Python Programming

Practical Questions - Database Management

Session 3

FUNCTIONS

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()`, `id()`, `type()`, `input()`, `eval()` etc.

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 to define a function in Python.

- Use the `def` keyword with the function name to define a function.
- Next, pass the number of parameters as per your requirement. (Optional).
- 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

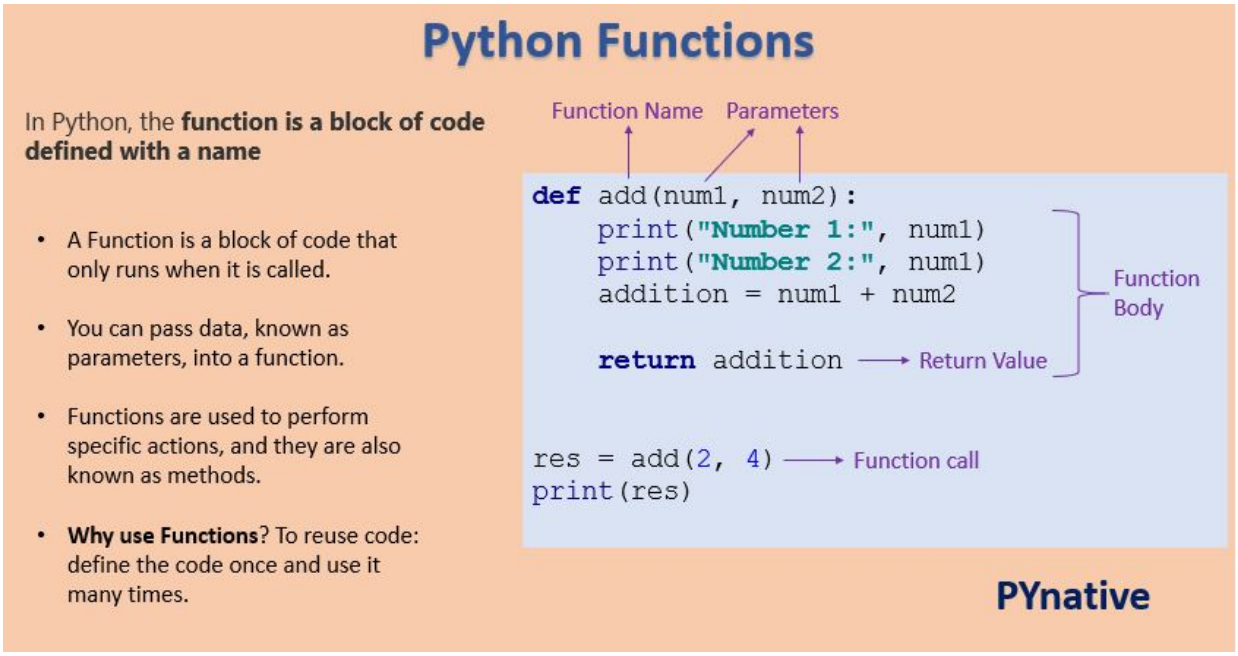
```
def function_name(parameter1, parameter2):
       # function body
       # write some action
return value
```

Here,

- `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).



Python Functions

Creating a function without any parameters

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

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

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

Output

```
Welcome to PYnative
```

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
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')
```

Output

```
Hello John Welcome to PYnative
Your course name is Python
```

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.

Example

```
# function
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(19)
# Output Odd Number
```

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**.

Example

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

addition(10, 2)
```

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.

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.

Example

```
def function1():
    # local variable
    loc_var = 888
    print("Value is :", loc_var)

def function2():

    print("Value is :", loc_var)

function1()
function2()
```

Output

```
Value is : 888
print("Value is :", loc_var) # gives error,
NameError: name 'loc_var' is not defined
```

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.

Example

```
global_var = 999

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

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

function1()
function2()
```

Output

```
Value in 1nd function : 999
Value in 2nd function : 999
```

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.
ex : `lambda n:n+n`

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

```
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)
```

Output

```
Even numbers are: [10, 12, 78, 6]
```

`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.

Syntax of `filter()` function:

```
filter(funtion, sequence)
```

where,

- 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)
```

Output

```
Positive numbers are: [5, 12, 6, 9]
```

`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.

Syntax of `map()` function:

```
map(function,sequence)
```

where,

- `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)
```

Output

```
Cube values are: [8, 27, 64, 512, 729]
```

`reduce()` function in Python

In Python, the `reduce()` function is used to **minimize sequence elements** into a **single value** by applying the specified condition.

Example: 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
```

Exercise 1: Create a function in Python

Write a program to create a function that takes two arguments, name and age, and print their value.

Hint

> - Use the `def` keyword with the function name to define a function.
> - Next, take two parameters

> - Print them using the print() function
> - Call function by passing name and age.

Exercise 2: Return multiple values from a function

Write a program to create function `calculation()` such that it can accept two variables and calculate addition and subtraction. Also, it must **return both addition and subtraction in a single return call**.

Given:

```
def calculation(a, b):
    # Your Code

res = calculation(40, 10)
print(res)
```

Expected Output

```
??
```

Exercise 3: Find the largest item from a given list

```
x = [4, 6, 8, 24, 12, 2]
```

Expected Output:

```
24
```

**Hint*

Use a certain built-in function