
CodroidHub Summer Training

Index

- Break
- Continue
- Predefined function
- User Defined Function
 - Default Function
 - Recursive Function
- Return Type Function
 - Package
 - Class and Object
 - Constructor

SUBMITTED BY: ASMIT

ROLL NO: (2322811)

BRANCH: AI&ML

SUBMITTED TO: Mr. DEVASHISH SIR
(FOUNDER, CodroidHub Private Limited)

Break

The break statement in Python is used to terminate the loop or statement in which it is present. After that, the control will pass to the statements that are present after the break statement, if available. If the break statement is present in the nested loop, then it terminates only those loops which contain the break statement.

Syntax of break

for / while loop:

 # statement(s)

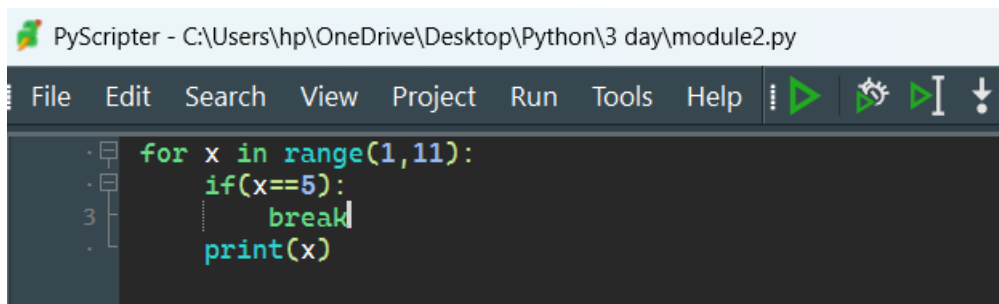
 if condition:

 break

 # statement(s)

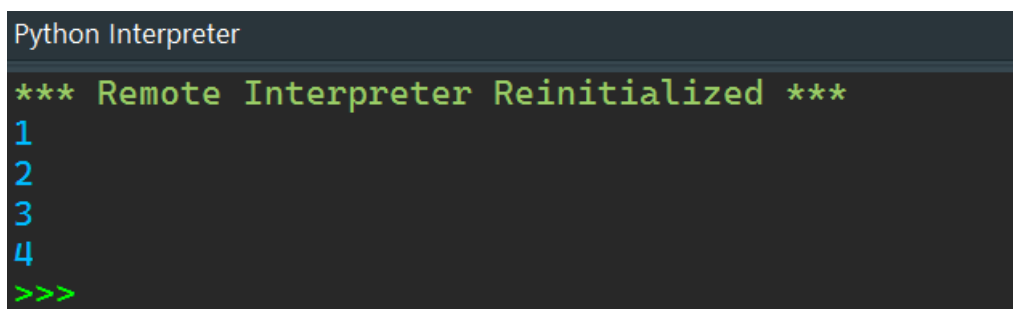
loop end

Example:



```
PyScripter - C:\Users\hp\OneDrive\Desktop\Python\3 day\module2.py
File Edit Search View Project Run Tools Help
for x in range(1,11):
    if(x==5):
        break
    print(x)
```

Output:



```
Python Interpreter
*** Remote Interpreter Reinitialized ***
1
2
3
4
>>>
```

Continue

Continue is also a loop control statement just like the break statement. continue statement is opposite to that of the break statement, instead of terminating the loop, it forces to execute the next iteration of the loop. As the name suggests the continue statement forces the loop to continue or execute the next iteration. When the continue statement is executed in the loop, the code inside the loop following the continue statement will be skipped and the next iteration of the loop will begin.

Syntax of Continue

for / while loop:

 # statement(s)

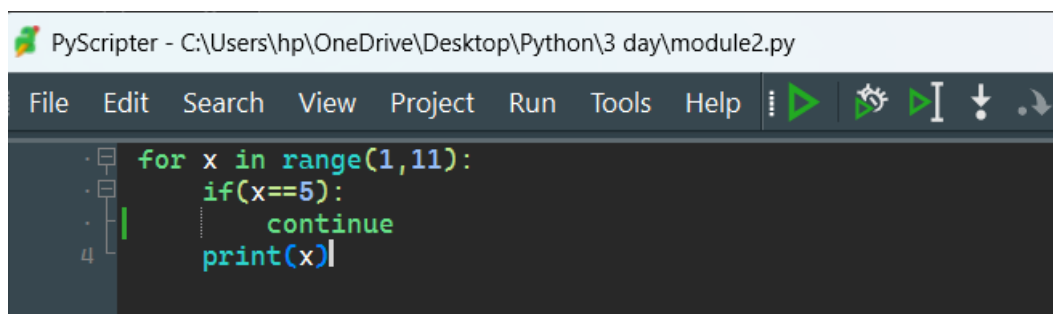
 if condition:

 continue

 # statement(s)

loop end

Example:



```
PyScripter - C:\Users\hp\OneDrive\Desktop\Python\3 day\module2.py
File Edit Search View Project Run Tools Help
for x in range(1,11):
    if(x==5):
        continue
    print(x)
```

Output:

```
Python Interpreter
*** Remote Interpreter Reinitialized ***
1
2
3
4
6
7
8
9
10
>>>
```

Predefined Function

Predefined functions, also known as built-in functions, are functions that are already defined in Python's standard library. These functions are readily available for use without any additional coding or importing of libraries.

Example:

```
PyScripter - C:\Users\hp\OneDrive\Desktop\Python\3 day\3.py
File Edit Search View Project Run Tools Help
print("Welcome to Python")
print(max(2,7,4))
print(min(2,7,4))
4
x=(7,9,3,1)
print("sum=",sum(x))
print("max=",max(x))
print(abs(-5.9))
10
myname = ["asmit","tarun","vikas"]
print(myname.count("asmit"))
```

Output:

```
Python Interpreter
>>>
*** Remote Interpreter Reinitialized ***
Welcome to Python
7
2
sum= 20
max= 9
5.9
1
>>>
```

User Defined Function

All the functions that are written by any of us come under the category of user-defined functions.

How to create User Defined Function

- A def keyword is used to declare user-defined functions.
- An indented block of statements follows the function name and arguments which contains the body of the function.

Example:

```
PyScripter - C:\Users\hp\OneDrive\Desktop\Python\3 day\3.py
File Edit Search View Project Run Tools Help
1 #user defined function
2
3 def myfunction():
4     print ("welcome to function")
5
6 myfunction() #calling function
7
8
9
10 #function with parameter
11
12 def myfun(message):
13     print (message)
14
15 myfun("hi")
```

Output:

```
Python Interpreter
*** Remote Interpreter Reinitialized ***
welcome to function
hi
>>>
```

Default Argument

Default arguments in Python are values provided in a function definition that are used if no corresponding argument is passed during the function call.

Default arguments are specified in the function definition by assigning a value to the parameter. When a function with default arguments is called, the default values are used if no explicit value is provided for those parameters.

Example:

```
PyScripter - C:\Users\hp\OneDrive\Desktop\Python\3 day\4.py
File Edit Search View Project Run Tools Help
def details(name, course="BIT"):
    print("Name:", name)
    print("COURSE:", course)

details("raj", "MIT")
```

Output:

```
4.py × class.py module2.py 3.py
Python Interpreter
*** Remote Interpreter Reinitialized ***
Name: raj
COURSE: MIT
>>>
```

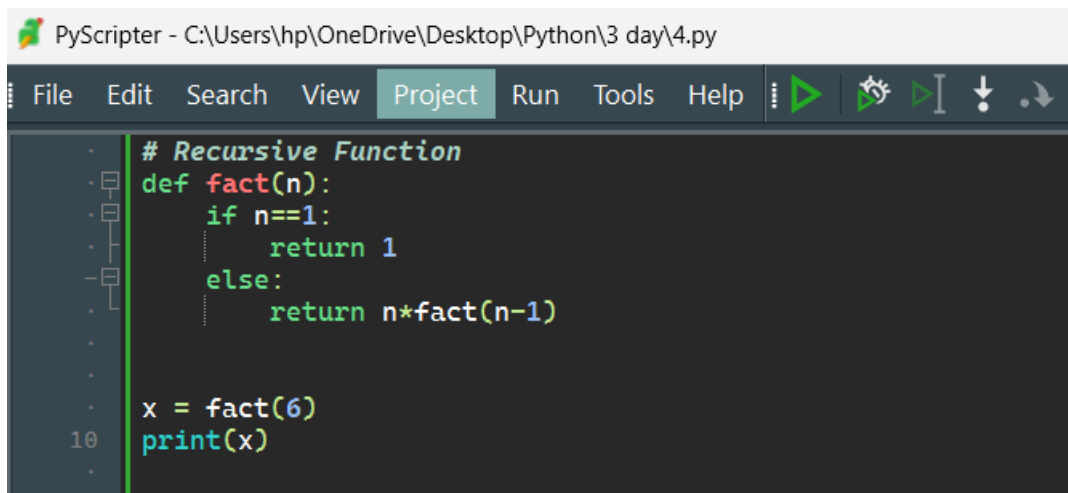
Recursive Function

A recursive function in Python is a function that calls itself in order to solve a problem. Recursive functions break down a problem into smaller subproblems of the same type. Each recursive call works on a smaller version of the original problem, and the recursion typically has a base case to stop the recursion.

Key Components of a Recursive Function

1. **Base Case:** The condition under which the recursion stops. Without a base case, the function would call itself indefinitely, leading to a stack overflow.
2. **Recursive Case:** The part of the function where it calls itself with a smaller or simpler input.

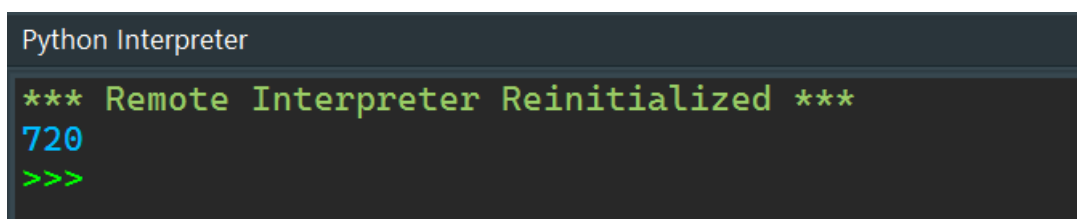
Example:

A screenshot of the PyScripter IDE. The title bar reads 'PyScripter - C:\Users\hp\OneDrive\Desktop\Python\3 day\4.py'. The menu bar includes File, Edit, Search, View, Project, Run, Tools, and Help. The code editor displays a Python script for a recursive factorial function. The code is as follows:

```
# Recursive Function
def fact(n):
    if n==1:
        return 1
    else:
        return n*fact(n-1)

x = fact(6)
print(x)
```

Output:

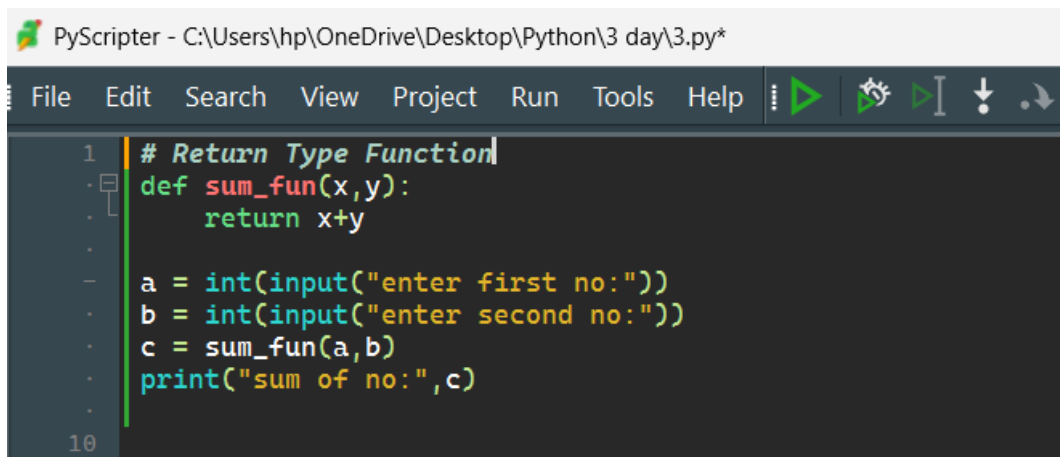
A screenshot of the Python Interpreter window. It shows the output of the recursive function. The text displayed is:

```
*** Remote Interpreter Reinitialized ***
720
>>>
```

Return Type Function

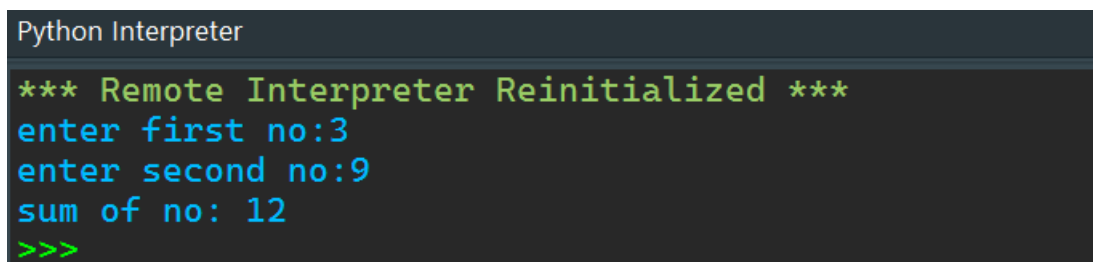
In Python, the return type of a function is the type of value that the function returns. Functions can return various types of values, including integers, floats, strings, lists, dictionaries, tuples, or even other functions. The return statement is used to exit a function and return a value from the function to the caller.

Example:



```
PyScripter - C:\Users\hp\OneDrive\Desktop\Python\3 day\3.py*
File Edit Search View Project Run Tools Help
1 # Return Type Function
  def sum_fun(x,y):
    return x+y
a = int(input("enter first no:"))
b = int(input("enter second no:"))
c = sum_fun(a,b)
print("sum of no:",c)
```

Output:

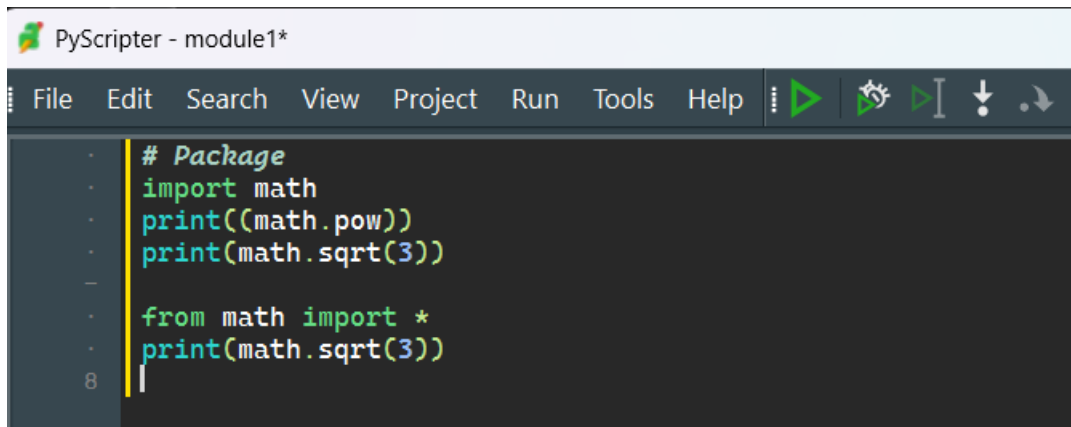


```
Python Interpreter
*** Remote Interpreter Reinitialized ***
enter first no:3
enter second no:9
sum of no: 12
>>>
```

Package

In Python, a package is a way of organizing related modules into a directory hierarchy. A package is essentially a directory that contains a special file named `__init__.py` (which can be empty), along with other module files and sub-packages. Packages allow you to structure your Python code into manageable sections and promote modularity and reusability.

Example:

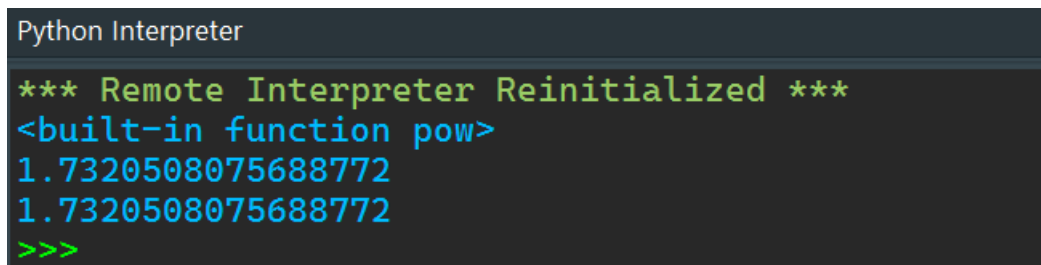
A screenshot of the PyScripter application window titled "PyScripter - module1*". The window has a menu bar with "File", "Edit", "Search", "View", "Project", "Run", "Tools", and "Help". Below the menu bar is a toolbar with icons for running, debugging, and other functions. The main text area contains the following Python code:

```
# Package
import math
print((math.pow))
print(math.sqrt(3))

from math import *
print(math.sqrt(3))
```

The line number 8 is visible on the left side of the code editor.

Output:

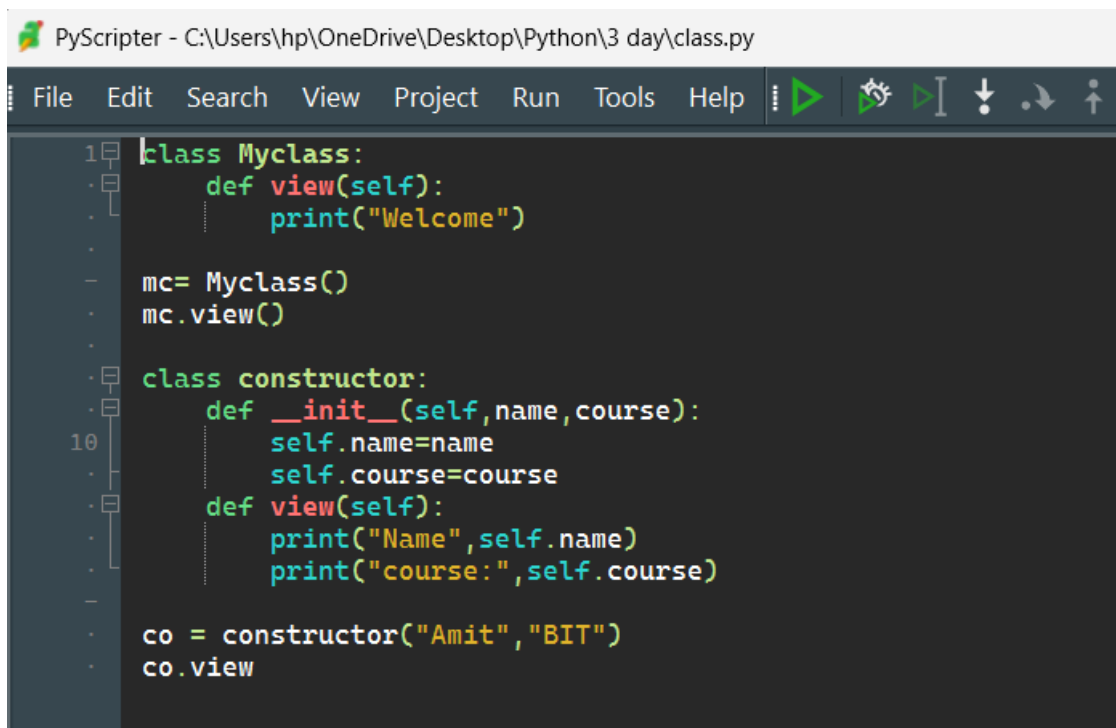
A screenshot of the Python Interpreter output window. It displays the following text:

```
*** Remote Interpreter Reinitialized ***
<built-in function pow>
1.7320508075688772
1.7320508075688772
>>>
```

Class and Object

A class is a user-defined blueprint or prototype from which objects are created. Classes provide a means of bundling data and functionality together. Creating a new class creates a new type of object, allowing new instances of that type to be made. Each class instance can have attributes attached to it to maintain its state. Class instances can also have methods (defined by their class) for modifying their state.

Example:



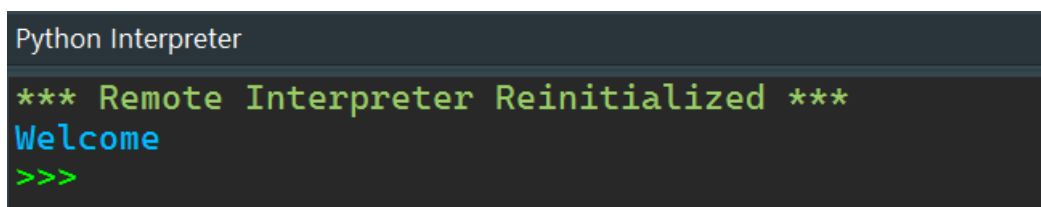
```
PyScripter - C:\Users\hp\OneDrive\Desktop\Python\3 day\class.py
File Edit Search View Project Run Tools Help
class Myclass:
    def view(self):
        print("Welcome")

mc= Myclass()
mc.view()

class constructor:
    def __init__(self,name,course):
        self.name=name
        self.course=course
    def view(self):
        print("Name",self.name)
        print("course:",self.course)

co = constructor("Amit","BIT")
co.view
```

Output:

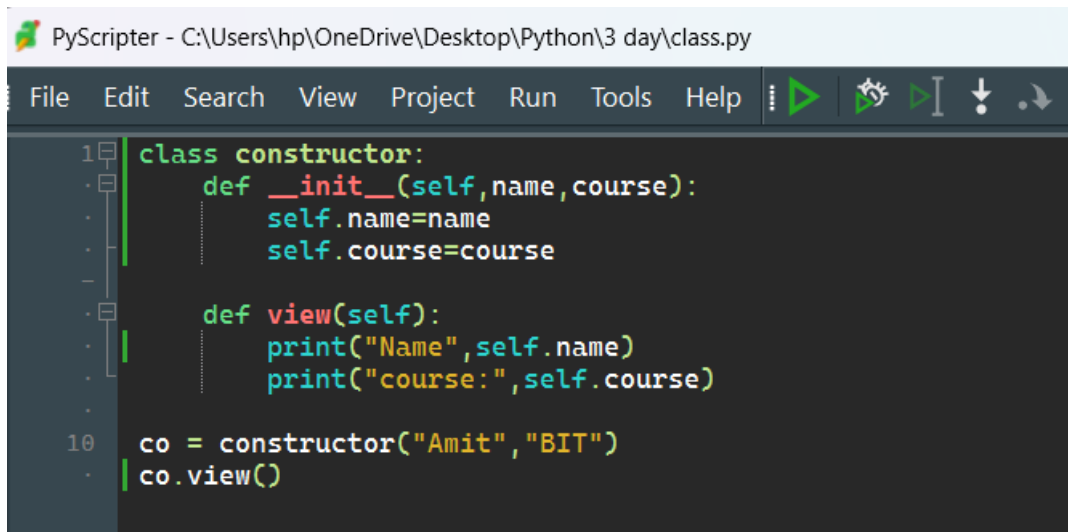


```
Python Interpreter
*** Remote Interpreter Reinitialized ***
Welcome
>>>
```

Constructor

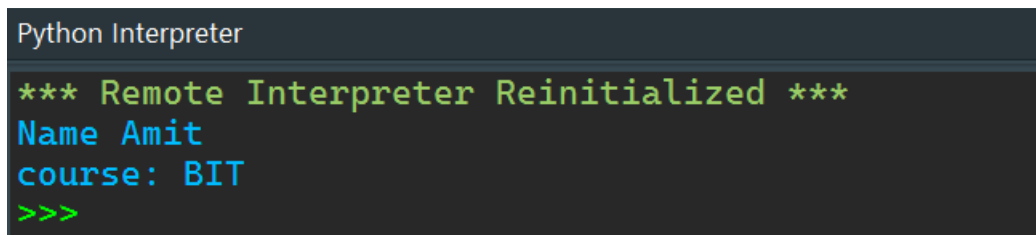
In Python, a constructor is a special method that is automatically called when an object of a class is created. The constructor method in Python is named `__init__`. The primary purpose of the constructor is to initialize the object's attributes and perform any setup or initialization tasks required.

Example:



```
PyScripter - C:\Users\hp\OneDrive\Desktop\Python\3 day\class.py
File Edit Search View Project Run Tools Help
1 class constructor:
2     def __init__(self, name, course):
3         self.name = name
4         self.course = course
5
6     def view(self):
7         print("Name", self.name)
8         print("course:", self.course)
9
10 co = constructor("Amit", "BIT")
11 co.view()
```

Output:



```
Python Interpreter
*** Remote Interpreter Reinitialized ***
Name Amit
course: BIT
>>>
```

