Python Functions

Dr Muhammad Aslam
Lecturer UWS Wuxi
Muhammad.Aslam@uws.ac.uk

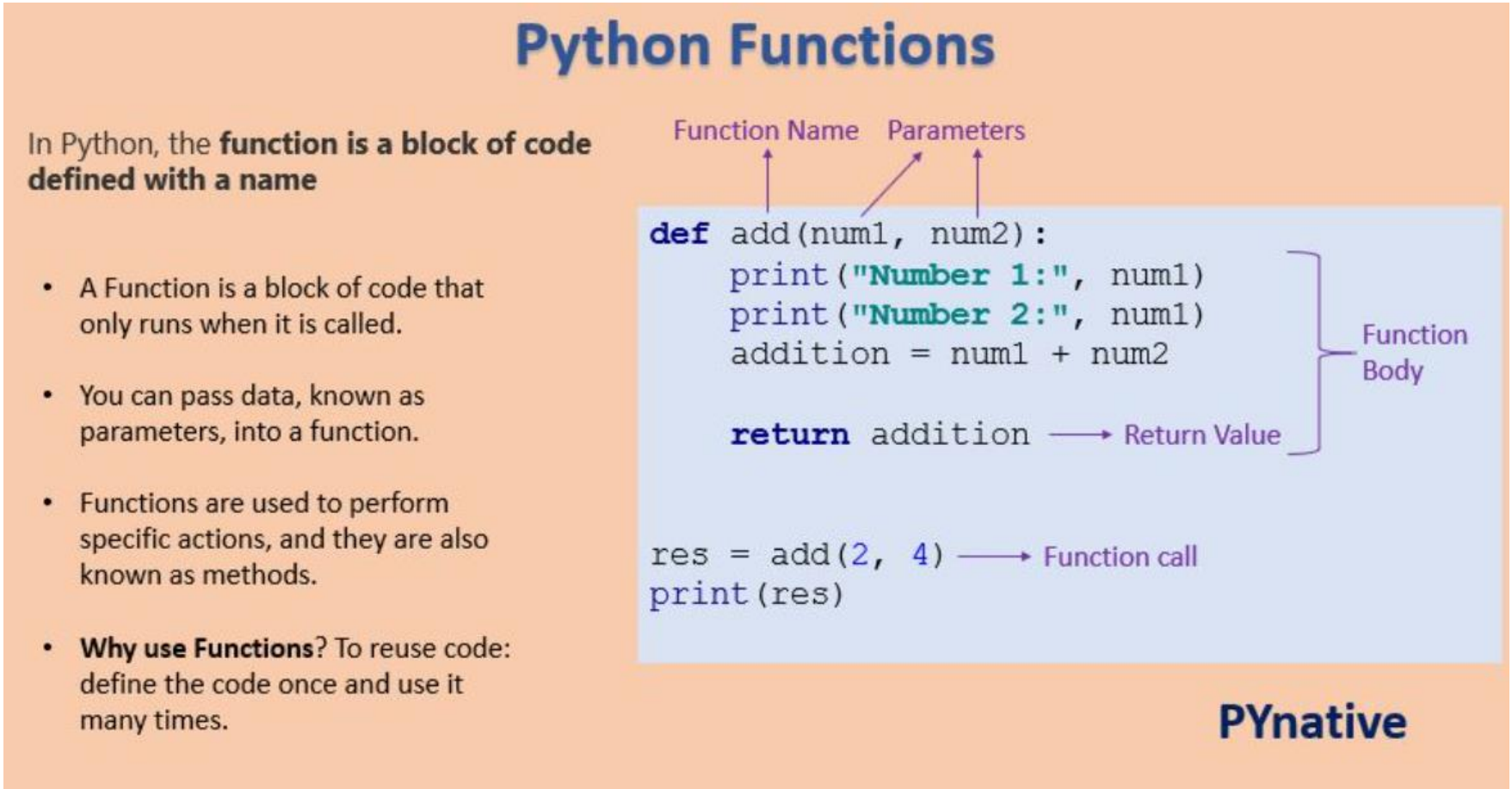
Introduction to Programming
Comp07027

Lecture 8: **Python Functions (2)**

User-defined function

- Functions which are created by programmer explicitly according to the requirement are called a user-defined function.
- **Creating a Function**
- Use the following steps to define a function in Python.
    1. Use the **def** keyword with the function name to define a function.
    2. Next, pass the number of parameters as per your requirement. (Optional).
    3. Next, define the function body with a **block of code**. This block of code is nothing but the action you wanted to perform.
- In Python, no need to specify curly braces for the function body. The only **indentation** is essential to separate code blocks. Otherwise, you will get an error.

Function Basic Flowchart



Local Variable in function

- A local variable is a variable declared inside the function that is not accessible from outside of the function. The scope of the local variable is limited to that function only where it is declared.
- If we try to access the local variable from the outside of the function, we will get the error as NameError.
-
```
def function1():
    # local variable
    loc_var = 888
    print("Value is :", loc_var)

def function2():

    print("Value is :", loc_var)

function1()
function2()
```

Example: Global Keyword in Function


```
# Global variable
global_var = 5

def function1():
    print("Value in 1st function :", global_var)

def function2():
    # Modify global variable
    # function will treat it as a local variable
    global_var = 44
    print("Value in 2nd function :", global_var)

def function3():
    print("Value in 3rd function :", global_var)

function1()
function2()
function3()
```

As you can see, function2() treated global_var as a new variable (local variable). To solve such issues or access/modify global variables inside a function, we use the global keyword.

Nested Function

- Python supports the concept of a "nested function" or "inner function", which is simply a function defined inside another function.
- The inner function can access the variables within the enclosing scope.
- Defining an Inner Function
- To define an inner function in Python, we simply create a function inside another function using the Python's def keyword. Here is an example:
-
```
def function1(): # outer function
    print ("Hello from outer function")
    def function2(): # inner function
        print ("Hello from inner function")
    function2()

function1()
```

Outer Function call for Nested Function

- It is important to mention that the outer function must be called for the inner function to execute.
- If the outer function is not called, the inner function will never execute. To demonstrate this, modify the above code to the following and run it:

```
def function1(): # outer function
    print ("Hello from outer function")
    def function2(): # inner function
        print ("Hello from inner function")
    function2()
```

Example:Outer Function call for Nested Function


```
def num1(x):
    def num2(y):
        return x * y
    return num2
res = num1(10)

print(res(5))
```

- # output is 50

- The code returns the multiplication of the two numbers, that is, 10 and 5. The example shows that an inner function is able to access variables accessible in the outer function.

Access of variable within Nested Function

- So far, you have seen that it is possible for us to access the variables of the outer function inside the inner function. What if we attempt to change the variables of the outer function from inside the inner function? Let us see what happens:

- ```
  def function1(): # outer function
      x = 2 # A variable defined within the outer function
      def function2(a): # inner function
          # Let's define a new variable within the inner function
          # rather than changing the value of x of the outer function
           x = 6
           print (a+x)
      print (x) # to display the value of x of the outer function
      function2(3)

  function1()
  ```

- The output shows that it is possible for us to display the value of a variable defined within the outer function from the inner function, but not change it. The statement x = 6 helped us create a new variable x inside the inner function function2() rather than changing the value of variable x defined in the outer function function1() .
  ```

Nonlocal Variable in Function

- In Python, nonlocal is the keyword used to declare a variable that acts as a global variable for a nested function (i.e., function within another function).

- We can use a nonlocal keyword when we want to declare a variable in the local scope but act as a global scope.
-
```
def outer_func():
    x = 777

    def inner_func():
        # local variable now acts as global variable
        nonlocal x
        x = 7000
        print("value of x inside inner function is :", x)

    inner_func()
    print("value of x inside outer function is :", x)

outer_func()
```

Python Function Arguments

- The argument is a value, a variable, or an object that we pass to a function or method call. In Python, there are four types of arguments allowed.

1.Positional arguments

2.keyword arguments

3.Default arguments

4.Variable-length arguments

Positional Arguments

- Positional arguments are arguments that are pass to function in proper positional order.
- That is, the 1st positional argument needs to be 1st when the function is called. The 2nd positional argument needs to be 2nd when the function is called, etc.
- See the following example for more understanding.
- ```
  def add(a, b):
      print(a - b)

  add(50, 10)
  # Output 40
  add(10, 50)
  # Output -40
  ```
- If you try to use pass more parameters you will get an error.
- ```
  add(50, 10, 30)
  ```

Keyword Arguments

- A keyword argument is an argument value, passed to function preceded by the variable name and an equals sign.

- ```
  def message(name, surname):
      print("Hello", name, surname)

  message(name="John", surname="Wilson")
  message(surname="Ault", name="Kelly")
  ```
- In keyword arguments order of argument is not matter, but the number of arguments must match. Otherwise, we will get an error.
  ```

Example: Keyword Arguments

- While using keyword and positional argument simultaneously, we need to pass 1st arguments as positional arguments and then keyword arguments.
- Otherwise, we will get SyntaxError. See the following example.

```
def message(first_nm, last_nm):
    print("Hello..!", first_nm, last_nm)

# correct use
message("John", "Wilson")
message("John", last_nm="Wilson")

# Error
# SyntaxError: positional argument follows keyword argument
message(first_nm="John", "Wilson")
```

Default Arguments

- Default arguments take the default value during the function call if we do not pass them.
- We can assign a default value to an argument in function definition using the = assignment operator.
-
```
def message(name="Guest"):
    print("Hello", name)

# calling function with argument
message("John")

# calling function without argument
message()
```

Variable-length Arguments

- In Python, sometimes, there is a situation where we need to pass multiple numbers of arguments to the function.
- Such types of arguments are called **variable-length arguments**.
- We can declare a variable-length argument with the * **(asterisk)** symbol.
-
```
def fun(*var):
    function body
```
- We can pass any number of arguments to this function. Internally all these values are represented in the form of a **tuple**.
-

```
def addition(*numbers):
    total = 0
    for no in numbers:
        total = total + no
    print("Sum is:", total)


# 0 arguments
addition()
# 5 arguments
addition(10, 5, 2, 5, 4)

# 3 arguments
addition(78, 7, 2.5)
```

Recursive Function

- A recursive function is a function that calls itself, again and again.
- Consider, calculating the factorial of a number is a repetitive activity, in that case, we can call a function again and again, which calculates factorial.

factorial(5)

5*factorial(4)

5*4*factorial(3)

5*4*3*factorial(2)

5*4*3*2*factorial(1)

5*4*3*2*1 = 120

```
def factorial(no):
    if no == 0:
        return 1
    else:
        return no * factorial(no - 1)

print("factorial of a number is:", factorial(8))
```

Recursive Function

- The advantages of the recursive function are:

1.By using recursive, we can reduce the length of the code.

2.The readability of code improves due to code reduction.

3.Useful for solving a complex problem

- The disadvantage of the recursive function:

1.The recursive function takes more memory and time for execution.

2.Debugging is not easy for the recursive function.

Python Anonymous/Lambda Function

- Sometimes we need to declare a function without any name. The nameless property function is called an anonymous function or lambda function.

- The reason behind the using anonymous function is for instant use, that is, one-time usage.
- Normal function is declared using the def function. Whereas the anonymous function is declared using the lambda keyword.
- In opposite to a normal function, a Python lambda function is a single expression.
- But, in a lambda body, we can expand with expressions over multiple lines using parentheses or a multiline string.

Python Anonymous/Lambda Function

- **Syntax of `lambda` function:**
- lambda: argument_list:expression
- When we define a function using the lambda keyword, the code is very concise so that there is more readability in the code.
- A lambda function can have any number of arguments but return only one value after expression evaluation.
- Let’s see an example to print even numbers without a lambda function and with a lambda function.
- See the difference in line of code as well as readability of code.

Example 1: Program for even numbers without lambda function


```
#Example 1: Program for even numbers without lambda function
def even_numbers(nums):
    even_list = []
    for n in nums:
        if n % 2 == 0:
            even_list.append(n)
    return even_list

num_list = [10, 5, 12, 78, 6, 1, 7, 9]
ans = even_numbers(num_list)
print("Even numbers are:", ans)
```

Example 1: Program for even numbers with a lambda function


```
#Example 1: Program for even numbers without lambda function
def even_numbers(nums):
    even_list = []
    for n in nums:
        if n % 2 == 0:
            even_list.append(n)
    return even_list

num_list = [10, 5, 12, 78, 6, 1, 7, 9]
ans = even_numbers(num_list)
print("Even numbers are:", ans)
#Example 2: Program for even number with a lambda function

l = [10, 5, 12, 78, 6, 1, 7, 9]
even_nos = list(filter(lambda x: x % 2 == 0, l))
print("Even numbers are: ", even_nos)
```

filter() function in Python

- In Python, the filter() function is used to return the filtered value. We use this function to filter values based on some conditions.
- **filter**(funtion, sequence)
- function – Function argument is responsible for performing condition checking.
- sequence – Sequence argument can be anything like list, tuple, string
- #Example: lambda function with filter()

```
l = [-10, 5, 12, -78, 6, -1, -7, 9]
positive_nos = list(filter(lambda x: x > 0, l))
print("Positive numbers are: ", positive_nos)
```

map() function in Python

- In Python, the map() function is used to apply some functionality for every element present in the given sequence and generate a new series with a required modification.
- Ex: for every element present in the sequence, perform cube operation and generate a new cube list.
- **map**(function, sequence)

•function – function argument responsible for applied on each element of the sequence

•sequence – Sequence argument can be anything like list, tuple, string

•#Example: lambda function with map() function

```
list1 = [2, 3, 4, 8, 9]
list2 = list(map(lambda x: x*x*x, list1))
print("Cube values are:", list2)
```

`reduce()` function in Python

- In Python, the reduce() function is used to **minimize sequence elements** into a **single value** by applying the specified condition.
- The reduce() function is present in the functools module; hence, we need to import it using the import statement before using it.
- **Syntax of reduce() function:** reduce(function, sequence)

```
#Example: lambda function with reduce()

from functools import reduce
list1 = [20, 13, 4, 8, 9]
add = reduce(lambda x, y: x+y, list1)
print("Addition of all list elements is : ", add)
```

Questions??