Python Function

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

Introduction to Programming
Comp07027

Lecture 7: **Python Functions (1)**

Python Functions

- In Python, the **function is a block of code defined with a name**.
- We use functions whenever we need to perform the same task multiple times without writing the same code again.
- It can take arguments and returns the value.
- Python has a DRY principle like other programming languages. DRY stands for Don’t Repeat Yourself.
- Consider a scenario where we need to do some action/task many times.
- We can define that action only once using a function and call that function whenever required to do the same activity.
- Function improves efficiency and reduces errors because of the reusability of a code. Once we create a function, we can call it anywhere and anytime. The benefit of using a function is reusability and modularity.

Types of Functions

- Python support two types of functions
  1. Built-in function
  2. User-defined function
- Built-in function: The functions which are come along with Python itself are called a built-in function or predefined function. Some of them are listed below.

range(), type(), input() print (), len() etc.

- Python range() function generates the immutable sequence of numbers starting from the given start integer to the stop integer.
-
```
for i in range(1, 10):
    print(i, end=' ')
# Output 1 2 3 4 5 6 7 8 9
```

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.

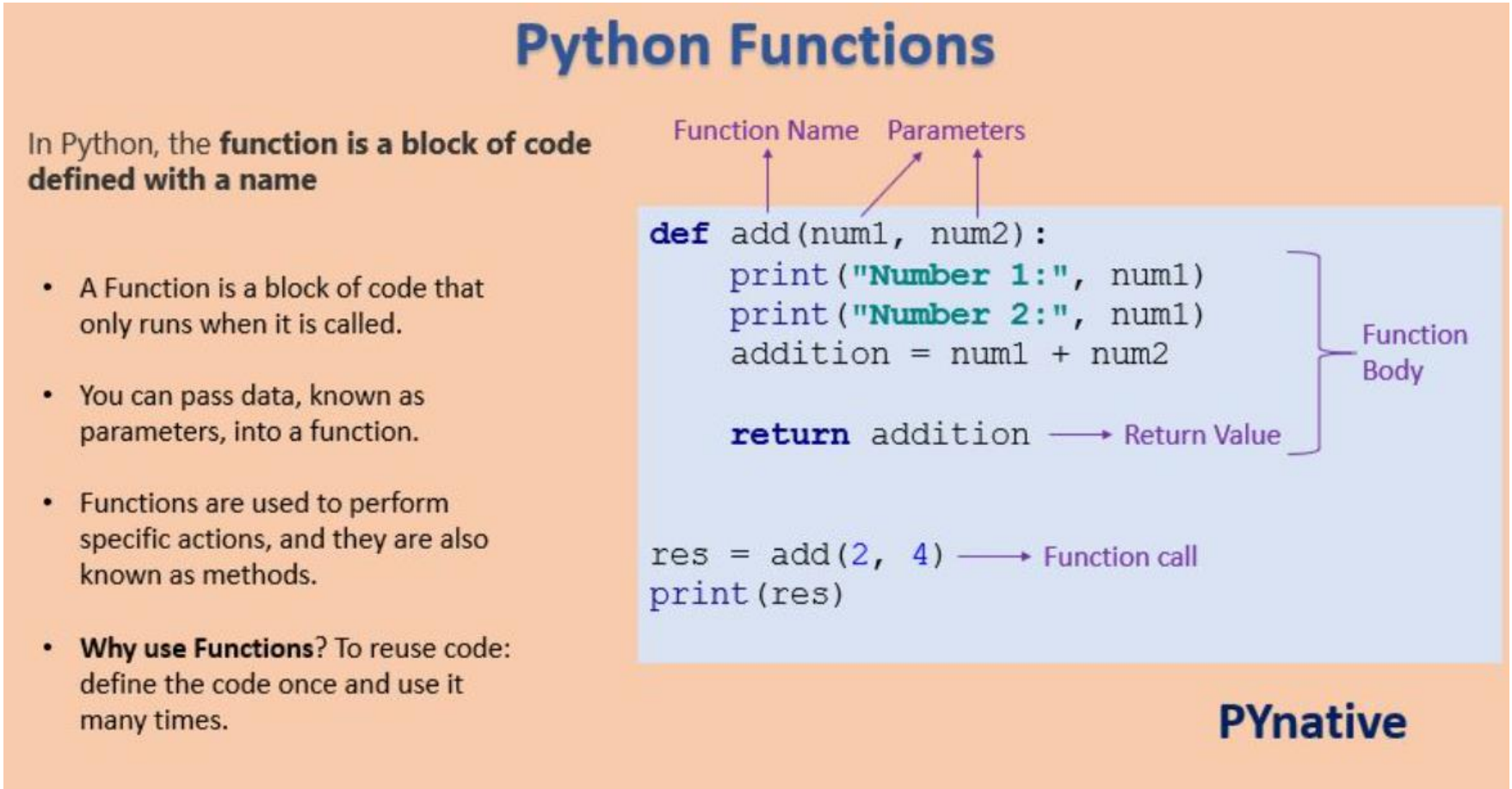
Syntax of creating a function

- **Syntax of creating a function**

```
def function_name(parameter1, parameter2):
        # function body
        # write some action
return value
```

- `function_name`: Function name is the name of the function. We can give any name to function.
- `parameter`: Parameter is the value passed to the function. We can pass any number of parameters. Function body uses the parameter's value to perform an action
- `function_body`: The function body is a block of code that performs some task. This block of code is nothing but the action you wanted to accomplish.
- `return value:` Return value is the output of the function.
- **Note:** While defining a function, we use two keywords, `def` (mandatory) and `return` (optional).

Function Basic Flowchart



Creating a function without any parameters

- Now, Let's the example of creating a simple function that prints a welcome message.

```
def message():
    print("Welcome to PYnative")

# call function using its name
message()
```

Creating a function with parameters

Let’s create a function that takes two parameters and displays their values.
In this example, we are creating function with two parameters ‘ name’ and ‘age’.

function

- ```
  # function
  def course_func(name, course_name):
      print("Hello", name, "Welcome to PYnative")
      print("Your course name is", course_name)

  # call function
  course_func('John', 'Python')
  ```
-
  ```

Creating a function with parameters and return value

- Functions can return a value. The return value is the output of the function. Use the return keyword to return value from a function.

- ```
  def calculator(a, b):
      add = a + b
      # return the addition
      return add


  # call function
  # take return value in variable
  res = calculator(20, 5)
  print("Addition :", res)
  ```
  ```

Calling a function

- Once we defined a function or finalized structure, we can call that function by using its name. We can also call that function from another function or program by importing it.
- To call a function, use the name of the function with the parenthesis, and if the function accepts parameters, then pass those parameters in the parenthesis.
-

```
def even_odd(n):
    # check numne ris even or odd
    if n % 2 == 0:
        print('Even number')
    else:
        print('Odd Number')

# calling function by its name
even_odd(30)
```

Calling a function of a module

- You can take advantage of the built-in module and use the functions defined in it.
- For example, Python has a random module that is used for generating random numbers and data.
- It has various functions to create different types of random data.
- Let's see how to use functions defined in any module.
    1. First, we need to use the import statement to import a specific function from a module.
    2. Next, we can call that function by its name.
    3.
```
# import randint function
from random import randint

# call randint function to get random number
print(randint(10, 20))
```

Docstrings

- In Python, the documentation string is also called a **docstring**. It is a descriptive text (like a comment) written by a programmer to let others know what block of code does.
- We write docstring in source code and define it immediately after module, class, function, or method definition.
- It is being declared using triple single quotes (''' ''') or triple-double quote(""" """).
- We can access docstring using doc attribute (__doc__) for any object like list, tuple, dict, and user-defined function, etc.

Multi-Line Docstring

- A multi-line Docstrings is the same single-line Docstrings, but it is followed by a single blank line with the descriptive text.
- The general format of writing a multi-line Docstring is as follows:
- ```
  # Multi-line Docstring
  def any_fun(parameter1):
      """
          Description of function

          Arguments:
          parameter1(int):Description of parameter1

          Returns:
          int value
      """
  ```
  ```

Return Value From a Function

- In Python, to return value from the function, a return statement is used. It returns the value of the expression following the returns keyword.
- Syntax of return statement
- def fun():
  statement-1
  statement-2
  statement-3
  .

  return [expression]

Example: Return Value From a Function

- The return value is nothing but a outcome of function.
    1. The return statement ends the function execution.
    2. For a function, it is not mandatory to return a value.
    3. If a return statement is used without any expression, then the None is returned.
    4. The return statement should be inside of the function block.

```
def is_even(list1):
    even_num = []
    for n in list1:
        if n % 2 == 0:
            even_num.append(n)
    # return a list
    return even_num
# Pass list to the function
even_num = is_even([2, 3, 42, 51, 62, 70, 5, 9])
print("Even numbers are:", even_num)
```

Return Multiple Values

- You can also return multiple values from a function.
- Use the return statement by separating each expression by a comma.
- **Example**: –

  In this example, we are returning three values from a function. We will also see how to process or read multiple return values in our code.

```
def arithmetic(num1, num2):
    add = num1 + num2
    sub = num1 - num2
    multiply = num1 * num2
    division = num1 / num2
    # return four values
    return add, sub, multiply, division

print(arithmetic(10, 2))
# read four return values in four variables
a, b, c, d = arithmetic(10, 2)
```

The pass Statement

- In Python, the pass is the keyword, which won't do anything. Sometimes there is a situation where we need to define a syntactically empty block. We can define that block using the pass keyword.
- When the interpreter finds a pass statement in the program, it returns **no operation**.
- # The pass statement for function

```
def addition(num1, num2):
    # Implementation of addition function in coming release
    # Pass statement
    pass

addition(10, 2)
```

How does Function work in Python?

- In Python, functions allow the programmer to create short and clean code to be reused in an entire program.
- The function helps us to organize code. The function accepts parameters as input, processes them, and in the end, returns values as output.
- Let's assume we defined a function that computes some task. When we call that function from another function, the program controller goes to that function, does some computation, and returns some value as output to the caller function.

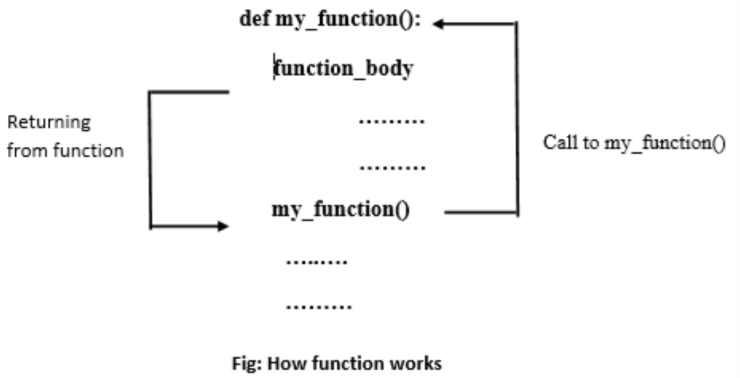


Fig: How function works

Scope and Lifetime of Variables

- When we define a function with variables, then those variables' scope is limited to that function. In Python, the scope of a variable is an area where a variable is declared. It is called the variable's local scope.
- We cannot access the local variables from outside of the function. Because the scope is local, those variables are not visible from the outside of the function.
- **Note**: The inner function does have access to the outer function's local scope.
- When we are executing a function, the life of the variables is up to running time. Once we return from the function, those variables get destroyed. So function does no need to remember the value of a variable from its previous call.
-

Example: Scope and Lifetime of Variables


```
# Varibale scope and areas

global_lang = 'DataScience'

def var_scope_test():
    local_lang = 'Python'
    print(local_lang)

var_scope_test()
# Output 'Python'

# outside of function
print(global_lang)
# Output 'DataScience'

# NameError: name 'local_lang' is not defined
print(local_lang)
```

•

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()
```

Global Variable in function

- A Global variable is a variable that declares outside of the function. The scope of a global variable is broad. It is accessible in all functions of the same module.
-

```
# Global Variable use
global_var = 999

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

def function2():
    print("Value in 2nd function :", global_var)

function1()
function2()
```

Global Keyword in Function

In Python, global is the keyword used to access the actual global variable from outside the function. we use the global keyword for two purposes:

1.To declare a global variable inside the function.

2.Declaring a variable as global, which makes it available to function to perform the modification.

Let’s see what happens when we don’t use global keyword to access the global variable in the function

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 = 555
    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.

Example: Global Keyword in Function

global keyword.

```
# Global variable
x = 5
# defining 1st function
def function1():
    print("Value in 1st function :", x)

# defining 2nd function
def function2():
    # Modify global variable using global keyword
    global x
    x = 555
    print("Value in 2nd function :", x)

# defining 3rd function
def function3():
    print("Value in 3rd function :", x)
function1()
function2()
function3()
```

Questions??