# PYTHON

**Lecture - 07**

## Lecture Topics

- Python Functions & Modules (Part – **02**)

# Scope of Variables in Python

The **scope of a variable** in Python is defined as **the specific area or region** where the variable is **accessible** to the user.

Python variables are classified in three categories −

- Local Variables
- Global Variables
- Nonlocal Variables

```
pi = 3.1416              → Global

for i in range(10):
    x = input()          → Local
```

# Scope of Variables in Python

## Local Variable

A local variable is **defined within a specific function or block of code**. It can only be **accessed by** the function or block where it was defined, and it has a limited scope.

```
def myfunction():
  a = 10                    #local variable
  b = 20                    #local variable
  print("variable a:", a)
  print("variable b:", b)
  return a+b


print (myfunction())
```

# Scope of Variables in Python

## Local Variable

A local variable is **defined within a specific function or block of code**. It can only be **accessed by** the function or block where it was defined, and it has a limited scope.

```
n = int(input())
list = []
for i in range(n):
    x = int(input())                #local variable
    list.append(x)
print(list)
```

# Scope of Variables in Python

## Global Variable

A global variable can be accessed from any part of the program, and it is defined outside any function or block of code.
It is not specific to any block or function.

```
x = 10              #Global variable
y = 5               #Global variable
def getSum():
  return x + y


sum = getSum()
print(sum)
```

# Scope of Variables in Python

## Nonlocal Variables

The Python variables that are not defined in either local or global scope are called nonlocal variables.
They are used in nested functions.

```
def yourfunction():
  a = 5
  b = 6
  # nested function
  def myfunction():
     # nonlocal function
     nonlocal a
     nonlocal b
     a = 10
     b = 20
     print("variable a:", a)
     print("variable b:", b)
     return a+b
  print (myfunction())
yourfunction()
```

**Output:**
**variable a: 10**
**variable b: 20**
**30**
*variable should not belong to the inner function*

# Scope of Variables in Python

## Local Variables Cannot Be Used in the Global Scope

```
def spam():
    eggs = 31337

spam()
print(eggs)
```

```
Traceback (most recent call last):
  File "/home/ribnat/Python.py", line 5, in <module>
    print(eggs)
NameError: name 'eggs' is not defined
```

# Scope of Variables in Python

## Local Scopes Cannot Use Variables in Other Local Scopes

```
def spam():
    eggs = 99 #local var
    bacon()
    print(eggs)

def bacon():
    ham = 101
    eggs = 0 #another local var

spam()
```

```
99
```

# Scope of Variables in Python

## Global Variables Can Be Read from a Local Scope

```
def spam():
    print(eggs)

eggs = 42
spam()
print(eggs)
```

```
42
42
```

# Scope of Variables in Python

## Local and Global Variables with the Same Name

```
def spam():
    eggs = 'spam local'
    print(eggs) # prints 'spam local'

def bacon():
    eggs = 'bacon local'
    print(eggs) # prints 'bacon local'
    spam()
    print(eggs) # prints 'bacon local'

eggs = 'global'
bacon()
print(eggs) # prints 'global'
```

```
bacon local
spam local
bacon local
global
```

# Scope of Variables in Python

## The global Statement

If you need to modify a global variable from within a function (local scope), use the **global** statement.

```
def spam():
    global eggs
    eggs = 'spam'

eggs = 'global'
print(eggs)
spam()
print(eggs)
```

```
global
spam
```

# Exception Handling

Getting an error, or exception, in your Python program means the entire program will crash.
So, it’s important to detect errors, handle them, and then continue to run.

```
def spam(divideBy):
    return 42 / divideBy

print(spam(2))
print(spam(12))
print(spam(0))
print(spam(1))
```

```
21.0
3.5
Traceback (most recent call last):
  File "/home/ribnat/Python.py", line 6,
in <module>
    print(spam(0))
  File "/home/ribnat/Python.py", line 2,
in spam
    return 42 / divideBy
ZeroDivisionError: division by zero
```

# Exception Handling

Errors can be handled with **try** and **except** statements.

```
def spam(divideBy):
    try:
        return 42 / divideBy
    except ZeroDivisionError:
        print('Error: Invalid
argument.')

print(spam(2))
print(spam(12))
print(spam(0))
print(spam(1))
```

```
21.0
3.5
Error: Invalid argument.
None
42.0
```

# Exception Handling

Any errors that occur in function calls in a try block will also be caught.

```
def spam(divideBy):
    return 42 / divideBy
try:
    print(spam(2))
    print(spam(12))
    print(spam(0))
    print(spam(1))
except ZeroDivisionError:
    print('Error: Invalid
argument.')
```

```
21.0
3.5
Error: Invalid argument.
```

# Python - Modules

A **module** is a file containing definition of **functions**, **classes**, **variables**, **constants** or any other Python **object**.

Contents of this file can be made available to any other program.

Python has the **import** keyword for this purpose.

```
import math          #here, math is a module
print ("Square root of 100:", math.sqrt(100))
```

# Python - Built-in Modules

Python's standard library comes bundled with a large number of modules. i.e.

math → mathematical operations for floating point numbers.
cmath → mathematical operations for complex numbers.
string → contains a number of functions for string processing.
random → generate pseudo-random numbers.

and many more…

# Python - Creating our own Module

Let us save the following code as **mymodule.py**

```
def SayHello(name):
  print ("Hi {}! How are you?".format(name))
  return
```

Now, run the following code through another python file.

```
import mymodule
mymodule.SayHello("Harish")
```

# Python - Creating our own Module

Firstly, save the following code as **myMath.py**

```
def sum(x,y):
    return x+y

def avg(x,y):
    return (x+y)/2

def power(x,y):
    return x**y

def maxTwo(x,y):
    if x>y:
        return x
    else:
        return y
```

Now, run the following code through another python file.

```
import myMath

print (myMath.sum(10,20))
# 30
print (myMath.avg(10,20))
# 15.0
print (myMath.power(10,2))
# 100
print (myMath.maxTwo(10,2))
# 10
```

# Python - Module (The import ... as Statement)

Firstly, save the following code as **myMath.py**

```
def sum(x,y):
    return x+y

def avg(x,y):
    return (x+y)/2

def power(x,y):
    return x**y

def maxTwo(x,y):
    if x>y:
        return x
    else:
        return y
```

Now, run the following code through another python file.

```
from myMath as x


print (x.sum(10,20))
# 30
print (x.avg(10,20))
# 15.0
print (x.power(10,2))
# 100
print (x.maxTwo(10,2))
# 10
```

# Python - Module (The from ... import Statement)

Firstly, save the following code as **myMath.py**

```
def sum(x,y):
    return x+y

def avg(x,y):
    return (x+y)/2

def power(x,y):
    return x**y

def maxTwo(x,y):
    if x>y:
        return x
    else:
        return y
```

Now, run the following code through another python file.

```
from myMath import sum, maxTwo

print (myMath.sum(10,20))
# 30
print (myMath.avg(10,20))
# Error
print (myMath.power(10,2))
# Error
print (myMath.maxTwo(10,2))
# 10
```

# Python - Module (The from ... import* Statement)

Firstly, save the following code as **myMath.py**

```
def sum(x,y):
    return x+y

def avg(x,y):
    return (x+y)/2

def power(x,y):
    return x**y

def maxTwo(x,y):
    if x>y:
        return x
    else:
        return y
```

Now, run the following code through another python file.

```
from myMath import *

print (myMath.sum(10,20))
# 30
print (myMath.avg(10,20))
# 15.0
print (myMath.power(10,2))
# 100
print (myMath.maxTwo(10,2))
# 10
```

# Exercise Time

# Exercise – 7.1

Create a function that takes **a list** of int numbers as a parameter and returns its **sum**.

# Exercise – 7.1 (ans)

```
def findSum(x):
    sum = 0
    for i in x:
        sum+=i
    return sum


List = [2,3,7,10]
print(findSum(List))
```

# Exercise – 7.2

Create a function that takes **two** of **int** numbers as a parameter and returns thair **GCD**.

# Exercise – 7.2 (ans)

```
def GCD(x,y):
    n = min(x,y)
    gcd = 0
    for i in range(1,n+1):
        if x%i == 0 and y%i == 0:
            gcd = i
    return gcd


ans = GCD(8,12)
print(ans)
```

# Exercise – 7.3

Build a **module** called **myString.py** that contain the following functions (each of them takes a string as parameter) —
**vowelCount()** → returns the count of vowel in the string.
**uniqueChar()** → returns the count of unique char of the string.

And then import and call the functions from another python file.

**For Example:**
vowelCount(“school”) → return 2, as there are 2 vowels.
uniqueChar(“exercise”) → return 6, as there are 6 unique char.

# Exercise – 7.3 (ans)

```
#save file as myString.py
def vowelCount(str):
    cnt = 0
    for ch in str:
        if(ch=='a' or ch=='e' or ch=='i' or ch=='o' or ch=='u'):
            cnt+=1
    return cnt

def uniqueChar(str):
    List = []
    for i in range(26):
        List.append(0)
    for ch in str:
        idx = int(ord(ch)) - int(ord('a'))
        List[idx]=1
    cnt = 0
    for i in List:
        if(i>0):
            cnt+=1
    return cnt
```

# Exercise – 7.3 (ans cont.)

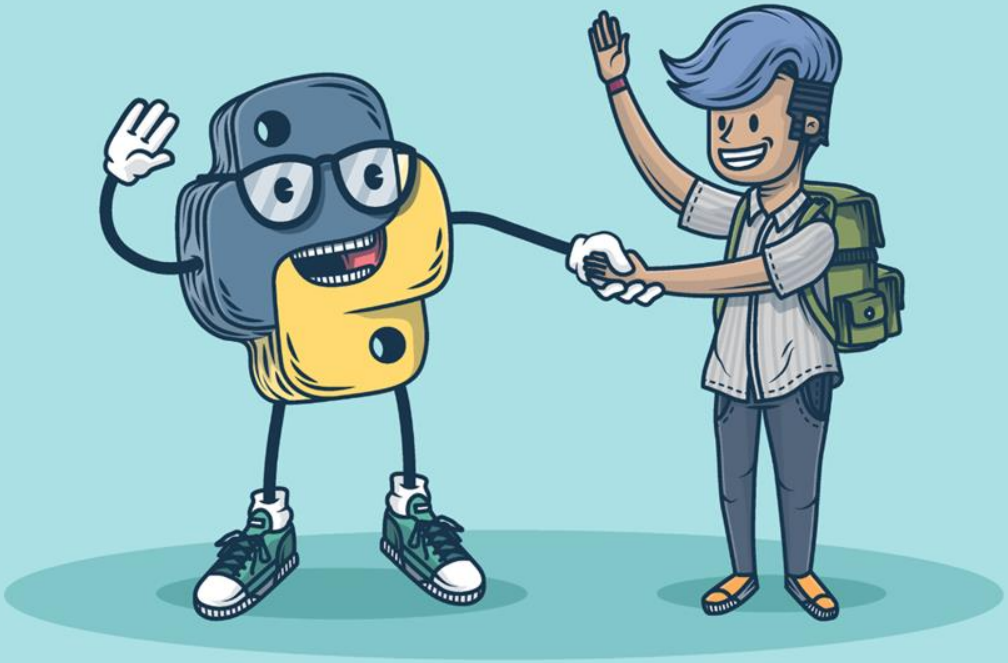
```
#In another python file, write the code and run

import myString

print(myString.vowelCount("school"))
print(myString.uniqueChar("exercise"))
```

# Resources

- https://www.tutorialspoint.com/python/index.htm
- https://www.w3resource.com/python/python-tutorial.php
- https://www.w3resource.com/python-exercises/string/
- https://www.w3schools.com/python/
- https://www.geeksforgeeks.org/python-programming-language/
- https://youtu.be/t2_Q2BRzeEE?si=OO6J_YNCZykedqsT
- https://realpython.com/
- Head First Python, 3rd Edition by Paul Barry
- Automate the Boring Stuff with Python By Al Sweigart.



# Thank You