Introduction to Python (Day 3)

Presenter: Dr. Sourav Saha

Structure of programming language

- Writing output
- Storing values
- Operators and Expressions
- Input Operation
- Decision Control
  - If-else
  - Loop
- Special Data Structures: List, Tuple, Set, Dictionary
- **Function**
- **File Handling**

Functions in Python

A function is a set of statements that take inputs, do some specific computation and produces output. The idea is to put some commonly or repeatedly done task together and make a function, so that instead of writing the same code again and again for different inputs, we can call the function.

```
def printMe():
    print("Let us learn Python Function...")
    print("We want more knowledge...")
printMe()
print("Repeat it...")
printMe()
```

Functions with parameters

```
# A simple Python function to check
# whether x is even or odd
def evenOdd( x ):
        if (x % 2 == 0):
                print ("even")
        else:
                print ("odd")

# Driver code
evenOdd(2)
evenOdd(3)
```

Pass by Reference or pass by value?

```
def myFun(x):
    x = 20

# Driver Code (Note that x is not modified
# after function call.
x = 10
myFun(x);
print(x)
```

Pass by Reference

```
def myFun2(x):
    x = [20, 30, 40]

# Driver Code (Note that lst is not modified
# after function call.
lst = [10, 11, 12, 13, 14, 15]
myFun2(lst);
print(lst)

# Here x is a new reference to same list lst
def myFun3(x):
    x[0] = 20

# Driver Code (Note that lst is modified
# after function call.
lst = [10, 11, 12, 13, 14, 15]
myFun3(lst);
print(lst)
```

Default arguments

A default argument is a parameter that assumes a default value if a value is not provided in the function call for that argument.

```
# Python program to demonstrate
# default arguments
def myFun4(x, y = 50):
        print("x: ", x)
        print("y: ", y)

# Driver code (We call myFun() with only
# argument)
myFun4(10)
myFun4(10, 20)
```

Keyword arguments

The idea is to allow caller to specify argument name with values so that caller does not need to remember order of parameters.

```
# Python program to demonstrate Keyword Arguments
def student(firstname, lastname):
        print(firstname, lastname)

# Keyword arguments
student(firstname ='CU Student', lastname ='Practice')
student(lastname ='Practice', firstname ='CU Student')
```

Variable length arguments

```
# Python program to illustrate
# *args for variable number of arguments
def myFun5(*argv):
        for arg in argv:
                print(arg, end = " ")

myFun5('Hello', 'Welcome', 'to', 'Python Workshop - Day', 3)
```

Anonymous functions

In Python, anonymous function means that a function is without a name. As we already know that def keyword is used to define the normal functions and the lambda keyword is used to create anonymous functions.

```
# Python code to illustrate cube of a number
# using labmda function

cube = lambda x: x*x*x
print(cube(7))
```

Variable scope: local vs global

```
n1 = 10 # global
def func1(n2):
    print("Local Var n2 = ",n2)
    n3 = n1 + n2 # accessing global and local var
    print("Local Var n3 = ",n3)
func1(20)
```

Global variable inside function

```
n1 = 10 # global
def func1(n2):
    print("Local Var n2 = ",n2)
    n3 = n1 + n2 # accessing global and local var
    print("Local Var n3 = ",n3)
    global n4
    n4 = 40
    print("Global Var n4 inside Function = ",n4)
func1(20)
print("Global Var n4 outside Function = ",n4)
```

Scope of a variable

```
var = "Good"
def show():
    # print("Inside function var = ", var)
    var = "Morning"
    print("Inside function var = ", var)
show()
print("Outside function var = ", var)
```

Nested function

```
def outer_function():
    outer_var = 10
    var = 30
    def inner_function():
        inner_var = 20
        var = 40
        # outer_var = outer_var + inner_var
        print("Inside inner_function inner_var = ", inner_var)
        print("Inside inner_function outer_var = ", outer_var)
        print("Inside inner_function var = ", var)
    inner_function()
    print("Inside outer_function var = ", var)
    print("Inside outer_function outer_var = ", outer_var)
outer_function()
```

```
Inside inner_function inner_var =  20
Inside inner_function outer_var =  10
Inside inner_function var =  40
Inside outer_function var =  30
Inside outer_function outer_var =  10
```

Returning multiple values

```
# This function returns a tuple
def fun():
        str = "Python Workshop, Day "
        x = 3
        return str, x; # Return tuple, we could also
                                        # write (str, x)

# Driver code to test above method
str, x = fun() # Assign returned tuple
print(str, x)
```

Recursive function

```
def factorial(n):
    if(n == 0 or n == 1):
        return 1
    else:
        return n * factorial(n - 1)
val = int(input("Enter a number: "))
print("Factorial of ", val," is ", factorial(val))
```

Main function

```
def funMain(str1):
        a = 3
        return str1 + str(a);

def main():
    print("Inside Main Function")
    str1 = funMain("Last slide on Python Function of Day ")
    print(str1)

if __name__ == "__main__":
    main()
```

Modules

Module is a python script with a .py extension that has definitions of all functions and variables that you would like to use in other script.

```
import math
print("The square root of 25 is ", math.sqrt(25))

from math import sqrt
print("The square root of 25 is ", sqrt(25))

from math import sqrt as sqr
print("The square root of 25 is ", sqr(25))
```

Custom modules

MyModule.py

File Edit Format Run Options Window Help

```
name = "MyModule"
def function():
    return "I am MyModule Function"
```

```
import MyModule
print("MyModule's Name is ", MyModule.name)
print("The function of MyModule returns : ", MyModule.function())
```

File handling: writing

```
# FILE OPERATIONS
file = open("File1.txt","w")
file.write("Python Workshop, Day 3")
file.close()
print("Textual Data Written into file \"File1.txt\"")
```

File handling: writing

```
file = open("File2.txt","w")
lines = [str(1)+": line1 \n", str(2) + ": line2 \n", str(3) + ": line3 \n"]
file.writelines(lines) # write list of strings
file.close()
print("Textual Lines Written into file \"File2.txt\"")
```

File handling: append mode

```
file = open("File1.txt","a")
file.write("\n# Append this Line 1")
file.write("\nAppend this Line 2")
file.close()
print("Textual Data appended into file \"File1.txt\"")
```

File handling: reading

```
file = open("File1.txt","r")
fileContent = file.read()
print(fileContent)
file.close()
file = open("File2.txt","r")
fileContent = file.read()
print(fileContent)
file.close()

file = open("File1.txt","r")
fileContent = file.read(6) # read 6 characters
print(fileContent)
file.close()
```

File handling: reading

```
# print all lines
file = open("File2.txt","r")
for line in file:
    print(line)
file.close()

# get lines as a list of strings
file = open("File2.txt","r")
linesList = file.readlines()
print(lines)
for line in linesList:
    print(line)
file.close()

# print line by line
file = open("File2.txt","r")
line = file.readline()
print(line)
line = file.readline()
print(line)
file.close()
```

File handling: safe open & splitting text

```
# with keyword for safe file operation
with open("File1.txt","r") as file:
          line = file.readline()
          print(line)

#splitting words
with open("File1.txt","r") as file:
    line = file.readline()
    wordsList = line.split()
    print(wordsList)
with open("File1.txt","r") as file:
    line = file.readline()
    wordsList = line.split(',')
    print(wordsList)
```

File handling: an example

```
# A Concluding Example

with open("File1.txt", "r") as file1:
    with open("File3.txt", "w") as file2:
        for line in file1:
            if line[0] == '#':
                continue
            else:
                file2.write(line)
```

Thank You &
Stay Tuned!