CodroidHub summer training

Title :

- Break & Continue
- Predefined function & User defined functions
- Find simple interest using user defined functions & Sum of 2 numbers
- Default arguments
- Variable length
- Recursive function
- Return type function
- Package
- Class & Object
- Constructor

Submitted by: DRISHTI GUPTA

ROLL NO: (2322825)

BRANCH: AI&ML

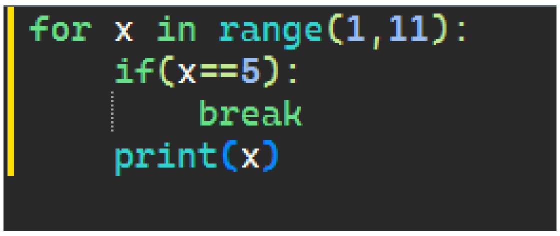
SUBMITTED TO: Mr. DEVASHISH SIR

(FOUNDER, CodroidHub Private Limited)

Break & Continue

- In Python, the break statement is used to terminate the execution of a loop prematurely. When a break statement is encountered inside a loop (either for or while), the loop stops running, and the control flow jumps to the statement immediately following the loop.

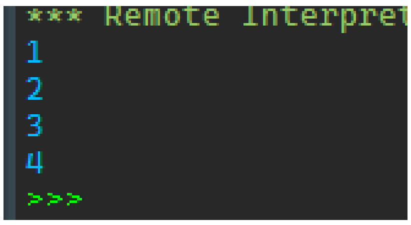
Example :



```
for x in range(1,11):
    if(x==5):
        break
    print(x)
```

Output :

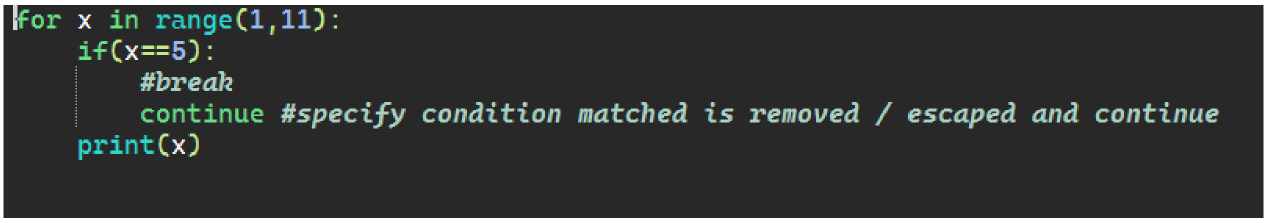
x = 1, x is not 5, so print(x) outputs 1.
x = 2, x is not 5, so print(x) outputs 2.
x = 3, x is not 5, so print(x) outputs 3.
x = 4, x is not 5, so print(x) outputs 4.
x = 5, x equals 5, so the break statement is executed, terminating the loop.



```
*** Remote Interpret
1
2
3
4
>>>
```

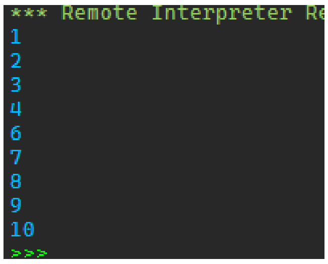
- In Python ,the Continue statement is used to skip the rest of the code inside a loop for the current iteration and move on to the next iteration of the loop. Unlike break, which terminates the loop entirely, continue only skips the current iteration.

Example :



```
for x in range(1,11):
    if(x==5):
        #break
        continue #specify condition matched is removed / escaped and continue
    print(x)
```

Output :



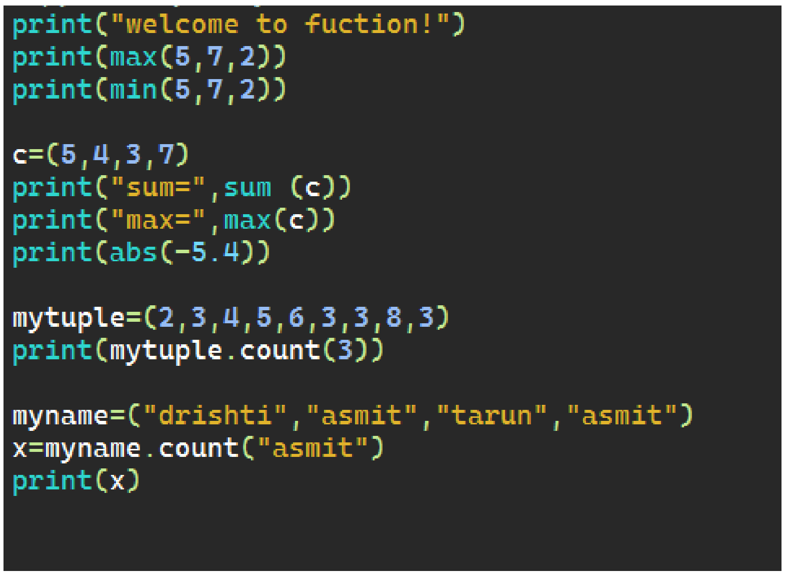
```
*** Remote Interpreter Re
1
2
3
4
6
7
8
9
10
>>>
```

Predefined function & User defined functions

Predefined Functions

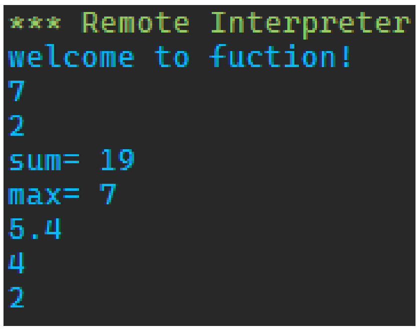
Predefined functions, also known as built-in functions, are functions that are already defined in Python's standard library. These functions are readily available for use without any additional coding or importing of libraries.

Example :



```
print("welcome to fuction!")
print(max(5,7,2))
print(min(5,7,2))

c=(5,4,3,7)
print("sum=",sum (c))
print("max=",max(c))
print(abs(-5.4))

mytuple=(2,3,4,5,6,3,3,8,3)
print(mytuple.count(3))

myname=("drishti","asmit","tarun","asmit")
x=myname.count("asmit")
print(x)
```

Output :

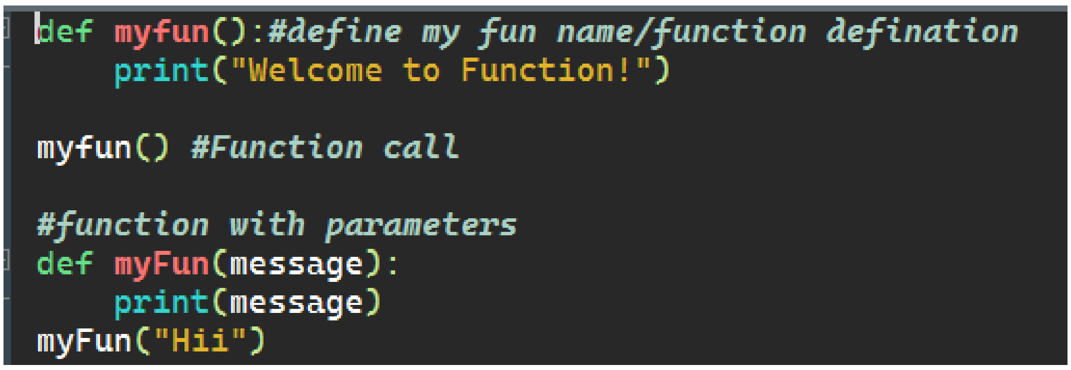


```
*** Remote Interpreter
welcome to fuction!
7
2
sum= 19
max= 7
5.4
4
2
```

User-Defined Functions

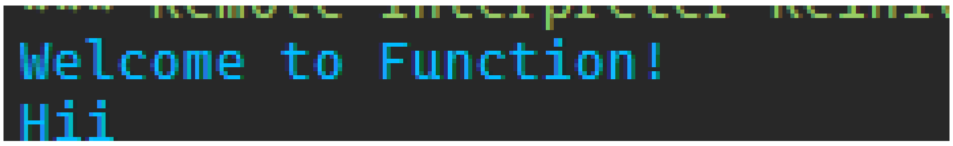
User-defined functions are functions that you define yourself to perform specific tasks. You create these functions using the def keyword, followed by the function name, parentheses (), and a colon :. The code block within the function is indented.

Example :



```
def myfun():#define my fun name/function defination
    print("Welcome to Function!")

myfun() #Function call

#function with parameters
def myFun(message):
    print(message)
myFun("Hii")
```

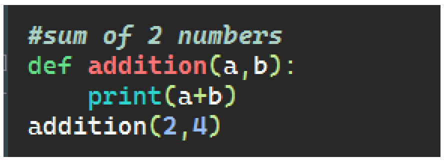
Output :



```
Welcome to Function!
Hii
```

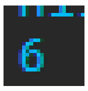
Find simple interest using user defined functions & Sum of numbers

Code : sum of numbers



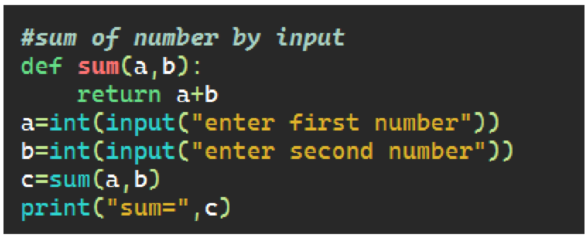
```
#sum of 2 numbers
def addition(a,b):
    print(a+b)
addition(2,4)
```

Output :



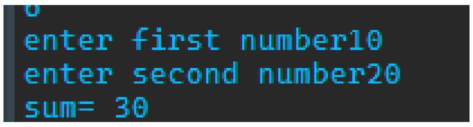
```
6
```

Code : sum of number by input method



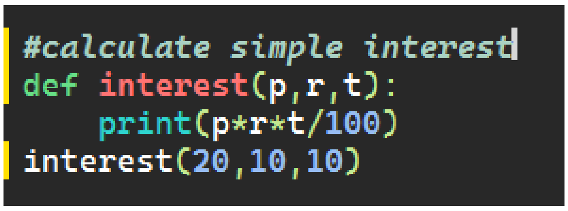
```
#sum of number by input
def sum(a,b):
    return a+b
a=int(input("enter first number"))
b=int(input("enter second number"))
c=sum(a,b)
print("sum=",c)
```

Output :



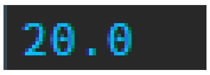
```
enter first number10
enter second number20
sum= 30
```

Code : simple interest



```
#calculate simple interest
def interest(p,r,t):
    print(p*r*t/100)
interest(20,10,10)
```

Output :

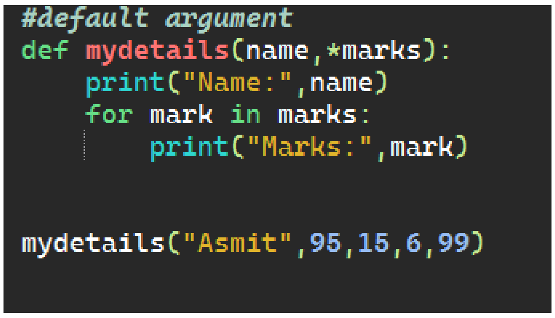


```
20.0
```

Default arguments

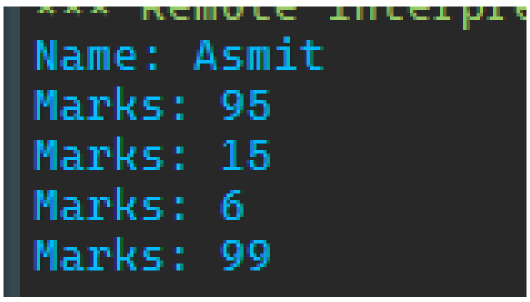
Default arguments in Python are values provided in a function definition that are used if no corresponding argument is passed during the function call. Default arguments are specified in the function definition by assigning a value to the parameter. When a function with default arguments is called, the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

Example :



```python
#default argument
def mydetails(name,*marks):
    print("Name:",name)
    for mark in marks:
        print("Marks:",mark)


mydetails("Asmit",95,15,6,99)
```

Output :



```
*** Remote Interpre
Name: Asmit
Marks: 95
Marks: 15
Marks: 6
Marks: 99
```

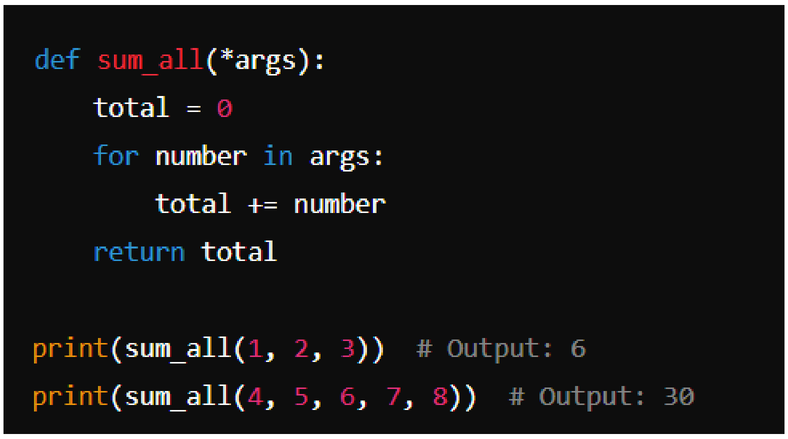
Variable length

In Python, you can define functions that accept a variable number of arguments using special syntax. This allows your functions to handle more flexible inputs. There are two main ways to achieve this: using *args for non-keyword variable-length arguments and **kwargs for keyword variable-length arguments.

1. `*args` (Non-keyword Variable-length Arguments)

The `*args` syntax allows you to pass a variable number of non-keyword arguments to a function. Inside the function, `args` is a tuple containing all the arguments passed to the function.

Example:



```python
def sum_all(*args):
    total = 0
    for number in args:
        total += number
    return total

print(sum_all(1, 2, 3))  # Output: 6
print(sum_all(4, 5, 6, 7, 8))  # Output: 30
```

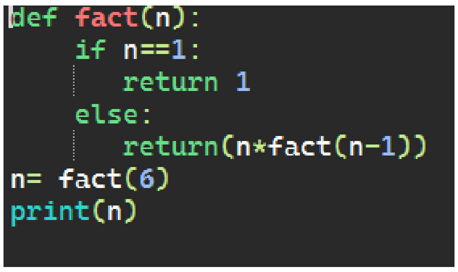
Recursive function

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

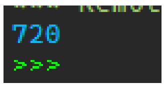
1. **Base Case**: The condition under which the recursion stops. Without a base case, the function would call itself indefinitely, leading to a stack overflow.
2. **Recursive Case**: The part of the function where it calls itself with a smaller or simpler input.

Example: Factorial Function



```python
def fact(n):
    if n==1:
        return 1
    else:
        return(n*fact(n-1))
n= fact(6)
print(n)
```

Output :

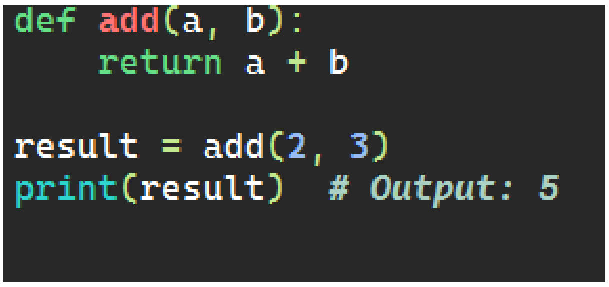


```
720
>>>
```

Return type function

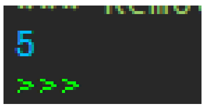
In Python, the return type of a function is the type of value that the function returns. Functions can return various types of values, including integers, floats, strings, lists, dictionaries, tuples, or even other functions. The return statement is used to exit a function and return a value from the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

Example :



```python
def add(a, b):
    return a + b

result = add(2, 3)
print(result)  # Output: 5
```

Output :

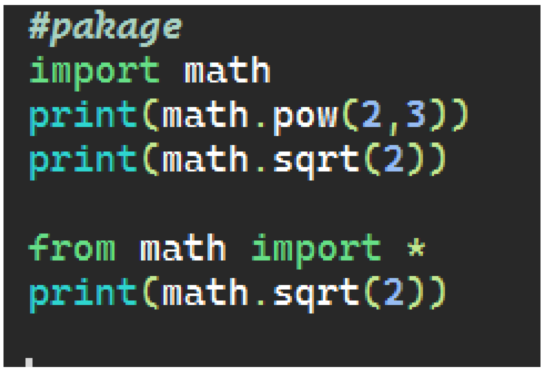


```
5
>>>
```

Package

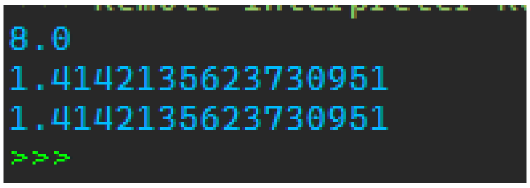
In Python, a package is a way of organizing related modules into a directory hierarchy. A package is essentially a directory that contains a special file named __init__.py (which can be empty), along with other module files and sub-packages. Packages allow you to structure your Python code into manageable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

Example :



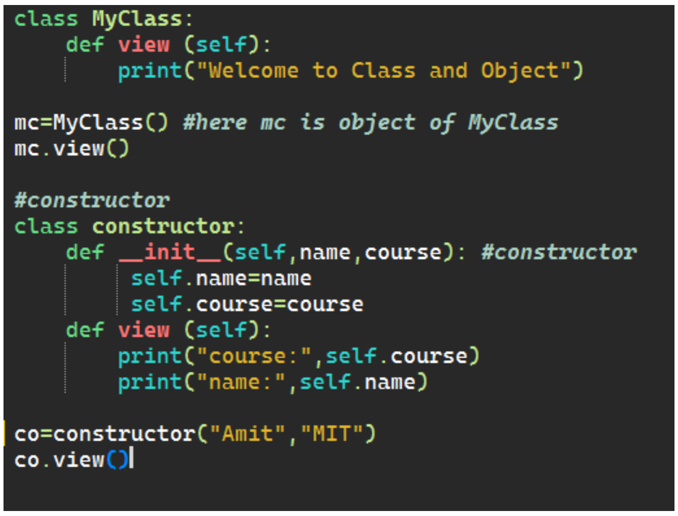
```python
#pakage
import math
print(math.pow(2,3))
print(math.sqrt(2))

from math import *
print(math.sqrt(2))
```

Output :



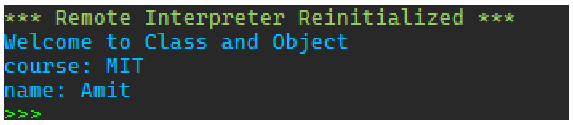
```
8.0
1.4142135623730951
1.4142135623730951
>>>
```

Class & Object



```python
class MyClass:
    def view (self):
        print("Welcome to Class and Object")

mc=MyClass() #here mc is object of MyClass
mc.view()

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

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

Output :

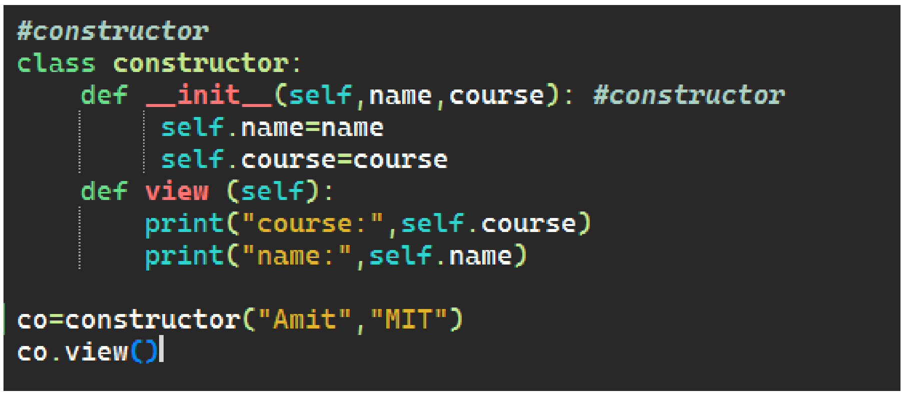


```
*** Remote Interpreter Reinitialized ***
Welcome to Class and Object
course: MIT
name: Amit
>>>
```

Constructor

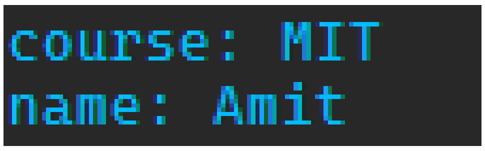
In Python, a constructor is a special method that is automatically called when an object of a class is created. The constructor method in Python is named __init__. The primary purpose of the constructor is to initialize the object's attributes and perform any setup or initialization tasks required.

Example :



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

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

Output :



```
course: MIT
name: Amit
```