O

6	**wb** Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
7	**w+** Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
8	**wb+** Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
9	**a** Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
10	**ab** Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.

https://www.tutorialspoint.com/python/index.htm

41

I/O

11	**a+** Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.
12	**ab+** Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

https://www.tutorialspoint.com/python/index.htm

42

NOTE

- In binary mode, newline characters are not automatically translated, and no encoding or decoding is performed on the file content.
- Binary mode is useful when working with non-text files, such as images, audio, or binary data.

43

I/O

- Example

```
#!/usr/bin/python3

# 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()
```

This produces the following result −

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

https://www.tutorialspoint.com/python/index.htm

44

I/O

- The file Object Attributes
  - Once a file is opened and you have one **file** object, you can get various information related to that file.

Sr.No.	Attribute & Description
1	**file.closed** Returns true if file is closed, false otherwise.
2	**file.mode** Returns access mode with which file was opened.
3	**file.name** Returns name of the file.

https://www.tutorialspoint.com/python/index.htm

45

I/O

- Reading and Writing file
  - The write() Method
    - The **write()** method writes any string to an open file.
    - It is important to note that Python strings can have binary data and not just text.
    - The write() method does not add a newline character ('\n') to the end of the string

```
fileObject.write(string);
```

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

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

https://www.tutorialspoint.com/python/index.htm

46

I/O

- Renaming and Deleting files

```
os.rename(current_file_name, new_file_name)
```

```
import os

# Rename a file from test1.txt to test2.txt
os.rename( "test1.txt", "test2.txt" )
```

```
os.remove(file_name)
```

```
import os

# Delete file test2.txt
os.remove("text2.txt")
```

https://www.tutorialspoint.com/python/index.htm

49

QUESTIONS

51