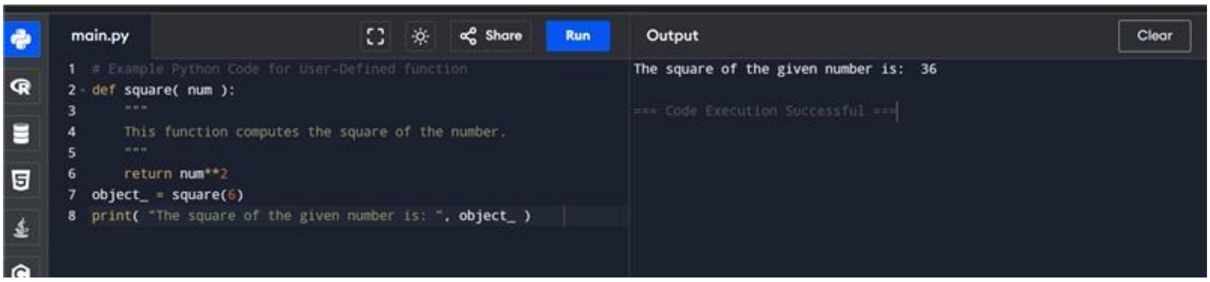
KARANAM YESWANTH TEJA

289224

1) PYTHON FUNCTIONS

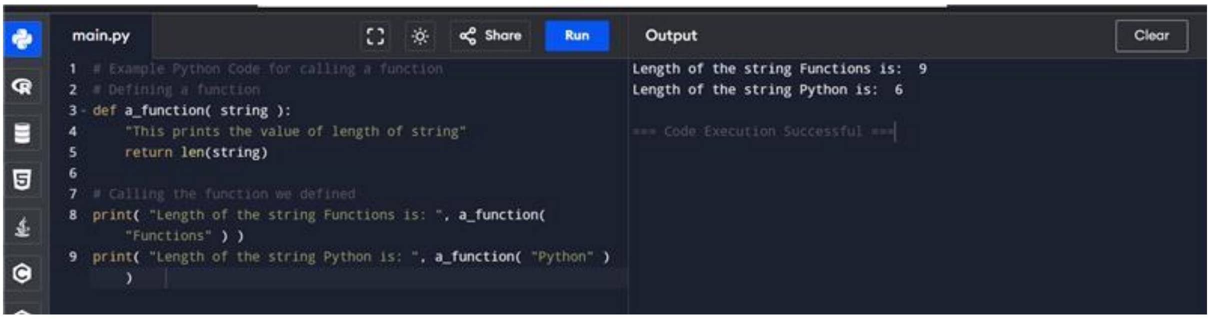
1.1) Example Python Code for User-Defined function



```
# Example Python Code for User-Defined function
def square( num ):
    """
    This function computes the square of the number.
    """
    return num**2
object_ = square(6)
print( "The square of the given number is: ", object_ )
```

```
The square of the given number is:  36

=== Code Execution Successful ===
```

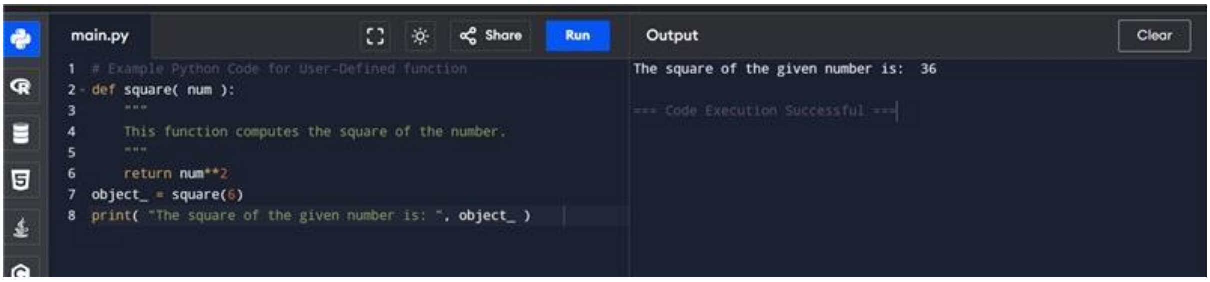
1.2) Example Python Code for calling a function



```
# Example Python Code for calling a function
# Defining a function
def a_function( string ):
    "This prints the value of length of string"
    return len(string)

# Calling the function we defined
print( "Length of the string Functions is: ", a_function( "Functions" ) )
print( "Length of the string Python is: ", a_function( "Python" ) )
```

```
Length of the string Functions is:  9
Length of the string Python is:  6

=== Code Execution Successful ===
```

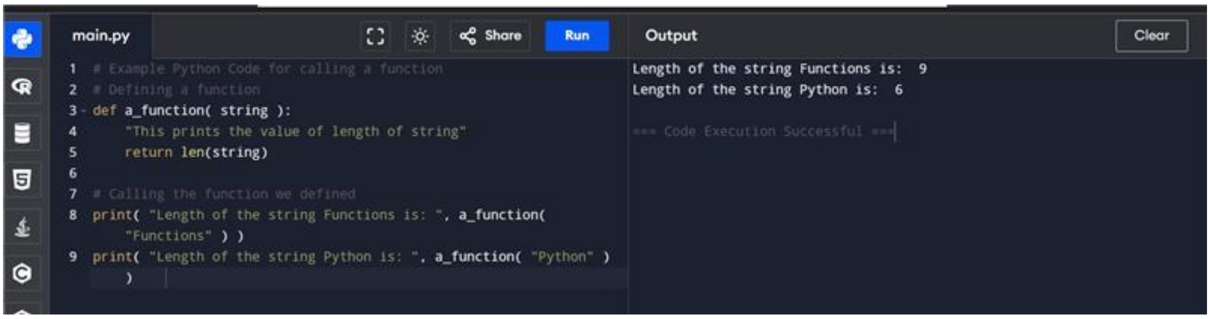
1.3) Example Python Code for User-Defined function



```
# Example Python Code for User-Defined function
def square( num ):
    """
    This function computes the square of the number.
    """
    return num**2
object_ = square(6)
print( "The square of the given number is: ", object_ )
```

```
The square of the given number is:  36

=== Code Execution Successful ===
```

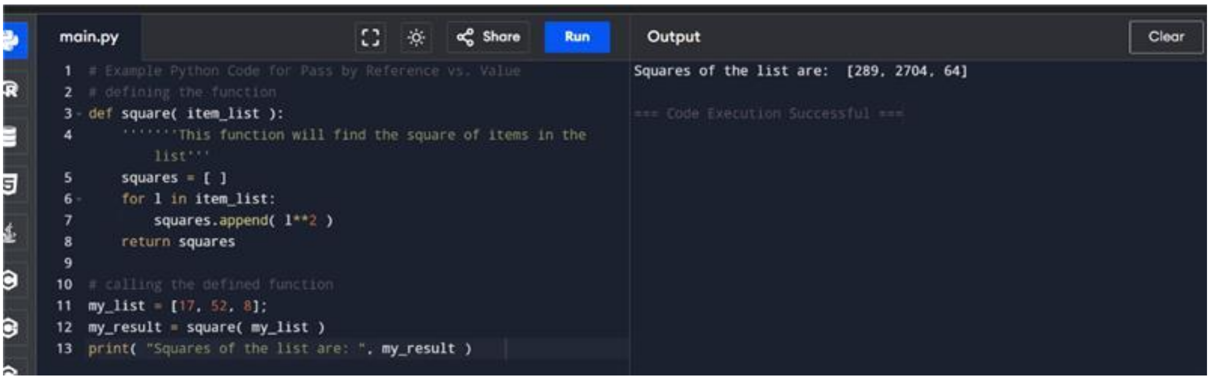
1.4) Example Python Code for calling a function



```python
# Example Python Code for calling a function
# Defining a function
def a_function( string ):
    "This prints the value of length of string"
    return len(string)

# Calling the function we defined
print( "Length of the string Functions is: ", a_function( "Functions" ) )
print( "Length of the string Python is: ", a_function( "Python" ) )
```

```
Length of the string Functions is:  9
Length of the string Python is:  6

=== Code Execution Successful ===
```

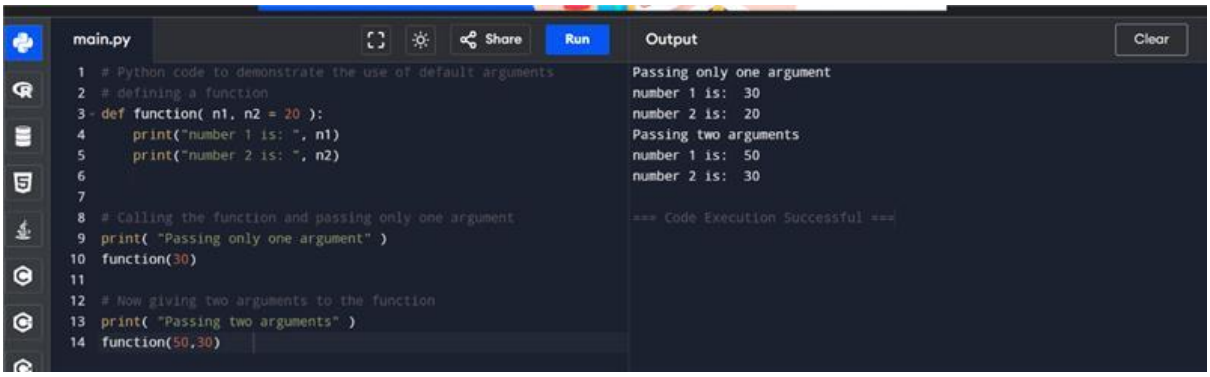
1.5) Pass by Reference vs. Pass by Value



```python
# Example Python Code for Pass by Reference vs. Value
# defining the function
def square( item_list ):
    '''''''This function will find the square of items in the list'''
    squares = [ ]
    for l in item_list:
        squares.append( l**2 )
    return squares

# calling the defined function
my_list = [17, 52, 8];
my_result = square( my_list )
print( "Squares of the list are: ", my_result )
```

```
Squares of the list are:  [289, 2704, 64]

=== Code Execution Successful ===
```

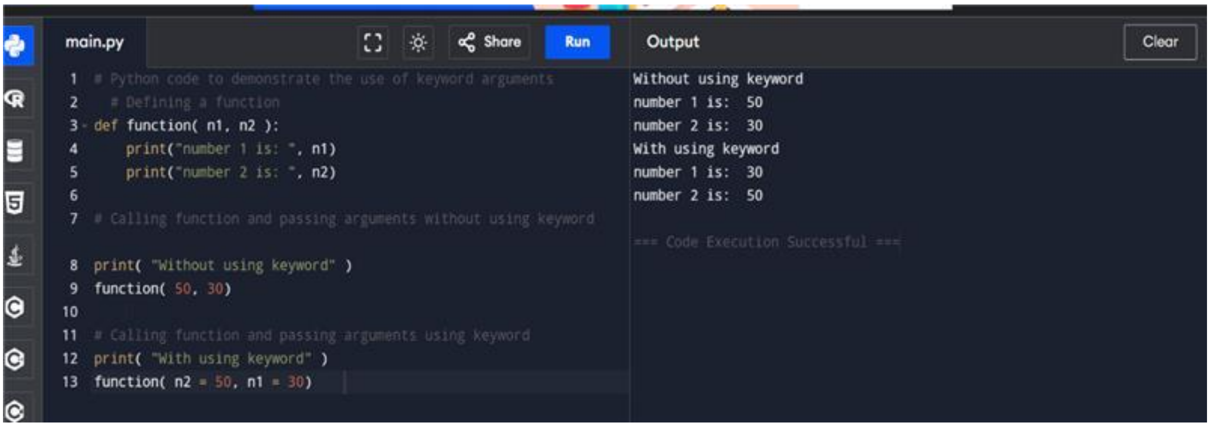
1.6) Default Arguments



```python
# Python code to demonstrate the use of default arguments
# defining a function
def function( n1, n2 = 20 ):
    print("number 1 is: ", n1)
    print("number 2 is: ", n2)


# Calling the function and passing only one argument
print( "Passing only one argument" )
function(30)

# Now giving two arguments to the function
print( "Passing two arguments" )
function(50,30)
```

```
Passing only one argument
number 1 is:  30
number 2 is:  20
Passing two arguments
number 1 is:  50
number 2 is:  30

=== Code Execution Successful ===
```

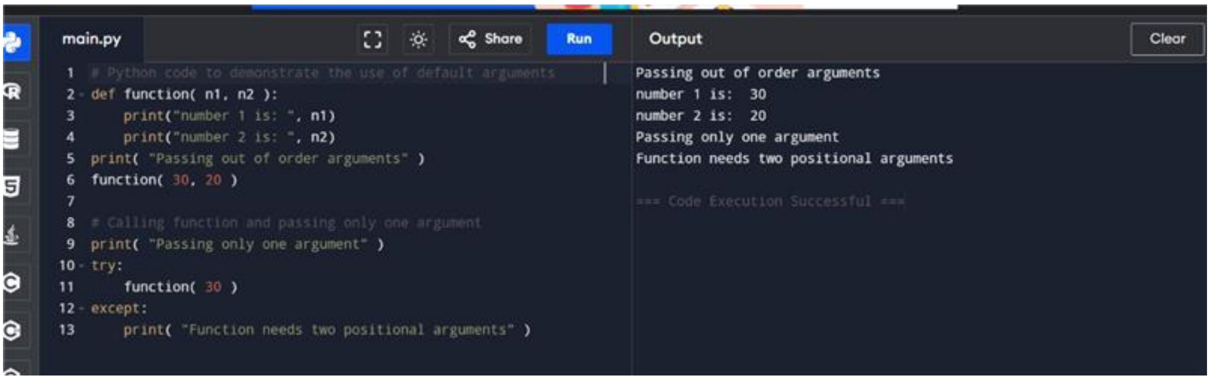
1.7) Keyword Arguments



```python
# Python code to demonstrate the use of keyword arguments
  # Defining a function
def function( n1, n2 ):
    print("number 1 is: ", n1)
    print("number 2 is: ", n2)

# Calling function and passing arguments without using keyword

print( "Without using keyword" )
function( 50, 30)

# Calling function and passing arguments using keyword
print( "With using keyword" )
function( n2 = 50, n1 = 30)
```

```
Without using keyword
number 1 is:  50
number 2 is:  30
With using keyword
number 1 is:  30
number 2 is:  50

=== Code Execution Successful ===
```

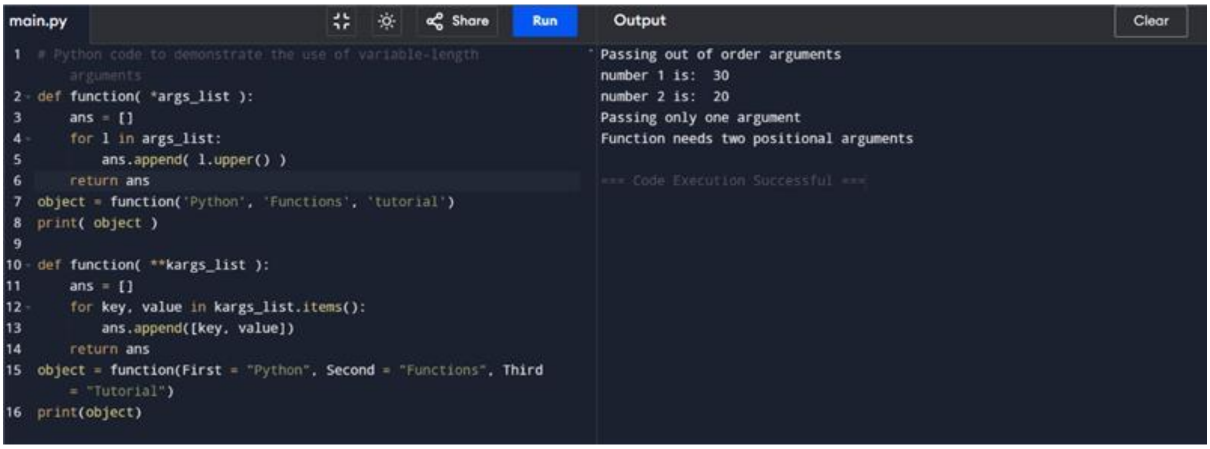
1.8) Required Arguments



```python
# Python code to demonstrate the use of default arguments
def function( n1, n2 ):
    print("number 1 is: ", n1)
    print("number 2 is: ", n2)
print( "Passing out of order arguments" )
function( 30, 20 )

# Calling function and passing only one argument
print( "Passing only one argument" )
try:
    function( 30 )
except:
    print( "Function needs two positional arguments" )
```

```
Passing out of order arguments
number 1 is:  30
number 2 is:  20
Passing only one argument
Function needs two positional arguments

=== Code Execution Successful ===
```

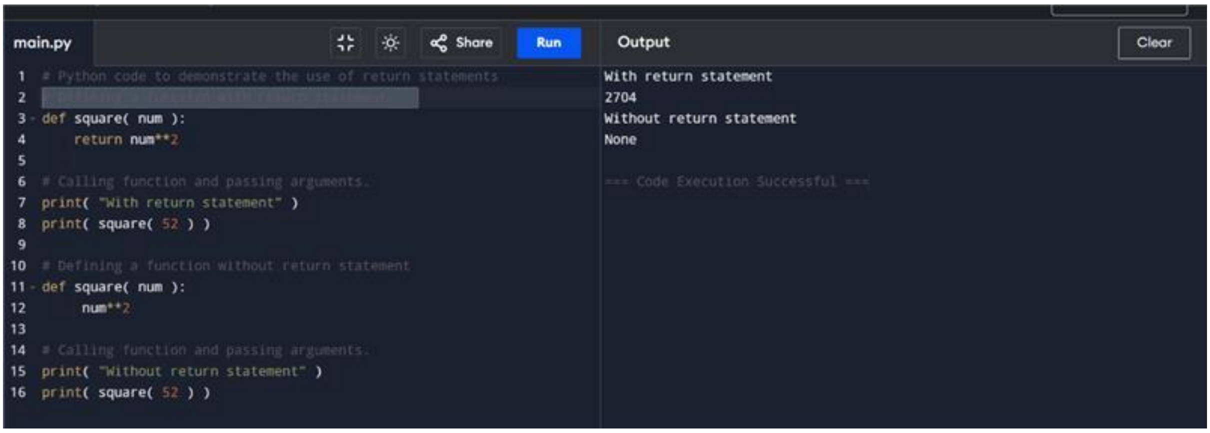
1.9) Variable-Length Arguments



```python
# Python code to demonstrate the use of variable-length arguments
def function( *args_list ):
    ans = []
    for l in args_list:
        ans.append( l.upper() )
    return ans
object = function('Python', 'Functions', 'tutorial')
print( object )

def function( **kargs_list ):
    ans = []
    for key, value in kargs_list.items():
        ans.append([key, value])
    return ans
object = function(First = "Python", Second = "Functions", Third = "Tutorial")
print(object)
```

```
Passing out of order arguments
number 1 is:  30
number 2 is:  20
Passing only one argument
Function needs two positional arguments

=== Code Execution Successful ===
```

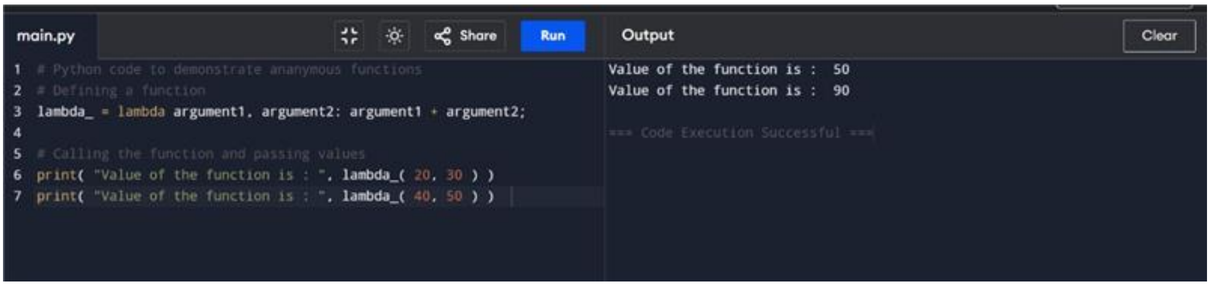
1.10) Return statement



```python
# Python code to demonstrate the use of return statements
# Defining a function with return statement
def square( num ):
    return num**2

# Calling function and passing arguments.
print( "With return statement" )
print( square( 52 ) )

# Defining a function without return statement
def square( num ):
     num**2

# Calling function and passing arguments.
print( "Without return statement" )
print( square( 52 ) )
```

```
With return statement
2704
Without return statement
None

=== Code Execution Successful ===
```

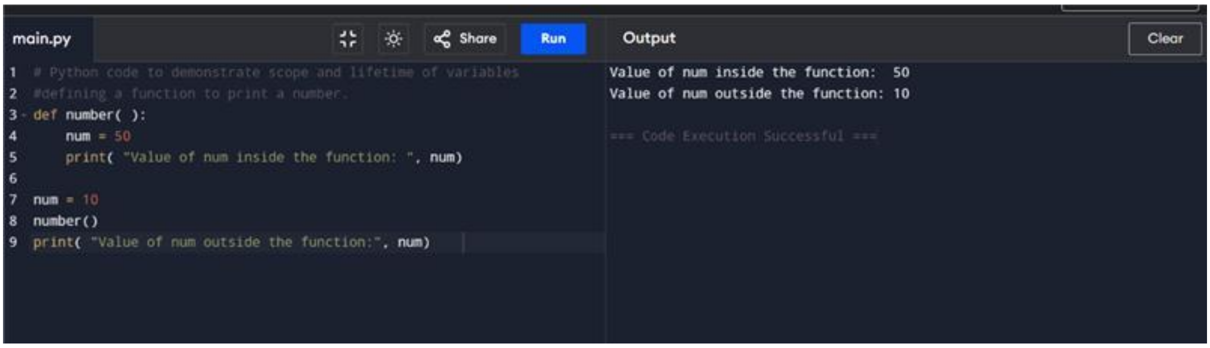
1.11) The Anonymous Functions



```python
# Python code to demonstrate ananymous functions
# Defining a function
lambda_ = lambda argument1, argument2: argument1 + argument2;

# Calling the function and passing values
print( "Value of the function is : ", lambda_( 20, 30 ) )
print( "Value of the function is : ", lambda_( 40, 50 ) )
```

```
Value of the function is :  50
Value of the function is :  90

=== Code Execution Successful ===
```

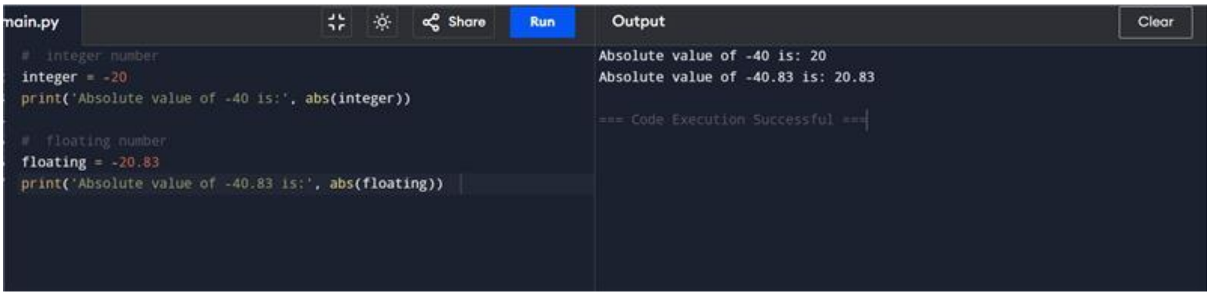
1.12) Scope and Lifetime of Variables



```python
# Python code to demonstrate scope and lifetime of variables
#defining a function to print a number.
def number( ):
    num = 50
    print( "Value of num inside the function: ", num)

num = 10
number()
print( "Value of num outside the function:", num)
```

```
Value of num inside the function:  50
Value of num outside the function: 10

=== Code Execution Successful ===
```

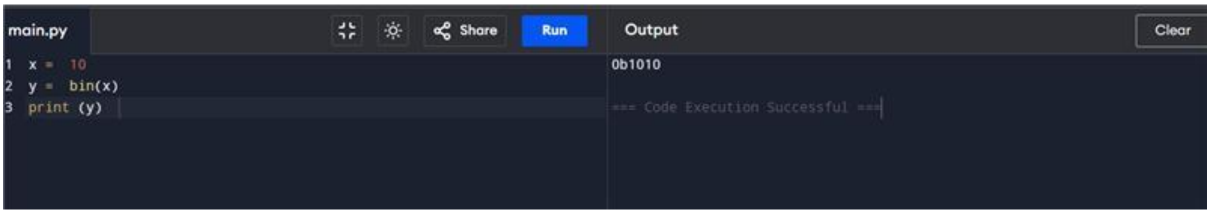
1.13) Python abs() Function



```python
#  integer number
integer = -20
print('Absolute value of -40 is:', abs(integer))

#  floating number
floating = -20.83
print('Absolute value of -40.83 is:', abs(floating))
```

```
Absolute value of -40 is: 20
Absolute value of -40.83 is: 20.83

=== Code Execution Successful ===
```

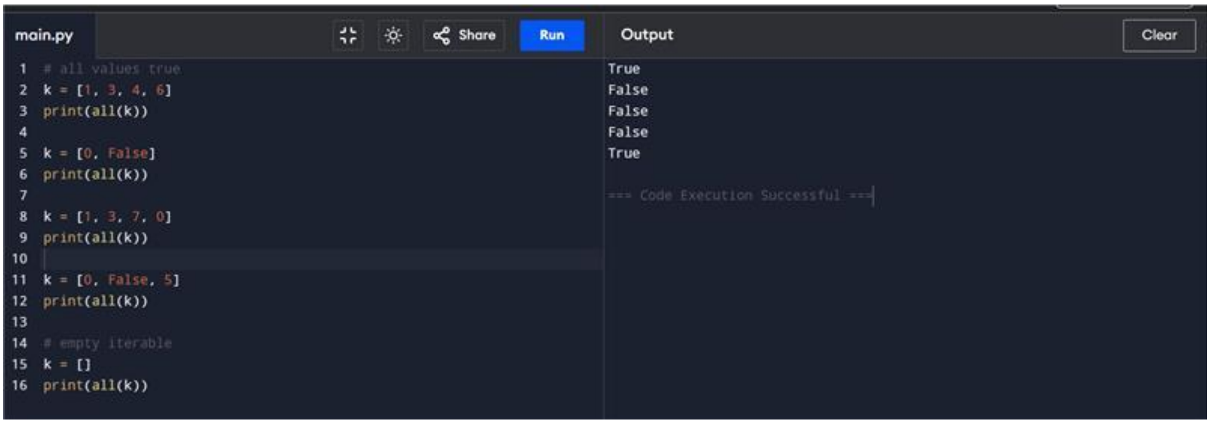
1.14) Python bin() Function



```python
x = 10
y = bin(x)
print (y)
```

```
0b1010

=== Code Execution Successful ===
```

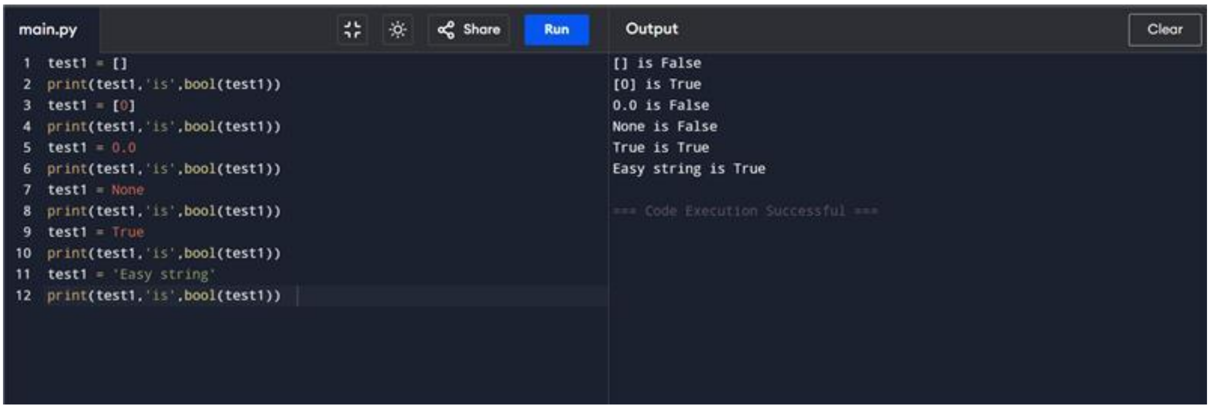
1.15) Python all() Function



```python
# all values true
k = [1, 3, 4, 6]
print(all(k))

k = [0, False]
print(all(k))

k = [1, 3, 7, 0]
print(all(k))

k = [0, False, 5]
print(all(k))

# empty iterable
k = []
print(all(k))
```

```
True
False
False
False
True

=== Code Execution Successful ===
```

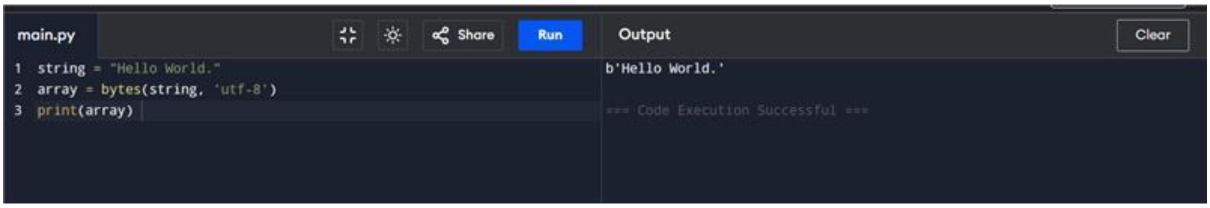
1.16) Python bool()



```python
test1 = []
print(test1,'is',bool(test1))
test1 = [0]
print(test1,'is',bool(test1))
test1 = 0.0
print(test1,'is',bool(test1))
test1 = None
print(test1,'is',bool(test1))
test1 = True
print(test1,'is',bool(test1))
test1 = 'Easy string'
print(test1,'is',bool(test1))
```

```
[] is False
[0] is True
0.0 is False
None is False
True is True
Easy string is True

=== Code Execution Successful ===
```

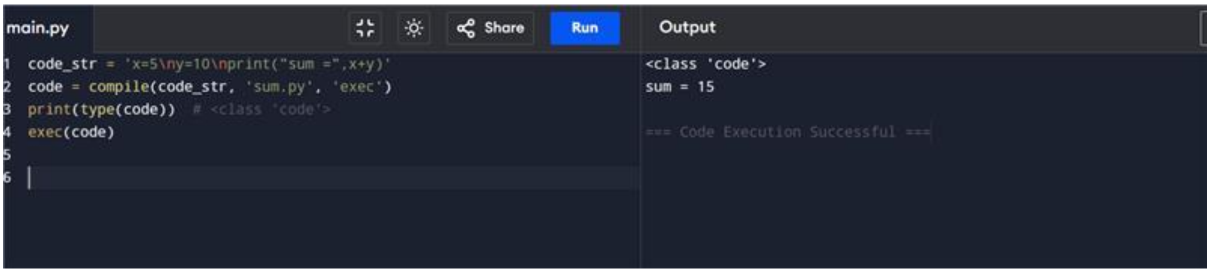
1.17) Python bytes()



```python
string = "Hello World."
array = bytes(string, 'utf-8')
print(array)
```

```
b'Hello World.'

=== Code Execution Successful ===
```

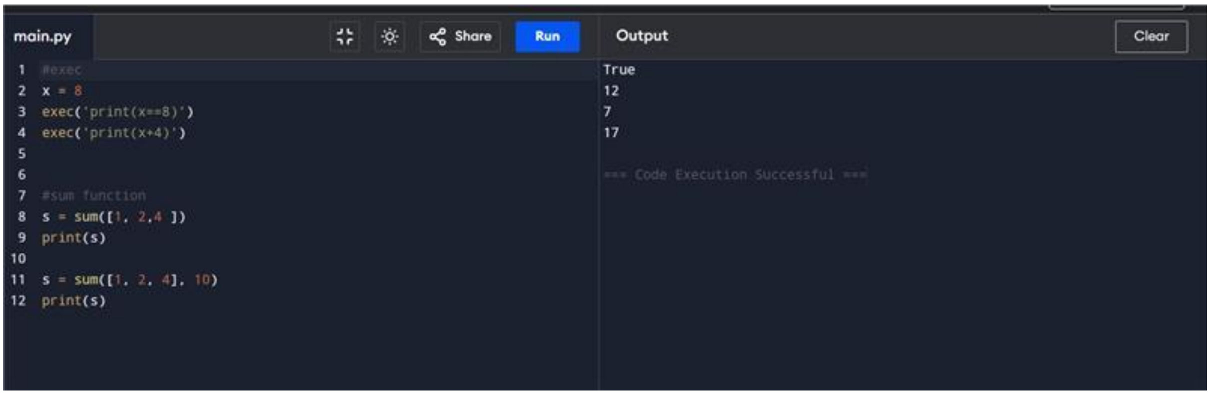
1.18) Python compile() Function



```python
code_str = 'x=5\ny=10\nprint("sum =",x+y)'
code = compile(code_str, 'sum.py', 'exec')
print(type(code))  # <class 'code'>
exec(code)
```

```
<class 'code'>
sum = 15

=== Code Execution Successful ===
```

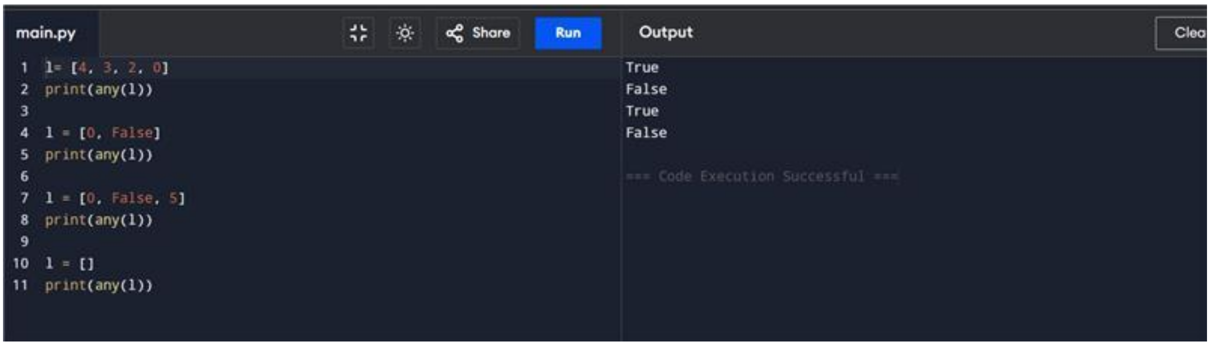
1.19) Python exec() and sum() Function Example



```python
#exec
x = 8
exec('print(x==8)')
exec('print(x+4)')


#sum function
s = sum([1, 2,4 ])
print(s)

s = sum([1, 2, 4], 10)
print(s)
```

```
True
12
7
17

=== Code Execution Successful ===
```

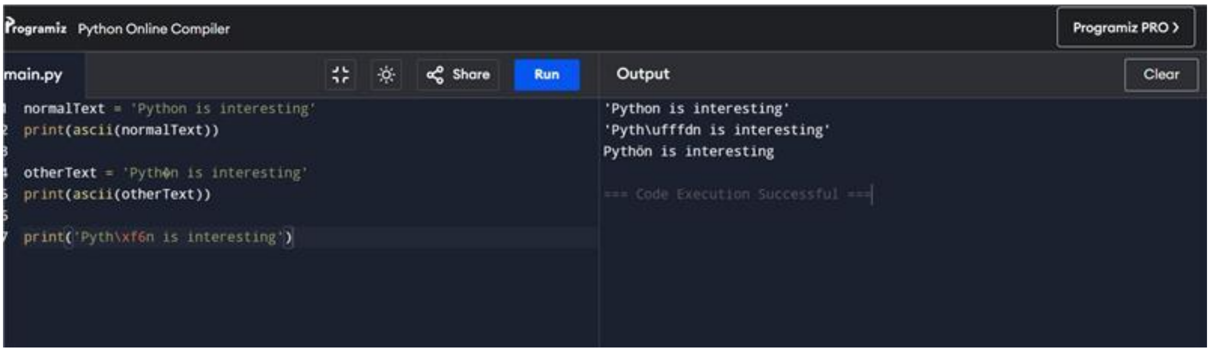
1.20) Any() function



```python
l= [4, 3, 2, 0]
print(any(l))

l = [0, False]
print(any(l))

l = [0, False, 5]
print(any(l))

l = []
print(any(l))
```

```
True
False
True
False

=== Code Execution Successful ===
```

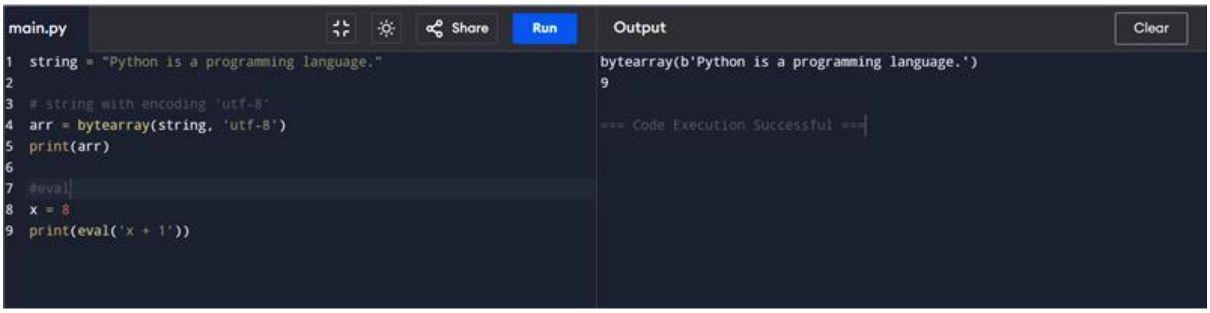
1.21) ASCII() function



```python
normalText = 'Python is interesting'
print(ascii(normalText))

otherText = 'Pyth�n is interesting'
print(ascii(otherText))

print('Pyth\xf6n is interesting')
```

```
'Python is interesting'
'Pyth\ufffdn is interesting'
Pythön is interesting

=== Code Execution Successful ===
```

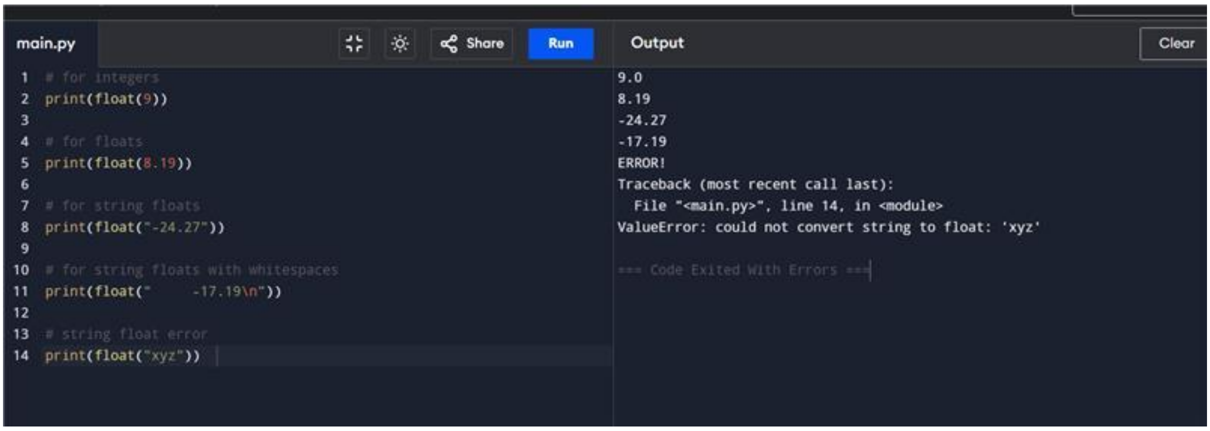
1.22) Byte array() and eval() functions



```python
string = "Python is a programming language."

# string with encoding 'utf-8'
arr = bytearray(string, 'utf-8')
print(arr)

#eval
x = 8
print(eval('x + 1'))
```

```
bytearray(b'Python is a programming language.')
9

=== Code Execution Successful ===
```

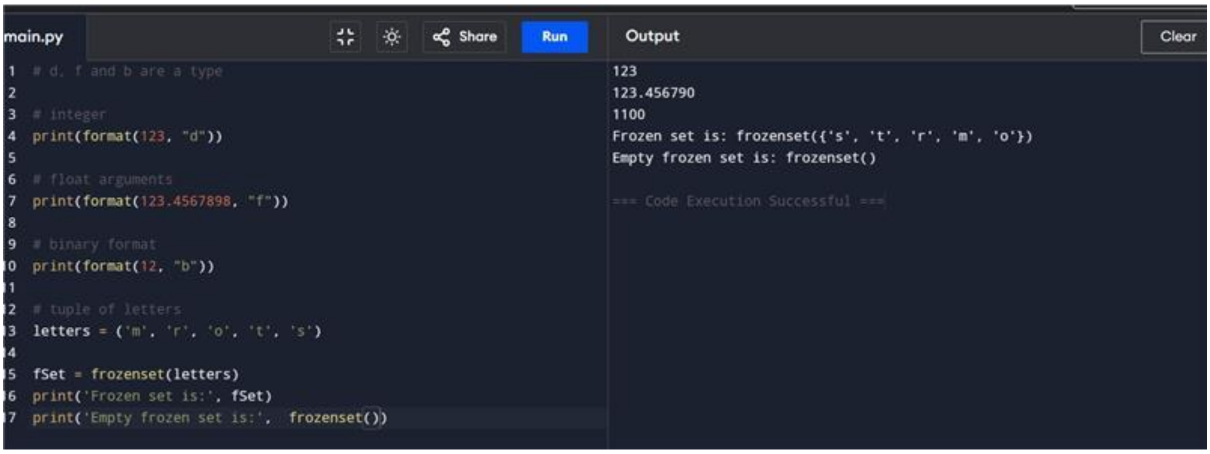
1.23) Float() function



```python
# for integers
print(float(9))

# for floats
print(float(8.19))

# for string floats
print(float("-24.27"))

# for string floats with whitespaces
print(float("     -17.19\n"))

# string float error
print(float("xyz"))
```

```
9.0
8.19
-24.27
-17.19
ERROR!
Traceback (most recent call last):
  File "<main.py>", line 14, in <module>
ValueError: could not convert string to float: 'xyz'

=== Code Exited With Errors ===
```

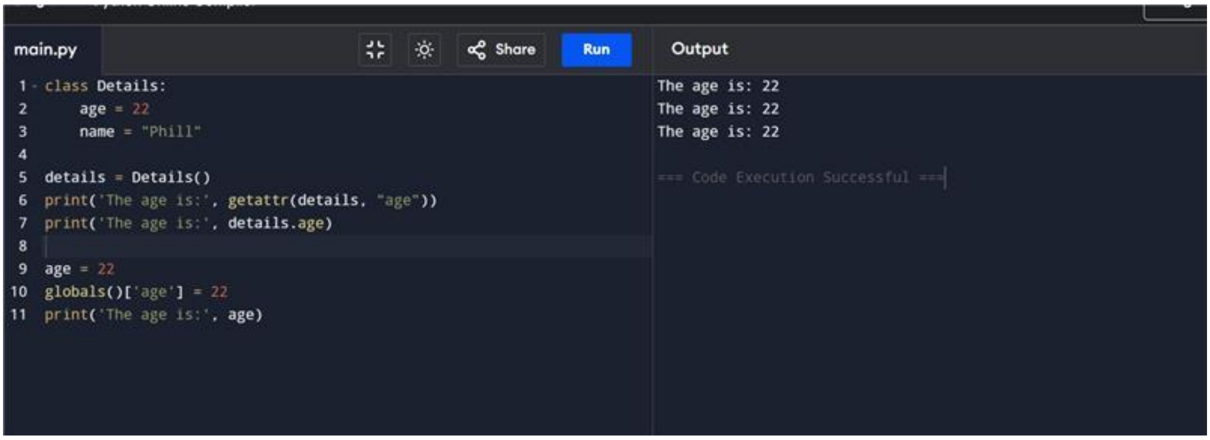
1.24) Frozen set() and format() fucntion



```python
# d, f and b are a type

# integer
print(format(123, "d"))

# float arguments
print(format(123.4567898, "f"))

# binary format
print(format(12, "b"))

# tuple of letters
letters = ('m', 'r', 'o', 't', 's')

fSet = frozenset(letters)
print('Frozen set is:', fSet)
print('Empty frozen set is:', frozenset())
```

```
123
123.456790
1100
Frozen set is: frozenset({'s', 't', 'r', 'm', 'o'})
Empty frozen set is: frozenset()

=== Code Execution Successful ===
```

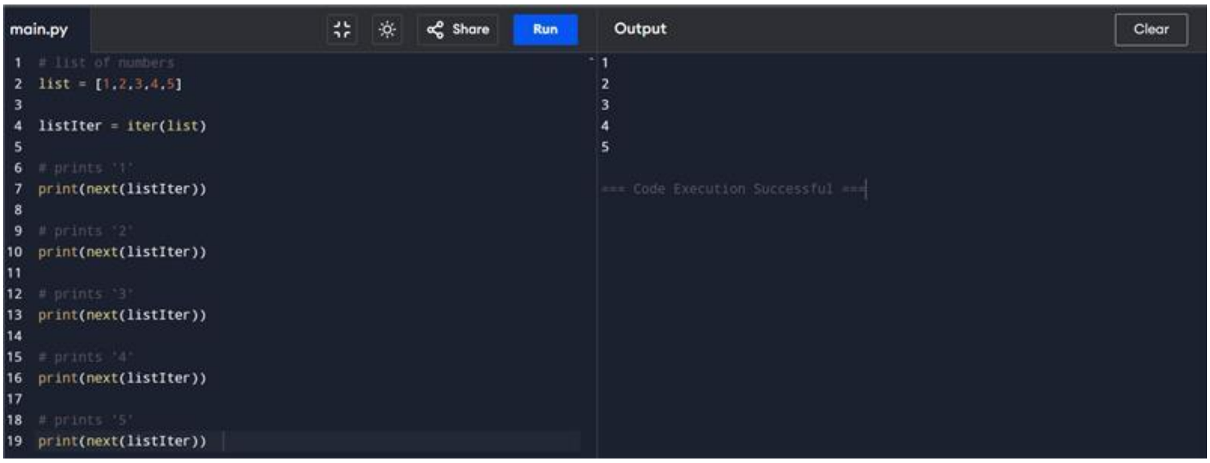
1.25) Gloabls() and getattr() function



```
class Details:
    age = 22
    name = "Phill"

details = Details()
print('The age is:', getattr(details, "age"))
print('The age is:', details.age)

age = 22
globals()['age'] = 22
print('The age is:', age)
```

```
The age is: 22
The age is: 22
The age is: 22

=== Code Execution Successful ===
```

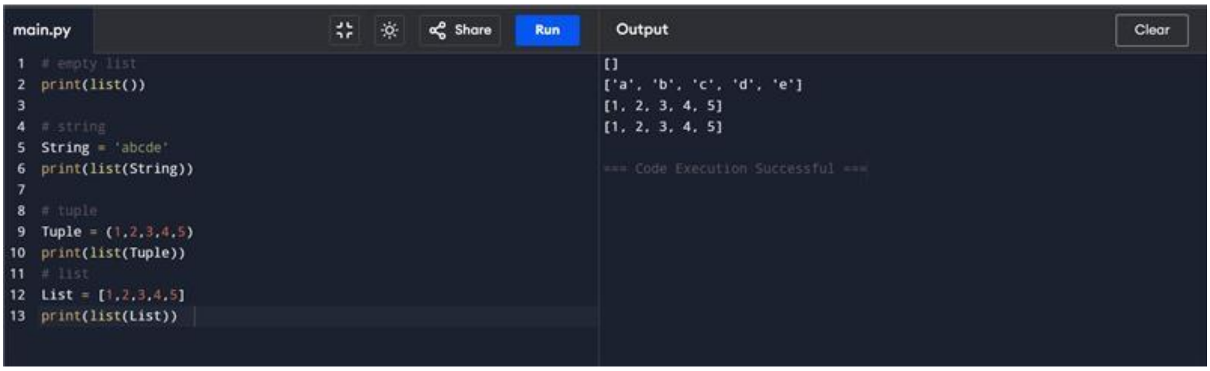
1.26) Python iter() Function



```
# list of numbers
list = [1,2,3,4,5]

listIter = iter(list)

# prints '1'
print(next(listIter))

# prints '2'
print(next(listIter))

# prints '3'
print(next(listIter))

# prints '4'
print(next(listIter))

# prints '5'
print(next(listIter))
```

```
1
2
3
4
5

=== Code Execution Successful ===
```

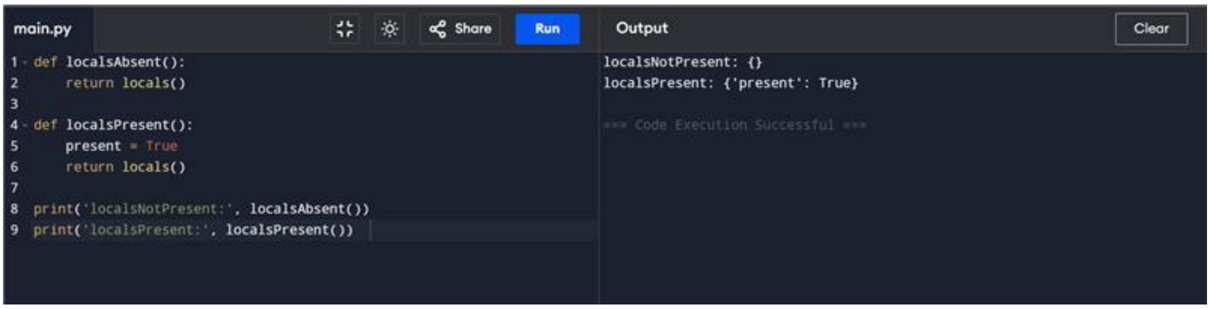
1.27) Python list()



```
# empty list
print(list())

# string
String = 'abcde'
print(list(String))

# tuple
Tuple = (1,2,3,4,5)
print(list(Tuple))
# list
List = [1,2,3,4,5]
print(list(List))
```

```
[]
['a', 'b', 'c', 'd', 'e']
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5]

=== Code Execution Successful ===
```

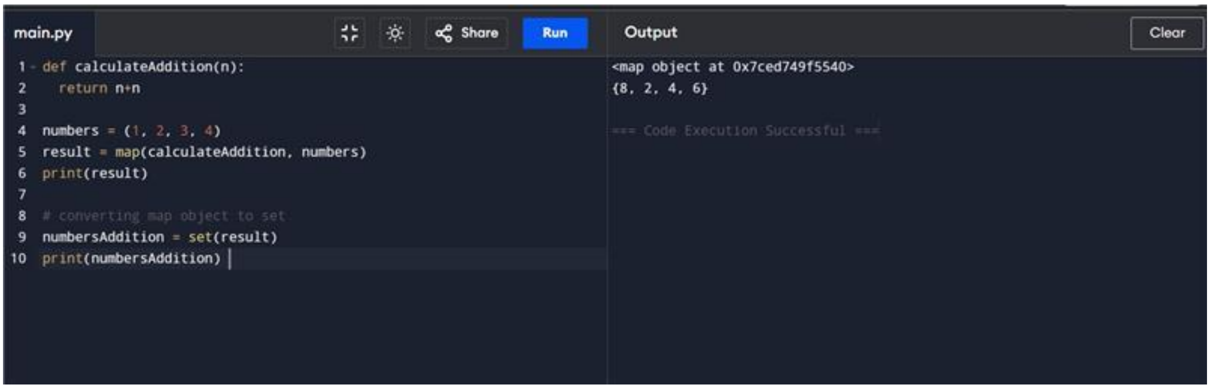
1.28) Python locals() Function Example



```python
def localsAbsent():
    return locals()

def localsPresent():
    present = True
    return locals()

print('localsNotPresent:', localsAbsent())
print('localsPresent:', localsPresent())
```

```
localsNotPresent: {}
localsPresent: {'present': True}

=== Code Execution Successful ===
```

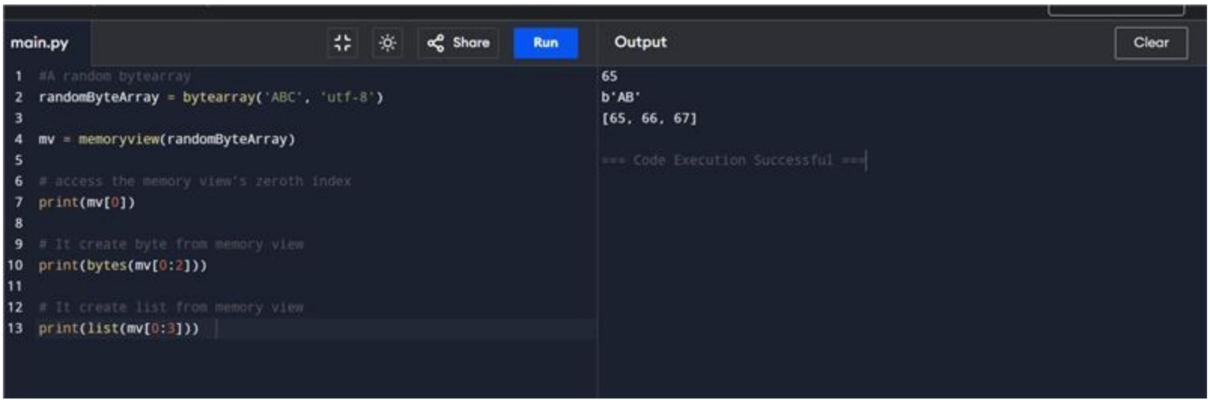
1.29) Map() function



```python
def calculateAddition(n):
  return n+n

numbers = (1, 2, 3, 4)
result = map(calculateAddition, numbers)
print(result)

# converting map object to set
numbersAddition = set(result)
print(numbersAddition)
```

```
<map object at 0x7ced749f5540>
{8, 2, 4, 6}

=== Code Execution Successful ===
```

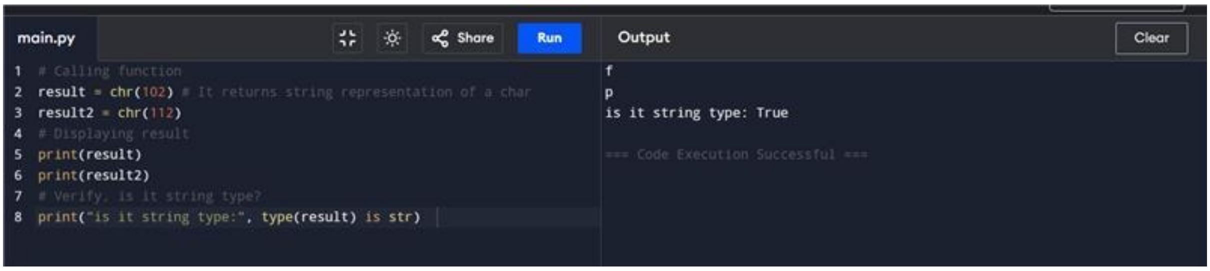
1.30) Python memoryview() Function



```python
#A random bytearray
randomByteArray = bytearray('ABC', 'utf-8')

mv = memoryview(randomByteArray)

# access the memory view's zeroth index
print(mv[0])

# It create byte from memory view
print(bytes(mv[0:2]))

# It create list from memory view
print(list(mv[0:3]))
```

```
65
b'AB'
[65, 66, 67]

=== Code Execution Successful ===
```

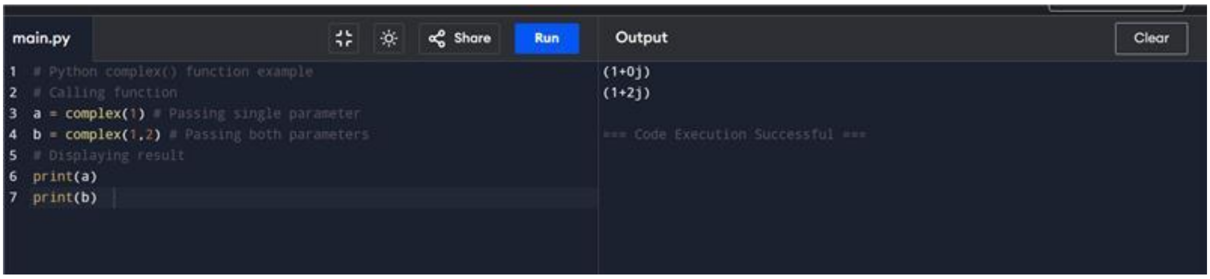
1.31) Python chr() Function



```python
# Calling function
result = chr(102) # It returns string representation of a char
result2 = chr(112)
# Displaying result
print(result)
print(result2)
# Verify, is it string type?
print("is it string type:", type(result) is str)
```

```
f
p
is it string type: True

=== Code Execution Successful ===
```

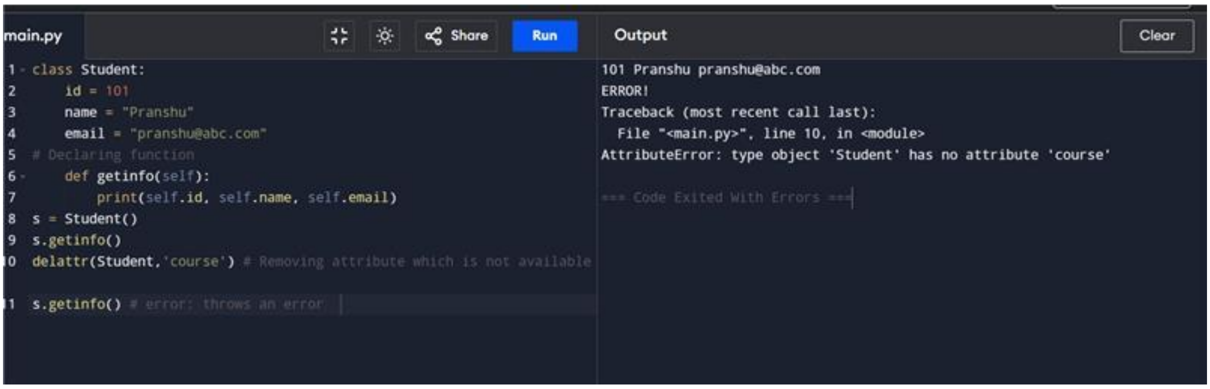
1.32) Python complex fun()



```python
# Python complex() function example
# Calling function
a = complex(1) # Passing single parameter
b = complex(1,2) # Passing both parameters
# Displaying result
print(a)
print(b)
```

```
(1+0j)
(1+2j)

=== Code Execution Successful ===
```

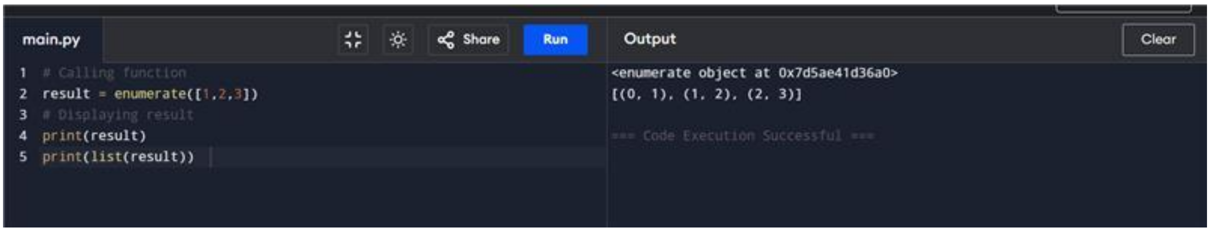
1.33) Python delattr() Function



```python
class Student:
    id = 101
    name = "Pranshu"
    email = "pranshu@abc.com"
# Declaring function
    def getinfo(self):
        print(self.id, self.name, self.email)
s = Student()
s.getinfo()
delattr(Student,'course') # Removing attribute which is not available

s.getinfo() # error: throws an error
```

```
101 Pranshu pranshu@abc.com
ERROR!
Traceback (most recent call last):
  File "<main.py>", line 10, in <module>
AttributeError: type object 'Student' has no attribute 'course'

=== Code Exited With Errors ===
```

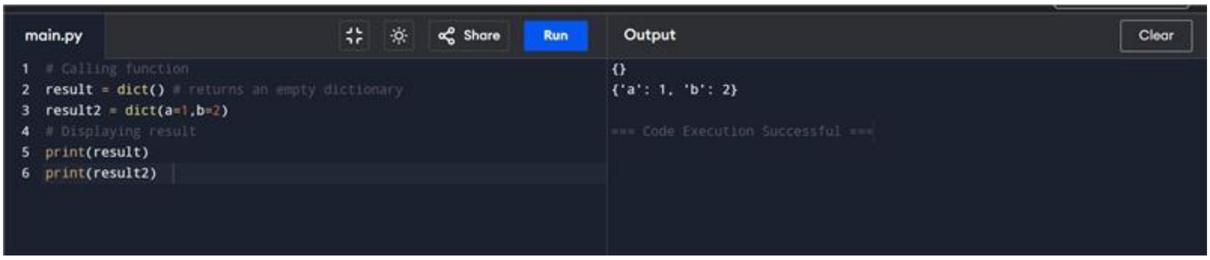
1.34) Python enum()



```python
# Calling function
result = enumerate([1,2,3])
# Displaying result
print(result)
print(list(result))
```

```
<enumerate object at 0x7d5ae41d36a0>
[(0, 1), (1, 2), (2, 3)]

=== Code Execution Successful ===
```

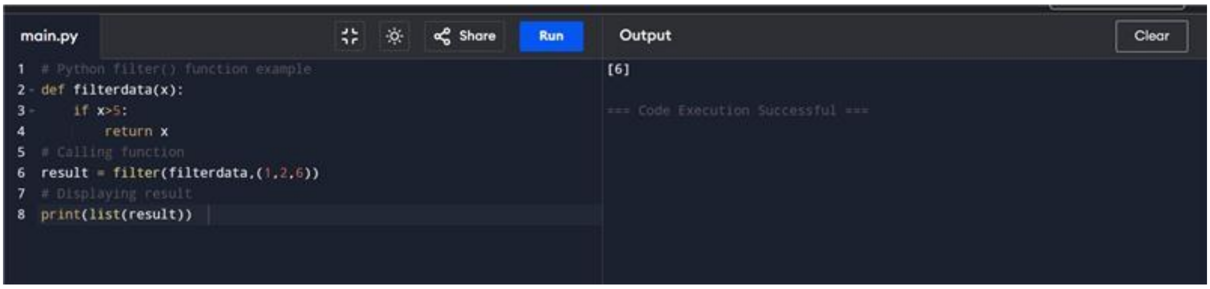
1.35) Python dict()



```python
# Calling function
result = dict() # returns an empty dictionary
result2 = dict(a=1,b=2)
# Displaying result
print(result)
print(result2)
```

```
{}
{'a': 1, 'b': 2}

=== Code Execution Successful ===
```

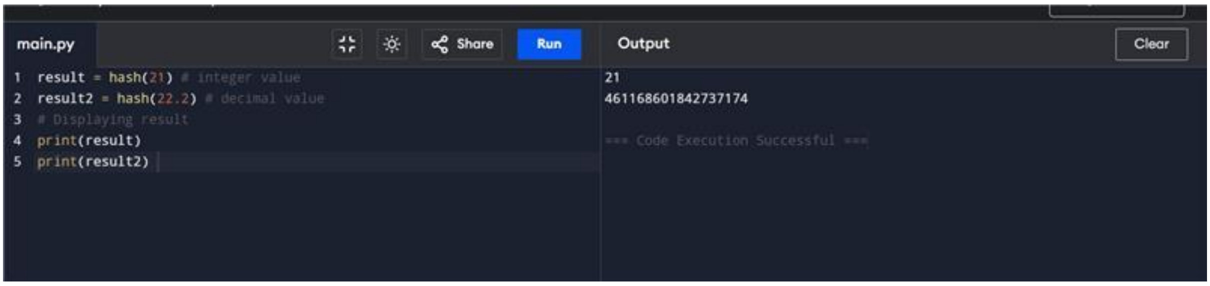
1.36) Python filter ()



```python
# Python filter() function example
def filterdata(x):
    if x>5:
        return x
# Calling function
result = filter(filterdata,(1,2,6))
# Displaying result
print(list(result))
```

```
[6]

=== Code Execution Successful ===
```

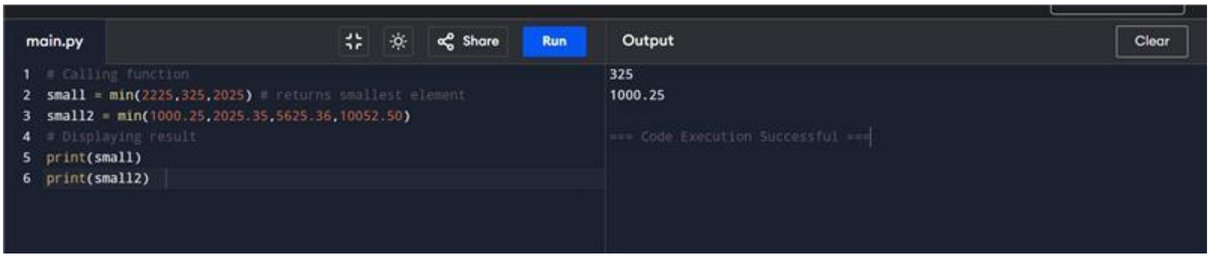
1.37) Python hash()



```python
result = hash(21) # integer value
result2 = hash(22.2) # decimal value
# Displaying result
print(result)
print(result2)
```

```
21
461168601842737174

=== Code Execution Successful ===
```

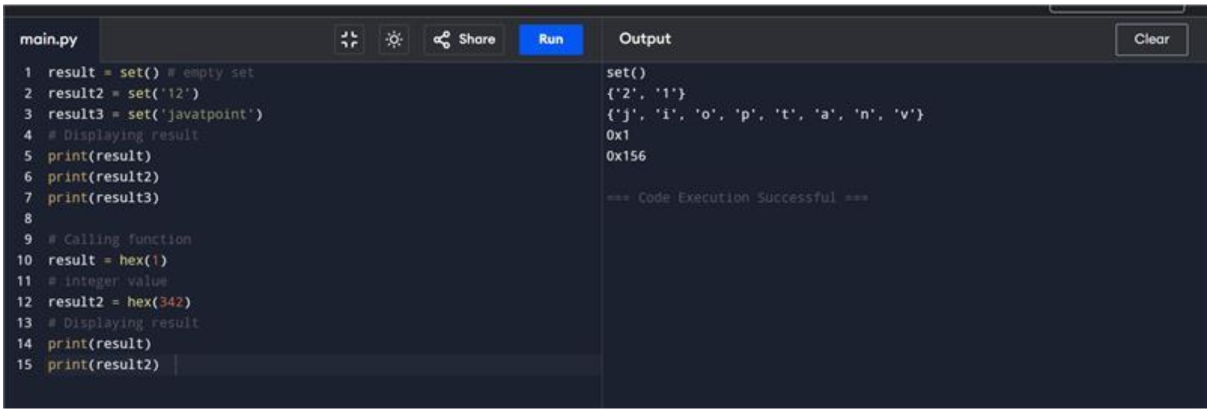
1.38) Python min()



```python
# Calling function
small = min(2225,325,2025) # returns smallest element
small2 = min(1000.25,2025.35,5625.36,10052.50)
# Displaying result
print(small)
print(small2)
```

```
325
1000.25

=== Code Execution Successful ===
```

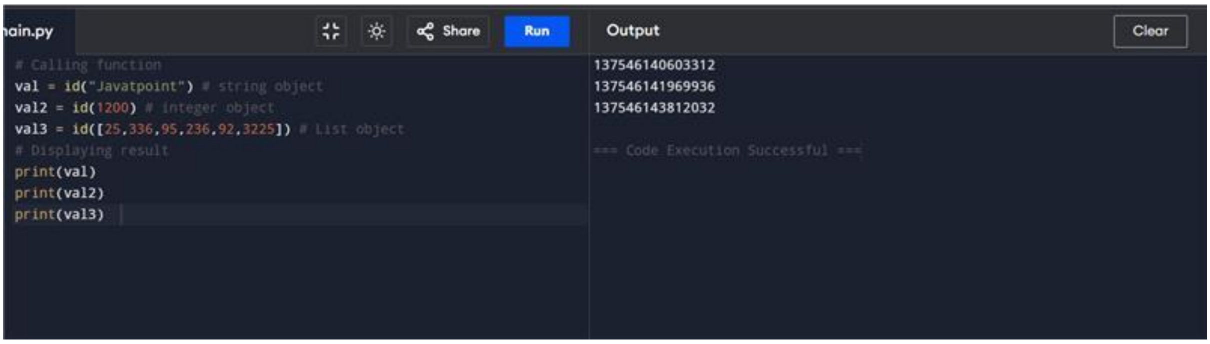
1.39) Python hex() and set() function



```python
result = set() # empty set
result2 = set('12')
result3 = set('javatpoint')
# Displaying result
print(result)
print(result2)
print(result3)

# Calling function
result = hex(1)
# integer value
result2 = hex(342)
# Displaying result
print(result)
print(result2)
```

```
set()
{'2', '1'}
{'j', 'i', 'o', 'p', 't', 'a', 'n', 'v'}
0x1
0x156

=== Code Execution Successful ===
```

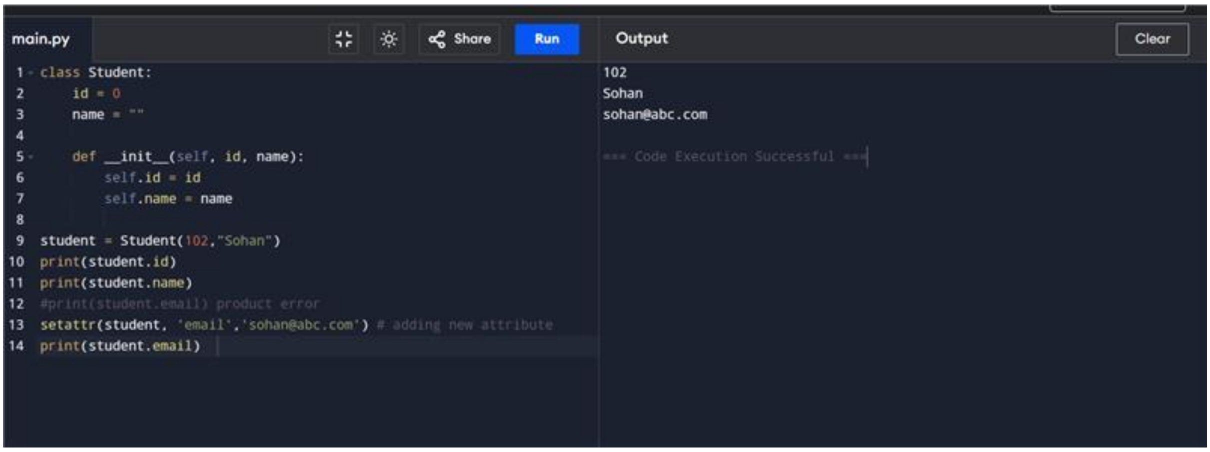
1.40) Python Id()



```python
# Calling function
val = id("Javatpoint") # string object
val2 = id(1200) # integer object
val3 = id([25,336,95,236,92,3225]) # List object
# Displaying result
print(val)
print(val2)
print(val3)
```

```
137546140603312
137546141969936
137546143812032

=== Code Execution Successful ===
```

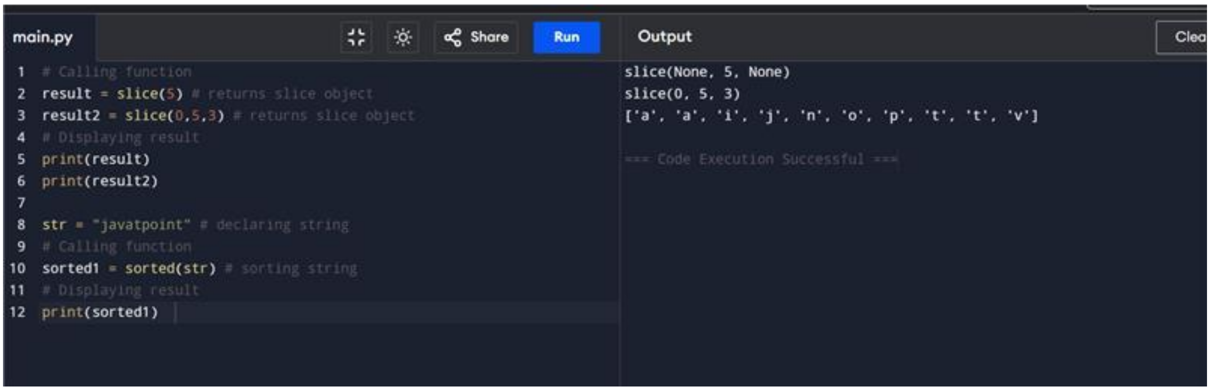
1.41) Python setattr()



```python
class Student:
    id = 0
    name = ""

    def __init__(self, id, name):
        self.id = id
        self.name = name

student = Student(102,"Sohan")
print(student.id)
print(student.name)
#print(student.email) product error
setattr(student, 'email','sohan@abc.com') # adding new attribute
print(student.email)
```

```
102
Sohan
sohan@abc.com

=== Code Execution Successful ===
```

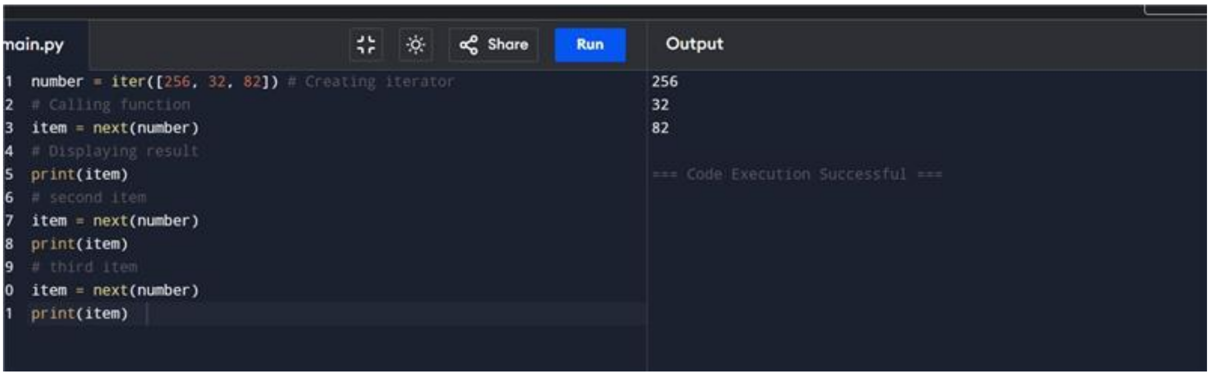
1.42) Python slice() and sorted()



```python
# Calling function
result = slice(5) # returns slice object
result2 = slice(0,5,3) # returns slice object
# Displaying result
print(result)
print(result2)

str = "javatpoint" # declaring string
# Calling function
sorted1 = sorted(str) # sorting string
# Displaying result
print(sorted1)
```

```
slice(None, 5, None)
slice(0, 5, 3)
['a', 'a', 'i', 'j', 'n', 'o', 'p', 't', 't', 'v']

=== Code Execution Successful ===
```

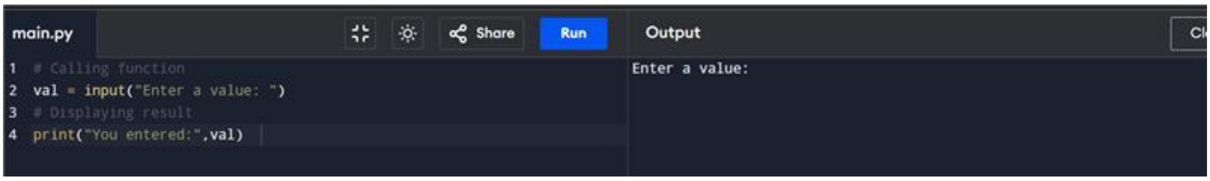
1.43) Python next()



```python
number = iter([256, 32, 82]) # Creating iterator
# Calling function
item = next(number)
# Displaying result
print(item)
# second item
item = next(number)
print(item)
# third item
item = next(number)
print(item)
```

```
256
32
82

=== Code Execution Successful ===
```

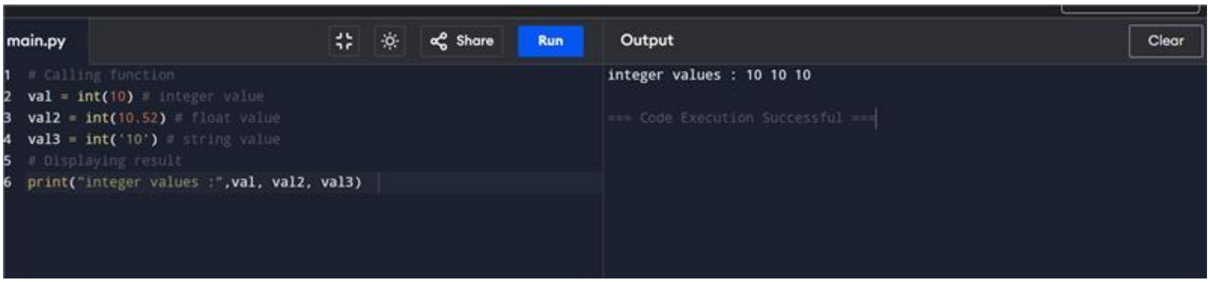
1.44) Python input()



```python
# Calling function
val = input("Enter a value: ")
# Displaying result
print("You entered:",val)
```

```
Enter a value:
```

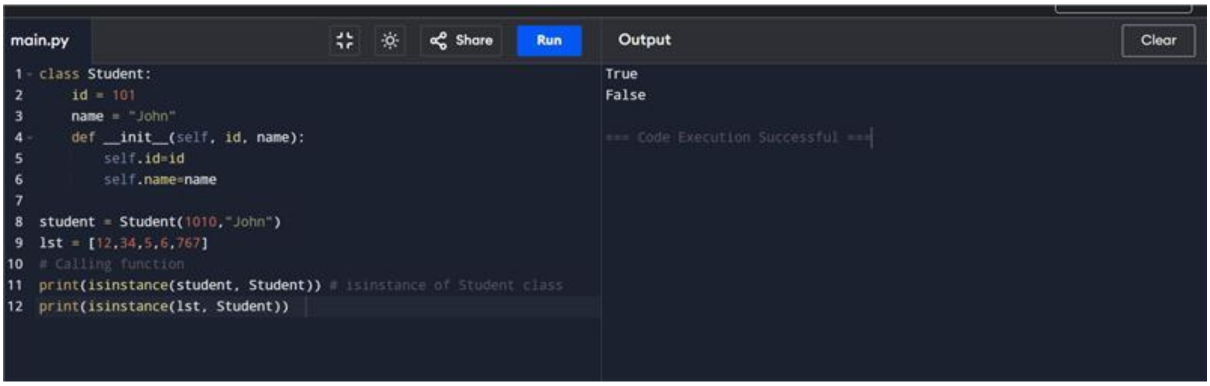
1.45) Python int()



```python
# Calling function
val = int(10) # integer value
val2 = int(10.52) # float value
val3 = int('10') # string value
# Displaying result
print("integer values :",val, val2, val3)
```

```
integer values : 10 10 10

=== Code Execution Successful ===
```

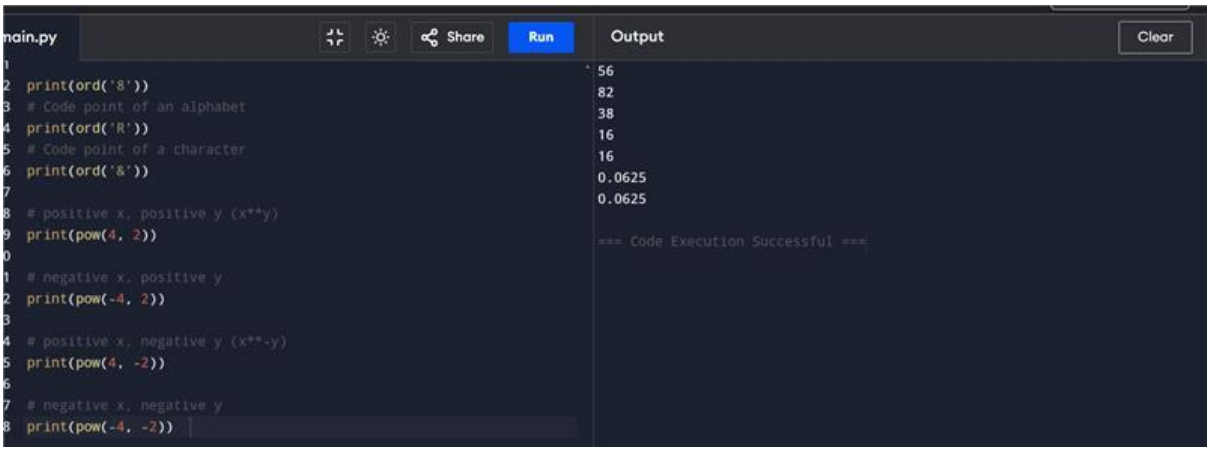
1.46) Python instance()



```python
class Student:
    id = 101
    name = "John"
    def __init__(self, id, name):
        self.id=id
        self.name=name

student = Student(1010,"John")
lst = [12,34,5,6,767]
# Calling function
print(isinstance(student, Student)) # isinstance of Student class
print(isinstance(lst, Student))
```

```
True
False

=== Code Execution Successful ===
```

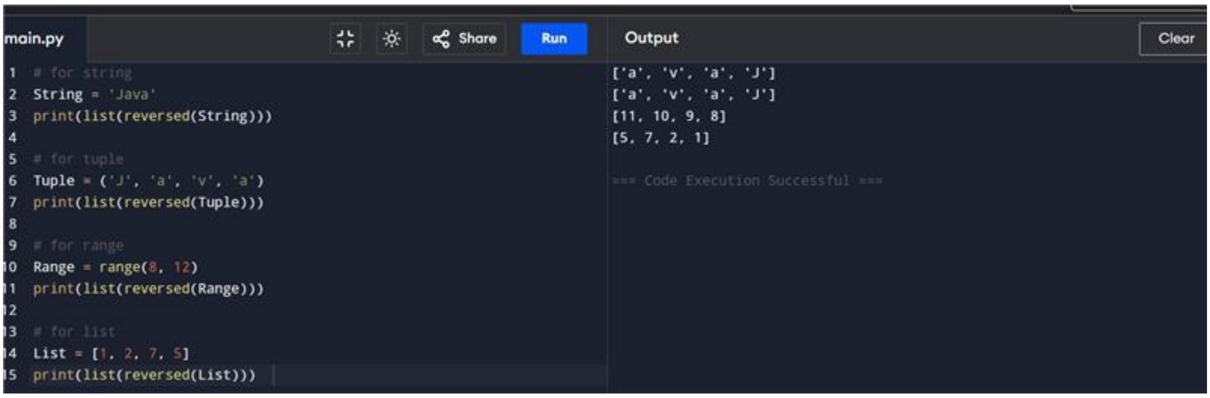
1.47) Python ord() and pow() function



```python
print(ord('8'))
# Code point of an alphabet
print(ord('R'))
# Code point of a character
print(ord('&'))

# positive x, positive y (x**y)
print(pow(4, 2))

# negative x, positive y
print(pow(-4, 2))

# positive x, negative y (x**-y)
print(pow(4, -2))

# negative x, negative y
print(pow(-4, -2))
```

```
56
82
38
16
16
0.0625
0.0625

=== Code Execution Successful ===
```

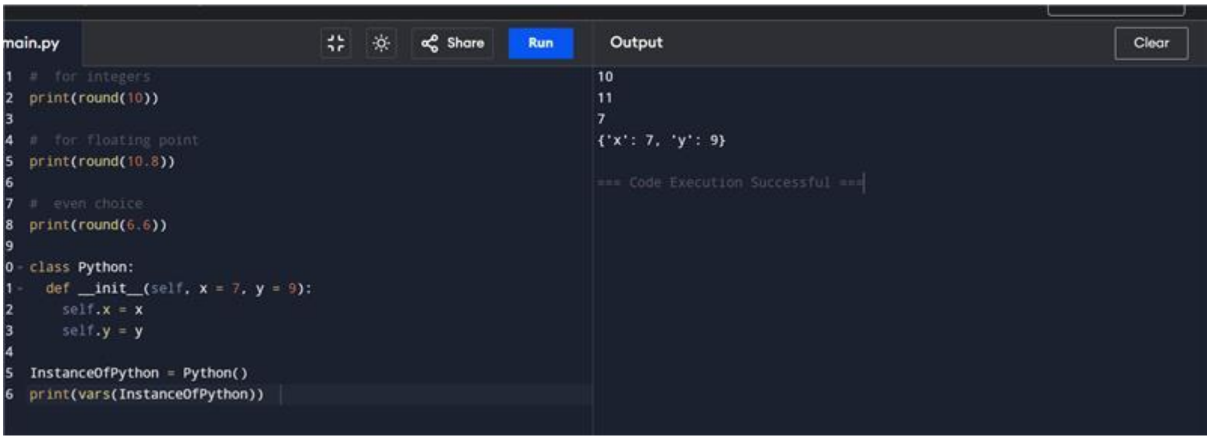
1.48) Python reversed()



```python
# for string
String = 'Java'
print(list(reversed(String)))

# for tuple
Tuple = ('J', 'a', 'v', 'a')
print(list(reversed(Tuple)))

# for range
Range = range(8, 12)
print(list(reversed(Range)))

# for list
List = [1, 2, 7, 5]
print(list(reversed(List)))
```

```
['a', 'v', 'a', 'J']
['a', 'v', 'a', 'J']
[11, 10, 9, 8]
[5, 7, 2, 1]

=== Code Execution Successful ===
```

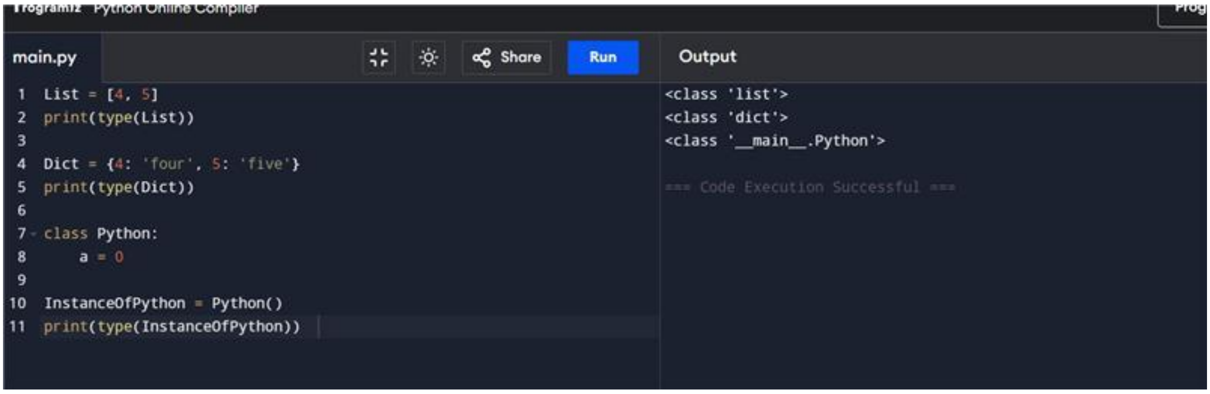
1.49) Python round() and var()



```python
#  for integers
print(round(10))

#  for floating point
print(round(10.8))

#  even choice
print(round(6.6))

class Python:
    def __init__(self, x = 7, y = 9):
        self.x = x
        self.y = y

InstanceOfPython = Python()
print(vars(InstanceOfPython))
```

```
10
11
7
{'x': 7, 'y': 9}

=== Code Execution Successful ===
```

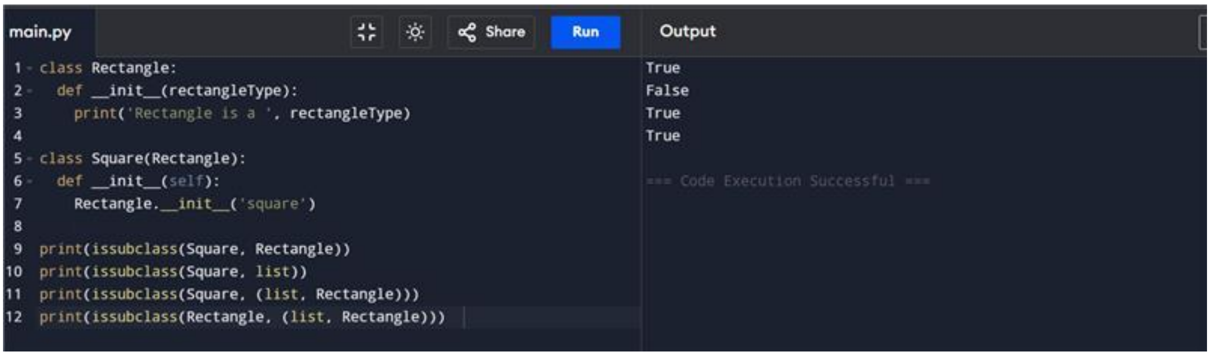
1.50) Python type() fucntion



```python
List = [4, 5]
print(type(List))

Dict = {4: 'four', 5: 'five'}
print(type(Dict))

class Python:
    a = 0

InstanceOfPython = Python()
print(type(InstanceOfPython))
```

```
<class 'list'>
<class 'dict'>
<class '__main__.Python'>

=== Code Execution Successful ===
```

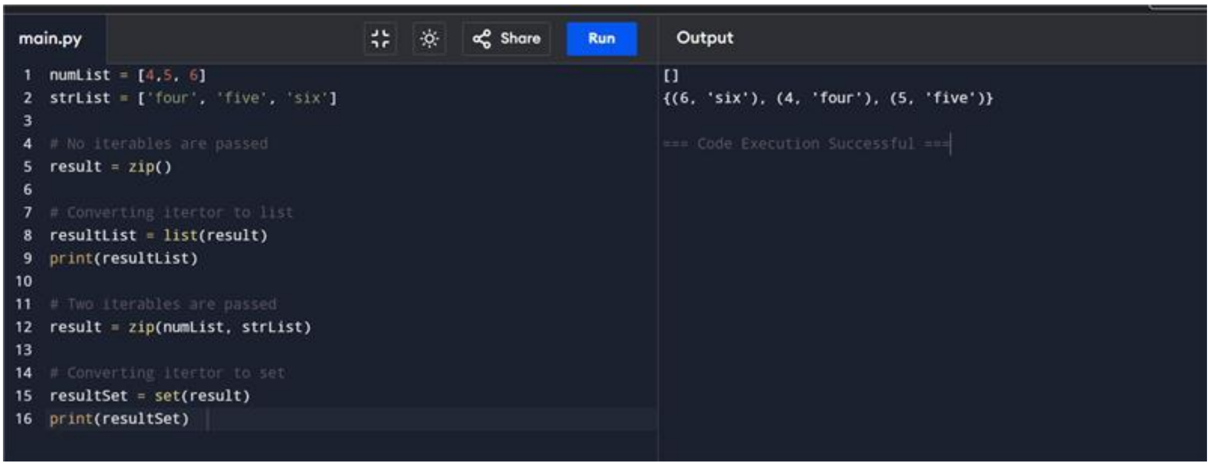
1.51) Python issubclass()



```python
class Rectangle:
  def __init__(rectangleType):
    print('Rectangle is a ', rectangleType)

class Square(Rectangle):
  def __init__(self):
    Rectangle.__init__('square')

print(issubclass(Square, Rectangle))
print(issubclass(Square, list))
print(issubclass(Square, (list, Rectangle)))
print(issubclass(Rectangle, (list, Rectangle)))
```

```
True
False
True
True

=== Code Execution Successful ===
```

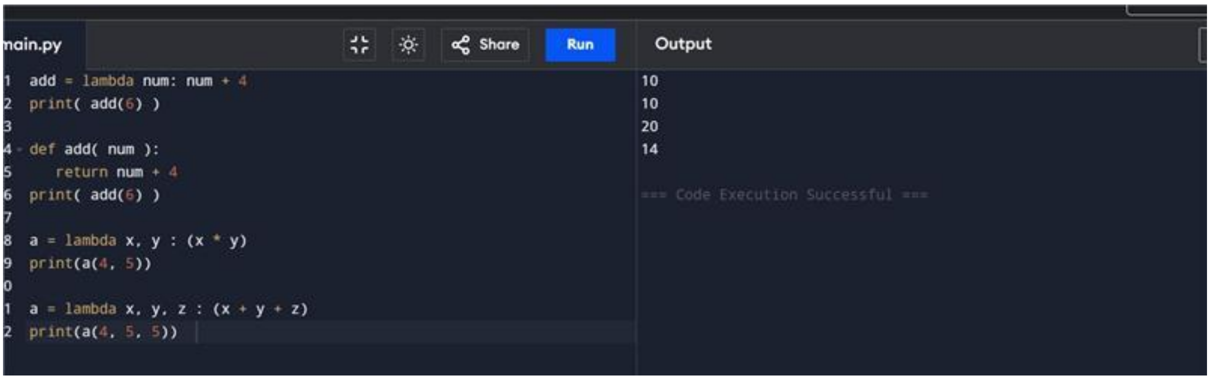
1.52) Python zip()



```python
numList = [4,5, 6]
strList = ['four', 'five', 'six']

# No iterables are passed
result = zip()

# Converting itertor to list
resultList = list(result)
print(resultList)

# Two iterables are passed
result = zip(numList, strList)

# Converting itertor to set
resultSet = set(result)
print(resultSet)
```

```
[]
{(6, 'six'), (4, 'four'), (5, 'five')}

=== Code Execution Successful ===
```

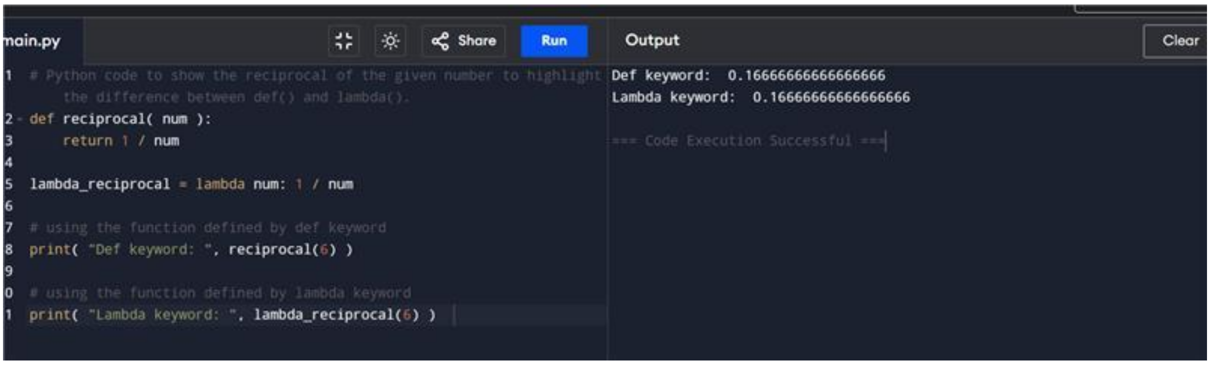
1.53) Python lambda()



```python
add = lambda num: num + 4
print( add(6) )

def add( num ):
    return num + 4
print( add(6) )

a = lambda x, y : (x * y)
print(a(4, 5))

a = lambda x, y, z : (x + y + z)
print(a(4, 5, 5))
```

```
10
10
20
14

=== Code Execution Successful ===
```

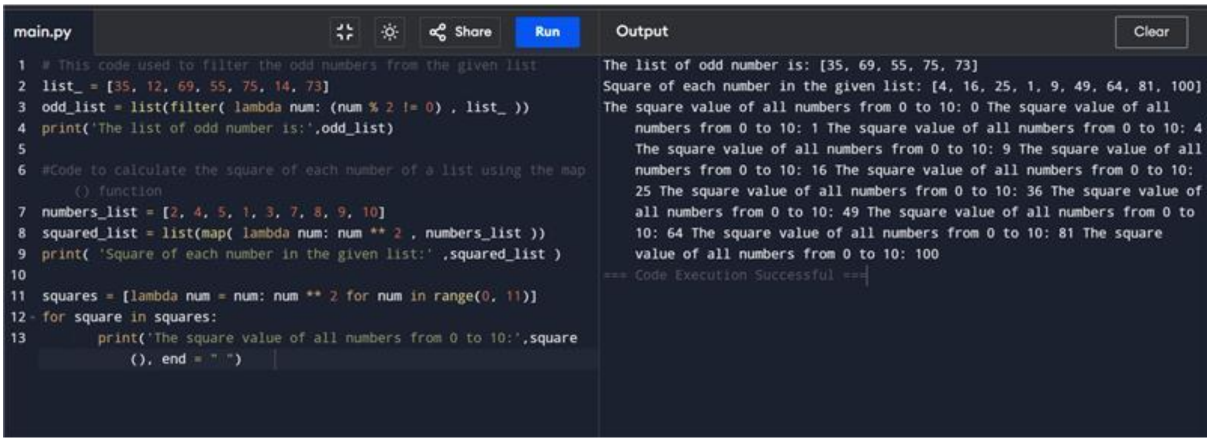
1.54) Def vs lambda difference



```python
# Python code to show the reciprocal of the given number to highlight the difference between def() and lambda().
def reciprocal( num ):
    return 1 / num

lambda_reciprocal = lambda num: 1 / num

# using the function defined by def keyword
print( "Def keyword: ", reciprocal(6) )

# using the function defined by lambda keyword
print( "Lambda keyword: ", lambda_reciprocal(6) )
```

```
Def keyword:  0.16666666666666666
Lambda keyword:  0.16666666666666666

=== Code Execution Successful ===
```

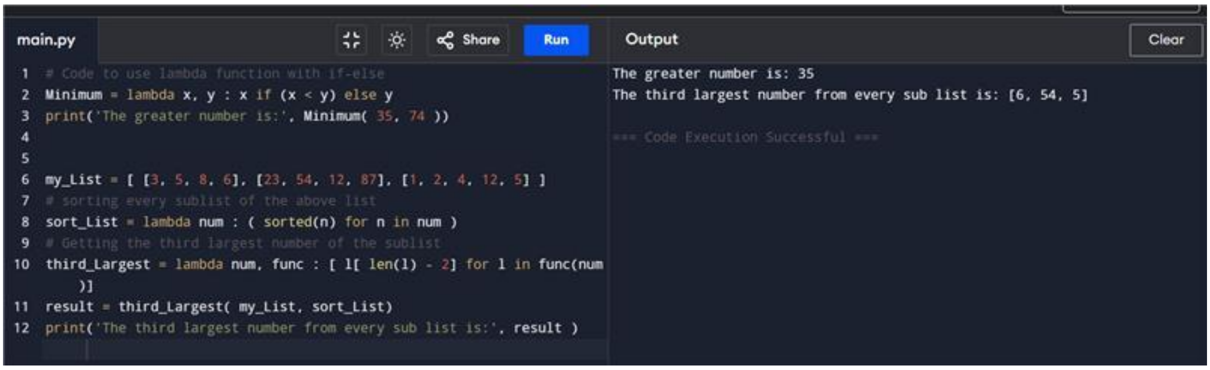
1.55) Using Lambda Function with filter(),map(),list comprehension()



```python
# This code used to filter the odd numbers from the given list
list_ = [35, 12, 69, 55, 75, 14, 73]
odd_list = list(filter( lambda num: (num % 2 != 0) , list_ ))
print('The list of odd number is:',odd_list)

#Code to calculate the square of each number of a list using the map() function
numbers_list = [2, 4, 5, 1, 3, 7, 8, 9, 10]
squared_list = list(map( lambda num: num ** 2 , numbers_list ))
print( 'Square of each number in the given list:' ,squared_list )

squares = [lambda num = num: num ** 2 for num in range(0, 11)]
for square in squares:
        print('The square value of all numbers from 0 to 10:',square(), end = " ")
```

```
The list of odd number is: [35, 69, 55, 75, 73]
Square of each number in the given list: [4, 16, 25, 1, 9, 49, 64, 81, 100]
The square value of all numbers from 0 to 10: 0 The square value of all
    numbers from 0 to 10: 1 The square value of all numbers from 0 to 10: 4
    The square value of all numbers from 0 to 10: 9 The square value of all
    numbers from 0 to 10: 16 The square value of all numbers from 0 to 10:
    25 The square value of all numbers from 0 to 10: 36 The square value of
    all numbers from 0 to 10: 49 The square value of all numbers from 0 to
    10: 64 The square value of all numbers from 0 to 10: 81 The square
    value of all numbers from 0 to 10: 100
=== Code Execution Successful ===
```

1.56) Lambda function with multiple statement and if else

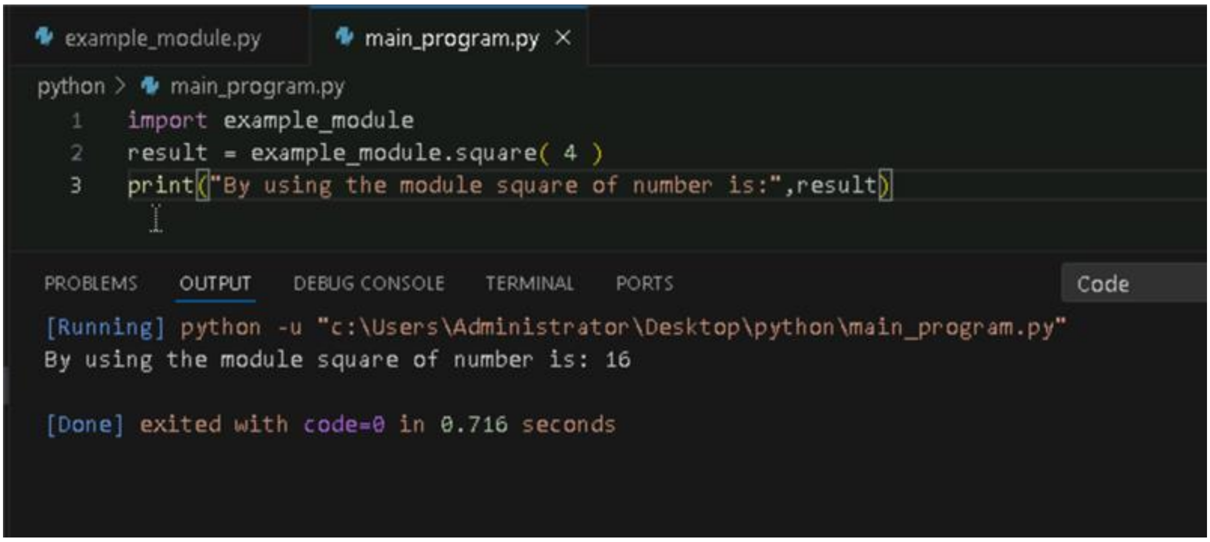


```python
# Code to use lambda function with if-else
Minimum = lambda x, y : x if (x < y) else y
print('The greater number is:', Minimum( 35, 74 ))


my_List = [ [3, 5, 8, 6], [23, 54, 12, 87], [1, 2, 4, 12, 5] ]
# sorting every sublist of the above list
sort_List = lambda num : ( sorted(n) for n in num )
# Getting the third largest number of the sublist
third_Largest = lambda num, func : [ l[ len(l) - 2] for l in func(num
    )]
result = third_Largest( my_List, sort_List)
print('The third largest number from every sub list is:', result )
```

```
The greater number is: 35
The third largest number from every sub list is: [6, 54, 5]

=== Code Execution Successful ===
```

2) MODULES

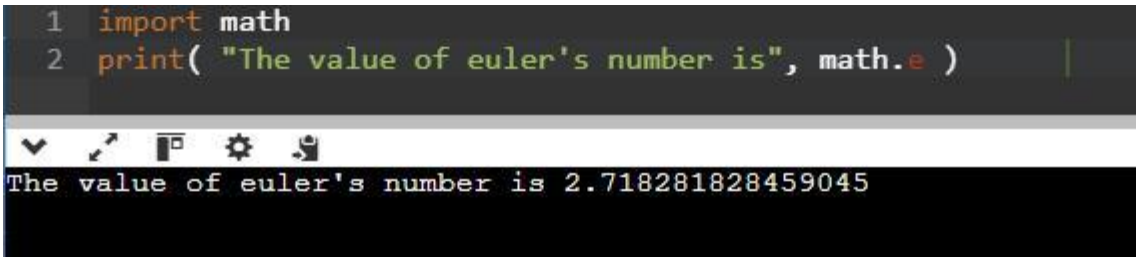
2.1.Importing modules



```python
import example_module
result = example_module.square( 4 )
print("By using the module square of number is:",result)
```

```
[Running] python -u "c:\Users\Administrator\Desktop\python\main_program.py"
By using the module square of number is: 16

[Done] exited with code=0 in 0.716 seconds
```

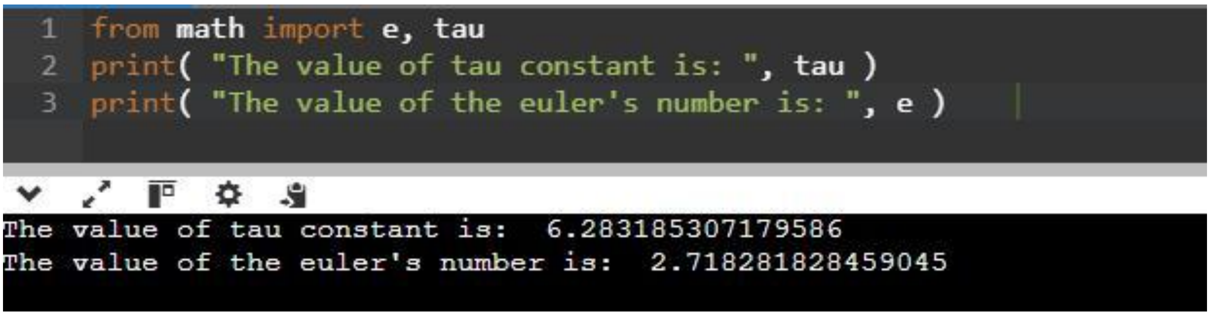
2.2. Importing and also Renaming:



```python
import math
print( "The value of euler's number is", math.e )
```

```
The value of euler's number is 2.718281828459045
```

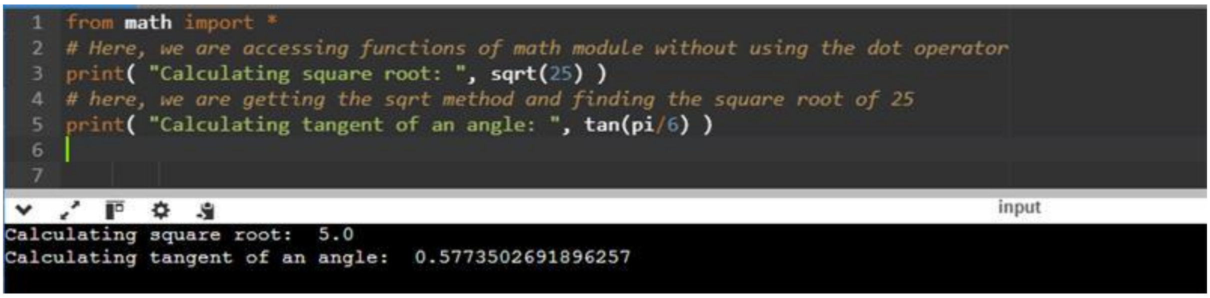
2.3. Python from...import Statement:



```python
from math import e, tau
print( "The value of tau constant is: ", tau )
print( "The value of the euler's number is: ", e )
```

```
The value of tau constant is:  6.283185307179586
The value of the euler's number is:  2.718281828459045
```

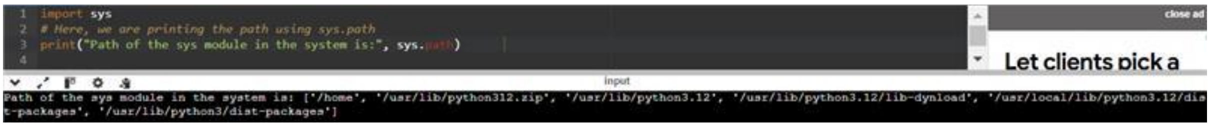
2.4. Import all Names - From import * Statement:



```python
from math import *
# Here, we are accessing functions of math module without using the dot operator
print( "Calculating square root: ", sqrt(25) )
# here, we are getting the sqrt method and finding the square root of 25
print( "Calculating tangent of an angle: ", tan(pi/6) )
```

```
Calculating square root:  5.0
Calculating tangent of an angle:  0.5773502691896257
```

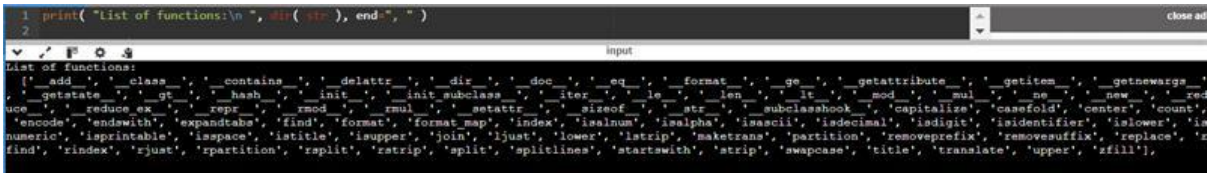
2.5. Locating Path of Modules:



```python
import sys
# Here, we are printing the path using sys.path
print("Path of the sys module in the system is:", sys.path)
```

```
Path of the sys module in the system is: ['/home', '/usr/lib/python312.zip', '/usr/lib/python3.12', '/usr/lib/python3.12/lib-dynload', '/usr/local/lib/python3.12/dis
t-packages', '/usr/lib/python3/dist-packages']
```

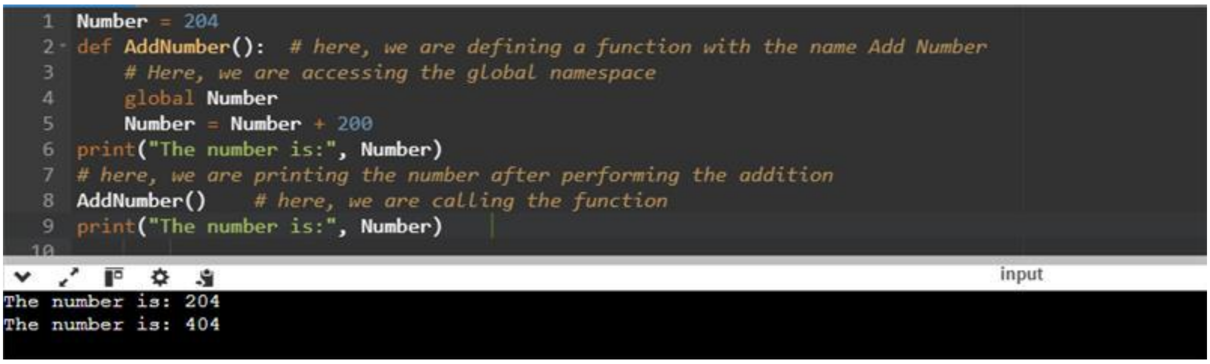
2.6. The dir() Built-in Function:



```python
print( "List of functions:\n ", dir( str ), end=", " )
```

```
List of functions:
  ['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__'
, '__getstate__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__red
uce__', '__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', 'capitalize', 'casefold', 'center', 'count',
 'encode', 'endswith', 'expandtabs', 'find', 'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isascii', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'is
numeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'removeprefix', 'removesuffix', 'replace', 'r
find', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill'],
```

2.7. Namespaces and Scoping:

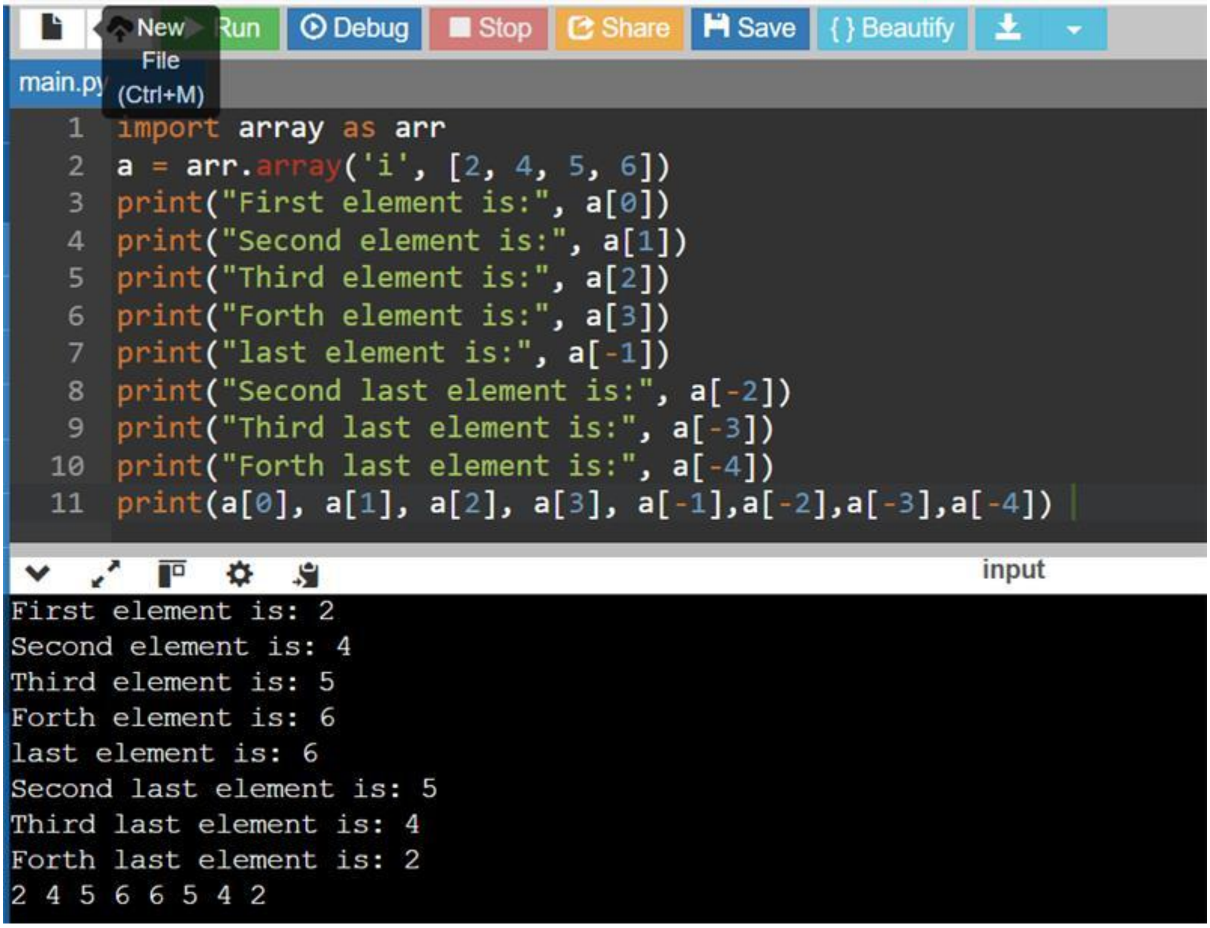


```python
Number = 204
def AddNumber():  # here, we are defining a function with the name Add Number
    # Here, we are accessing the global namespace
    global Number
    Number = Number + 200
print("The number is:", Number)
# here, we are printing the number after performing the addition
AddNumber()    # here, we are calling the function
print("The number is:", Number)
```

```
The number is: 204
The number is: 404
```

3) PYTHON ARRAYS

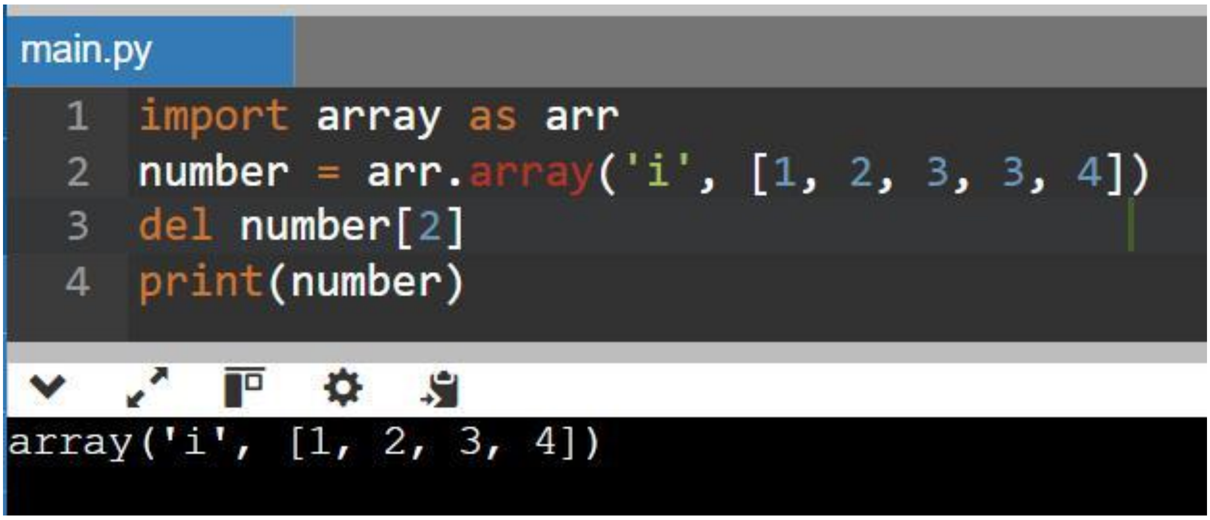
3.1. Accessing array elements:



```
import array as arr
a = arr.array('i', [2, 4, 5, 6])
print("First element is:", a[0])
print("Second element is:", a[1])
print("Third element is:", a[2])
print("Forth element is:", a[3])
print("last element is:", a[-1])
print("Second last element is:", a[-2])
print("Third last element is:", a[-3])
print("Forth last element is:", a[-4])
print(a[0], a[1], a[2], a[3], a[-1],a[-2],a[-3],a[-4])
```

```
First element is: 2
Second element is: 4
Third element is: 5
Forth element is: 6
last element is: 6
Second last element is: 5
Third last element is: 4
Forth last element is: 2
2 4 5 6 6 5 4 2
```

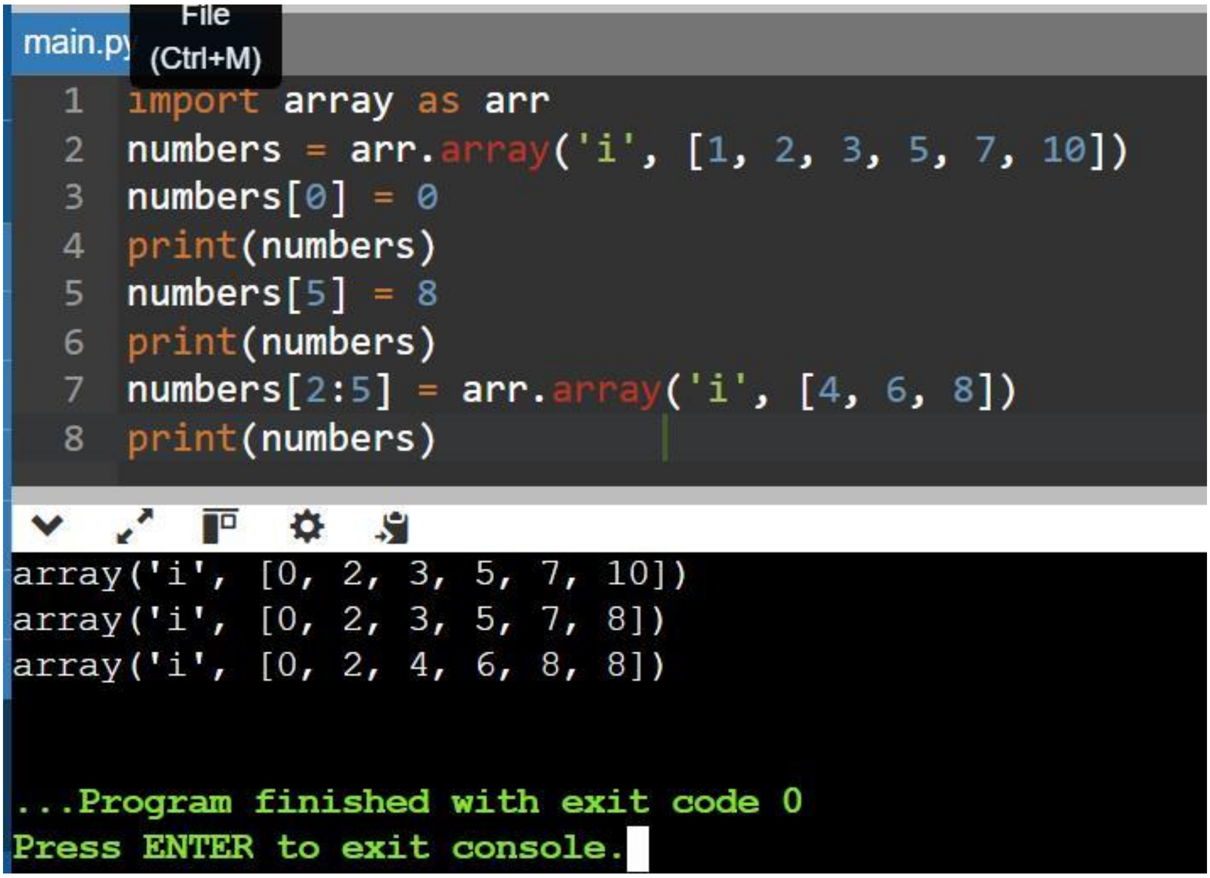
3.2. Deleting the elements from Array



```
import array as arr
number = arr.array('i', [1, 2, 3, 3, 4])
del number[2]
print(number)
```

```
array('i', [1, 2, 3, 4])
```

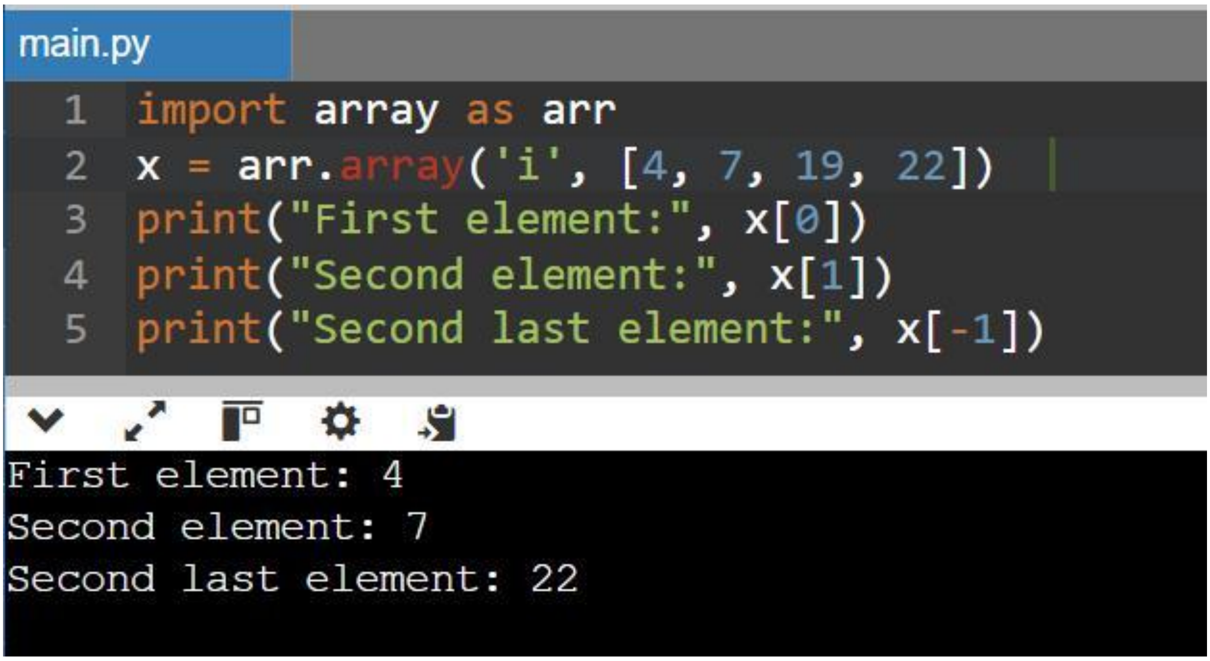
3.3. Adding or changing the elements in Array



```python
import array as arr
numbers = arr.array('i', [1, 2, 3, 5, 7, 10])
numbers[0] = 0
print(numbers)
numbers[5] = 8
print(numbers)
numbers[2:5] = arr.array('i', [4, 6, 8])
print(numbers)
```

```
array('i', [0, 2, 3, 5, 7, 10])
array('i', [0, 2, 3, 5, 7, 8])
array('i', [0, 2, 4, 6, 8, 8])


...Program finished with exit code 0
Press ENTER to exit console.
```

3.4. To find the length of array

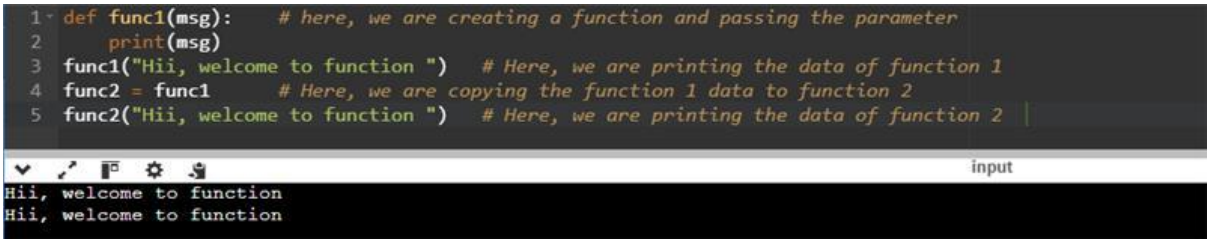


```python
import array as arr
x = arr.array('i', [4, 7, 19, 22])
print("First element:", x[0])
print("Second element:", x[1])
print("Second last element:", x[-1])
```

```
First element: 4
Second element: 7
Second last element: 22
```

4) PYTHON DECORATOR

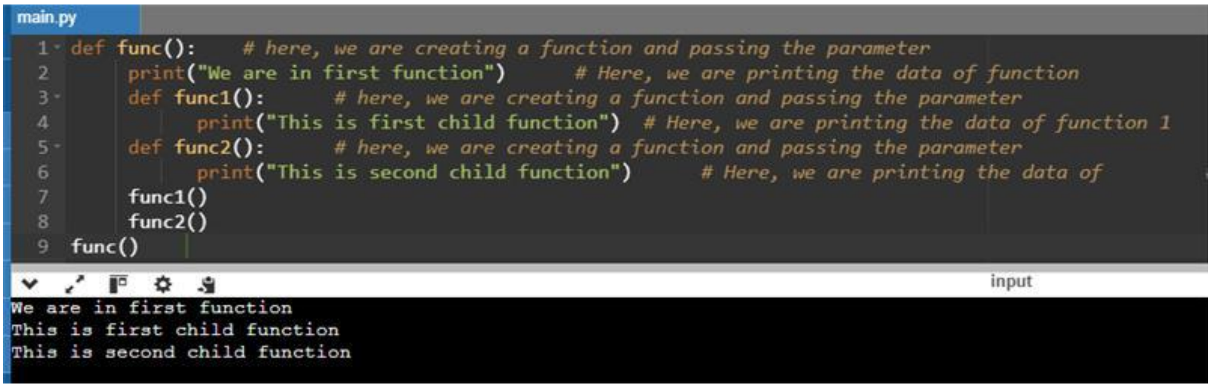
4.1



```
def func1(msg):     # here, we are creating a function and passing the parameter
    print(msg)
func1("Hii, welcome to function ")   # Here, we are printing the data of function 1
func2 = func1       # Here, we are copying the function 1 data to function 2
func2("Hii, welcome to function ")   # Here, we are printing the data of function 2
```

```
Hii, welcome to function
Hii, welcome to function
```

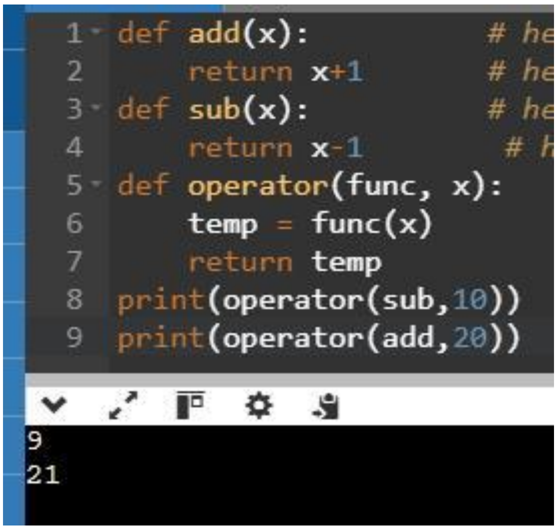
2. Inner Function



```
def func():     # here, we are creating a function and passing the parameter
    print("We are in first function")       # Here, we are printing the data of function
    def func1():      # here, we are creating a function and passing the parameter
        print("This is first child function")  # Here, we are printing the data of function 1
    def func2():      # here, we are creating a function and passing the parameter
        print("This is second child function")      # Here, we are printing the data of
    func1()
    func2()
func()
```

```
We are in first function
This is first child function
This is second child function
```

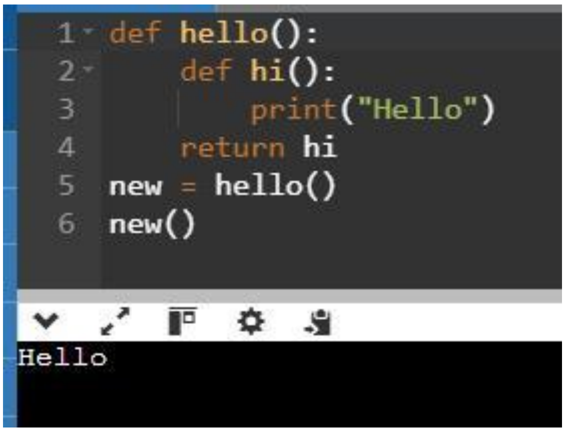
4.3.



```
def add(x):             # he
    return x+1          # he
def sub(x):             # he
    return x-1           # h
def operator(func, x):
    temp = func(x)
    return temp
print(operator(sub,10))
print(operator(add,20))
```

```
9
21
```

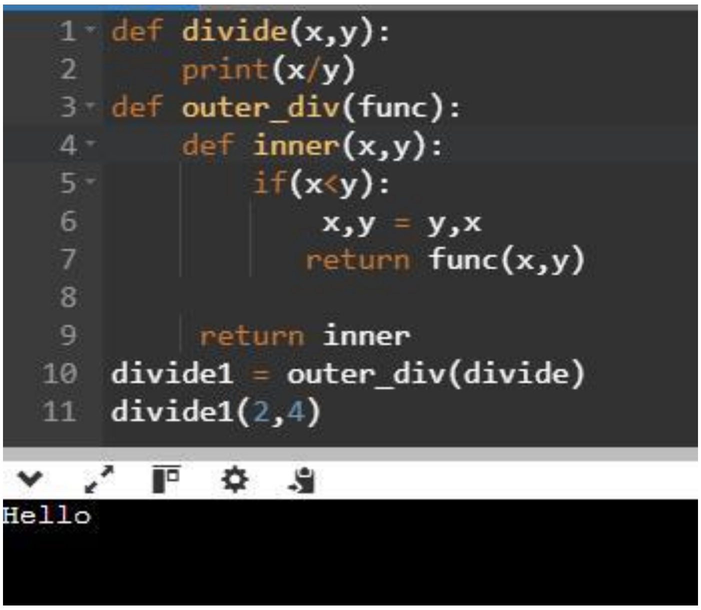
4.4.



```
def hello():
    def hi():
        print("Hello")
    return hi
new = hello()
new()
```

```
Hello
```

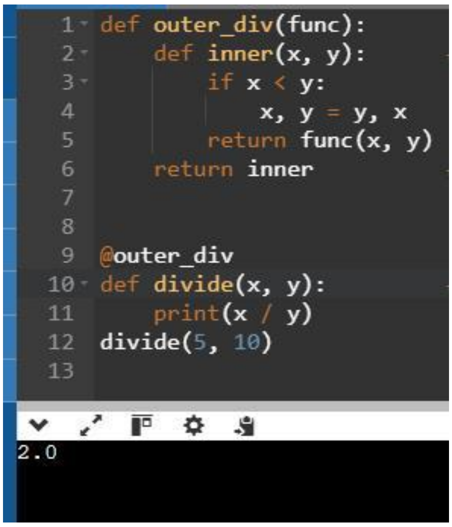
4.5.Decorating functions with parameters:



```
def divide(x,y):
    print(x/y)
def outer_div(func):
    def inner(x,y):
        if(x<y):
            x,y = y,x
           return func(x,y)

     return inner
divide1 = outer_div(divide)
divide1(2,4)
```

```
Hello
```

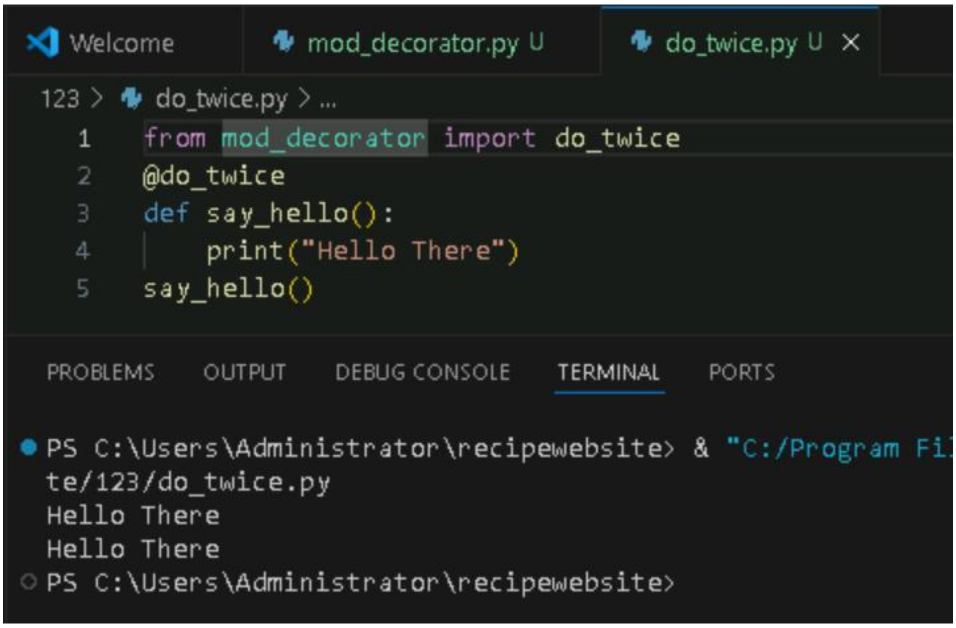
4.6.Syntactic Decorator:



```
def outer_div(func):
    def inner(x, y):
        if x < y:
            x, y = y, x
        return func(x, y)
    return inner


@outer_div
def divide(x, y):
    print(x / y)
divide(5, 10)

```

```
2.0
```

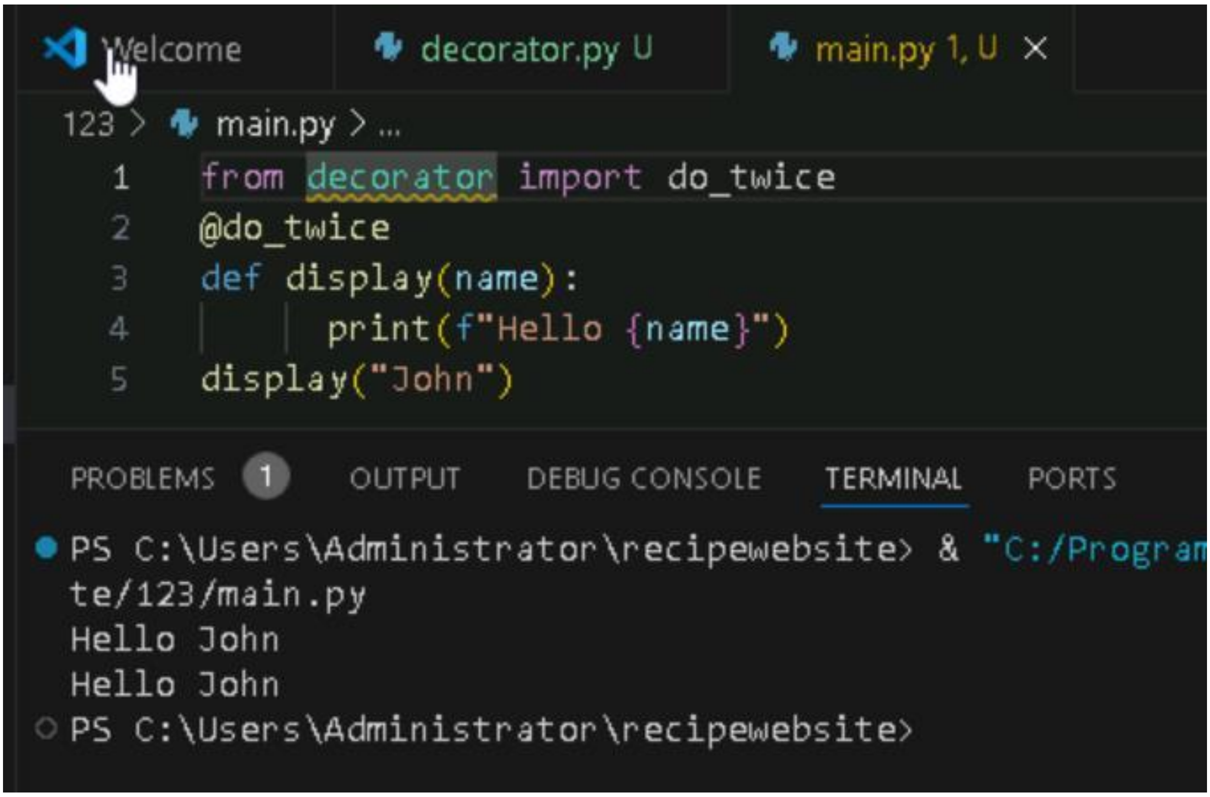
4.7.Reusing Decorator



```python
from mod_decorator import do_twice
@do_twice
def say_hello():
    print("Hello There")
say_hello()
```

```
PS C:\Users\Administrator\recipewebsite> & "C:/Program Fi
te/123/do_twice.py
Hello There
Hello There
PS C:\Users\Administrator\recipewebsite>
```

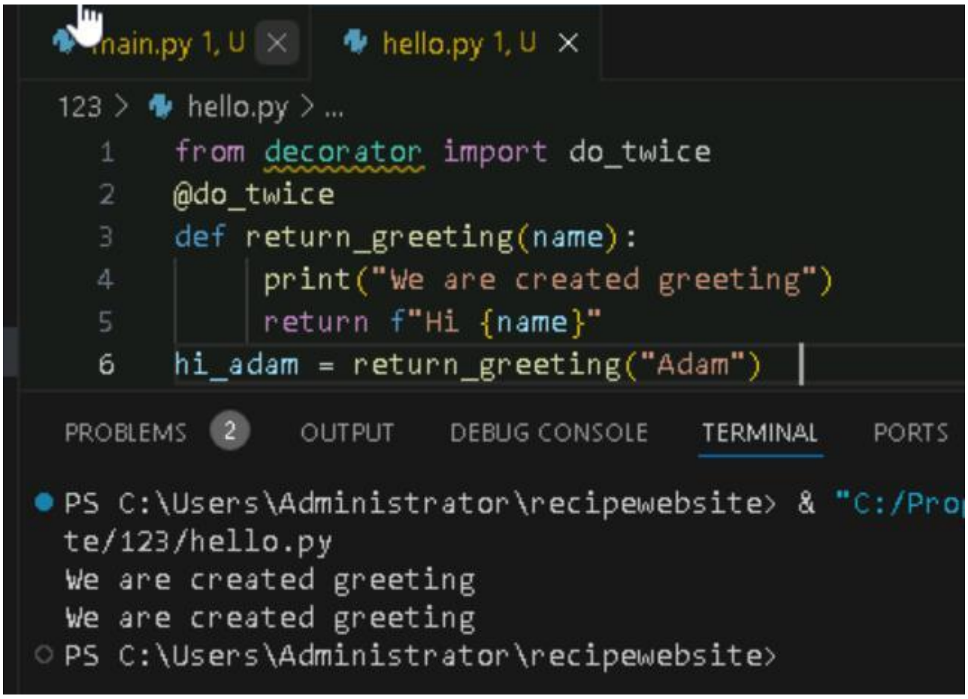
4.8.Python Decorator with Argument



```python
from decorator import do_twice
@do_twice
def display(name):
        print(f"Hello {name}")
display("John")
```

```
PS C:\Users\Administrator\recipewebsite> & "C:/Program
te/123/main.py
Hello John
Hello John
PS C:\Users\Administrator\recipewebsite>
```

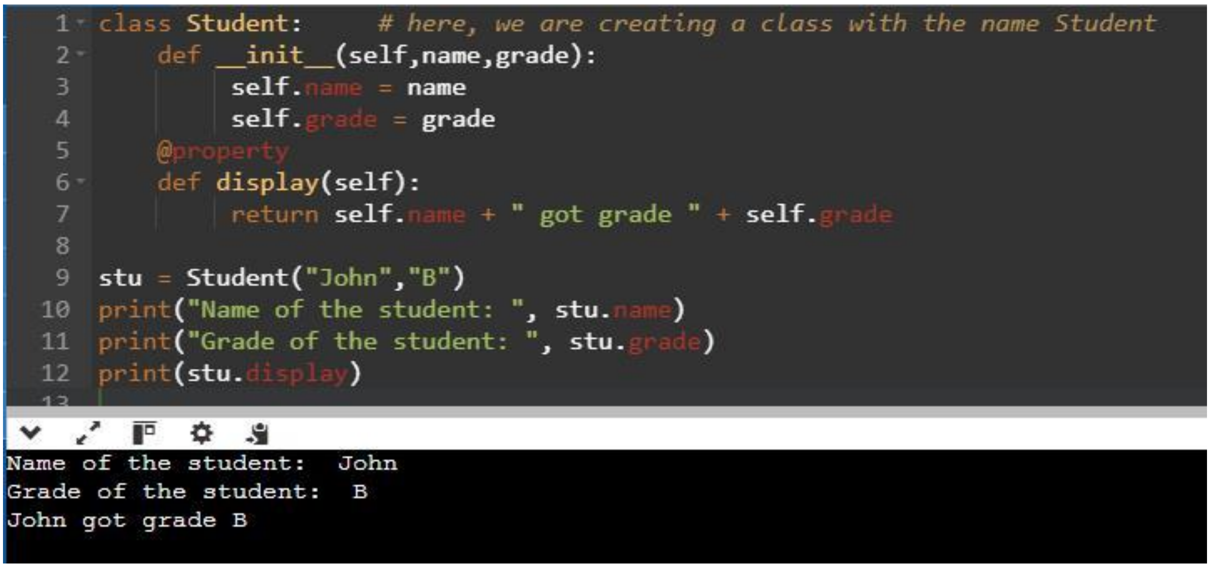
4.9.Returning Values from Decorated Functions



```python
from decorator import do_twice
@do_twice
def return_greeting(name):
    print("We are created greeting")
    return f"Hi {name}"
hi_adam = return_greeting("Adam")
```

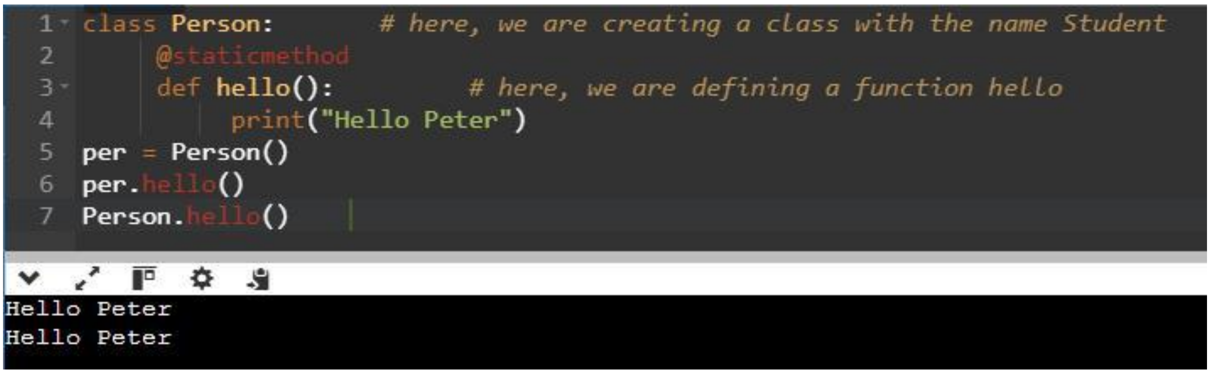
```
PS C:\Users\Administrator\recipewebsite> & "C:/Pro
te/123/hello.py
We are created greeting
We are created greeting
PS C:\Users\Administrator\recipewebsite>
```

4.10.Fancy Decorators



```python
class Student:     # here, we are creating a class with the name Student
    def __init__(self,name,grade):
        self.name = name
        self.grade = grade
    @property
    def display(self):
        return self.name + " got grade " + self.grade

stu = Student("John","B")
print("Name of the student: ", stu.name)
print("Grade of the student: ", stu.grade)
print(stu.display)
```

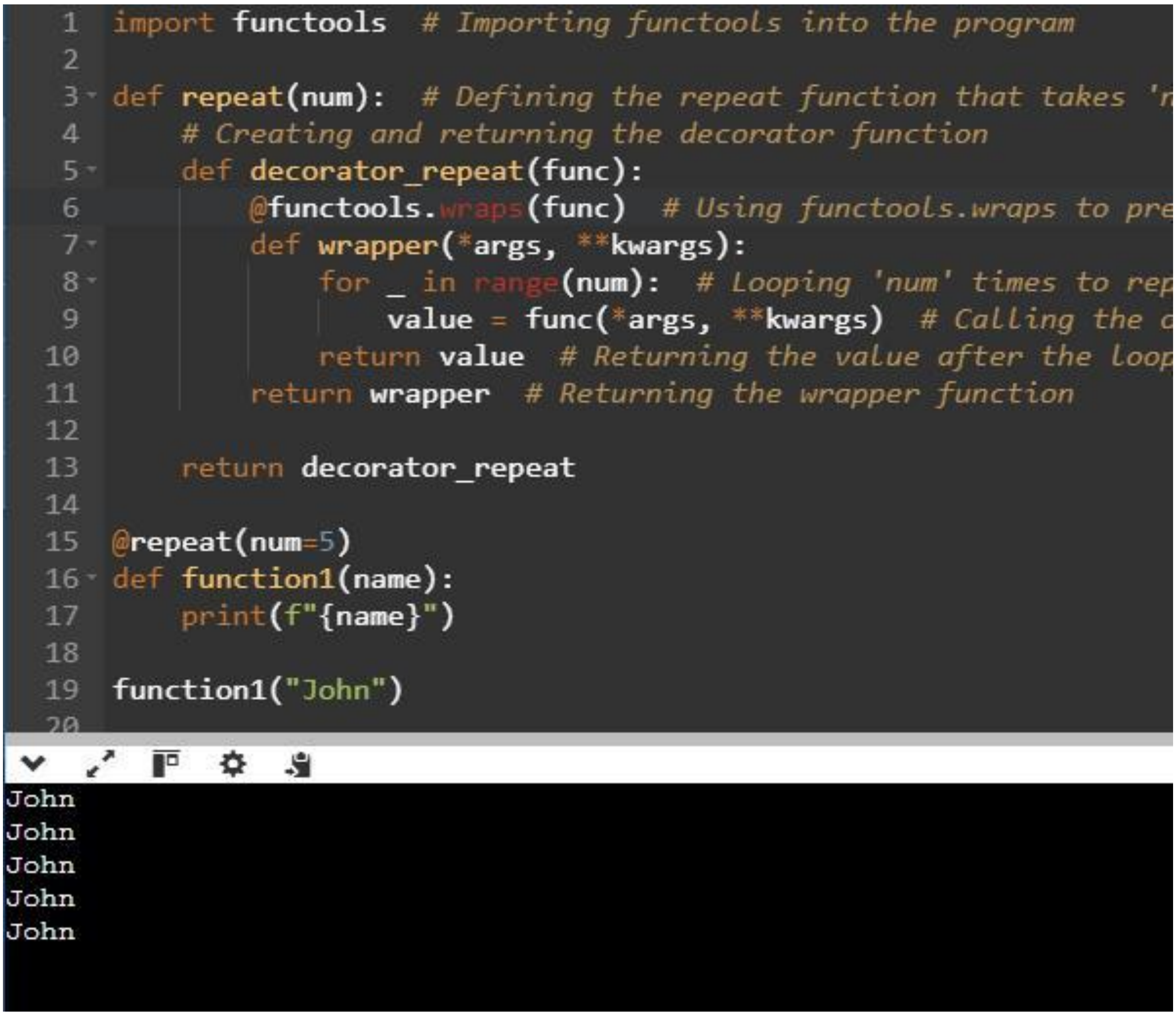
```
Name of the student:  John
Grade of the student:  B
John got grade B
```



```python
class Person:        # here, we are creating a class with the name Student
    @staticmethod
    def hello():          # here, we are defining a function hello
        print("Hello Peter")
per = Person()
per.hello()
Person.hello()
```

```
Hello Peter
Hello Peter
```

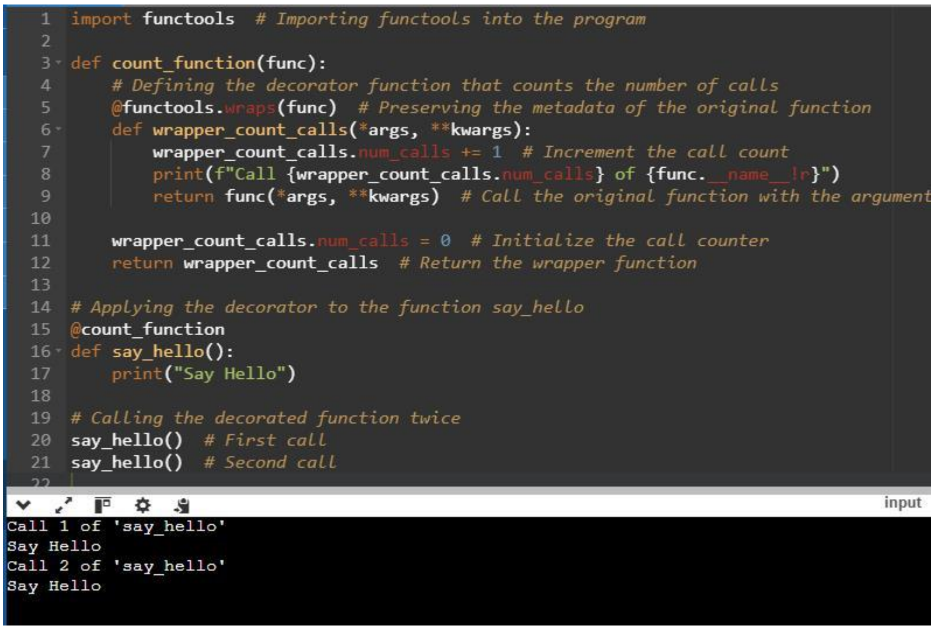
4.11.Decorator with Arguments



```
import functools  # Importing functools into the program

def repeat(num):  # Defining the repeat function that takes 'n
    # Creating and returning the decorator function
    def decorator_repeat(func):
        @functools.wraps(func)  # Using functools.wraps to pre
        def wrapper(*args, **kwargs):
            for _ in range(num):  # Looping 'num' times to rep
                value = func(*args, **kwargs)  # Calling the c
            return value  # Returning the value after the loop
        return wrapper  # Returning the wrapper function

    return decorator_repeat

@repeat(num=5)
def function1(name):
    print(f"{name}")

function1("John")
```

```
John
John
John
John
John
```

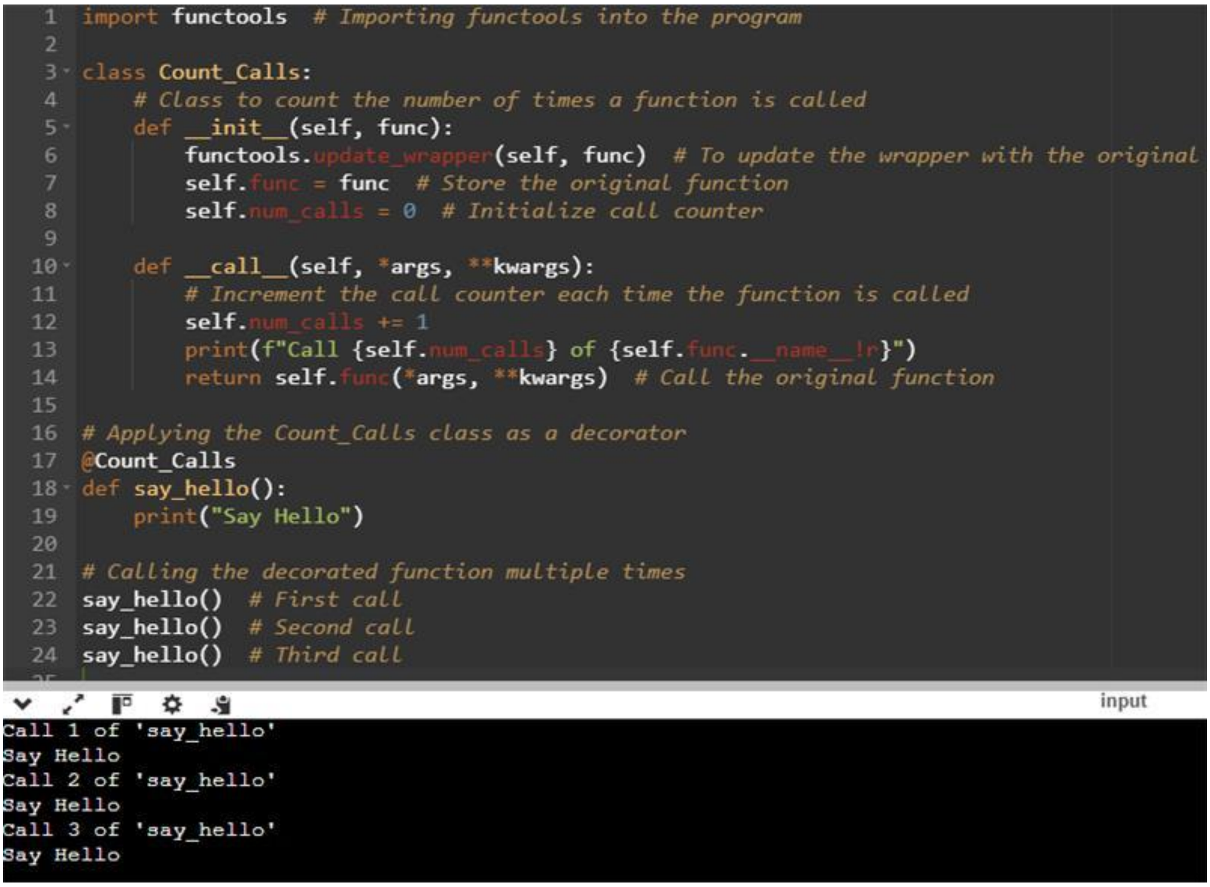
4.12.Stateful Decorators



```
import functools  # Importing functools into the program

def count_function(func):
    # Defining the decorator function that counts the number of calls
    @functools.wraps(func)  # Preserving the metadata of the original function
    def wrapper_count_calls(*args, **kwargs):
        wrapper_count_calls.num_calls += 1  # Increment the call count
        print(f"Call {wrapper_count_calls.num_calls} of {func.__name__!r}")
        return func(*args, **kwargs)  # Call the original function with the argument

    wrapper_count_calls.num_calls = 0  # Initialize the call counter
    return wrapper_count_calls  # Return the wrapper function

# Applying the decorator to the function say_hello
@count_function
def say_hello():
    print("Say Hello")

# Calling the decorated function twice
say_hello()  # First call
say_hello()  # Second call
```

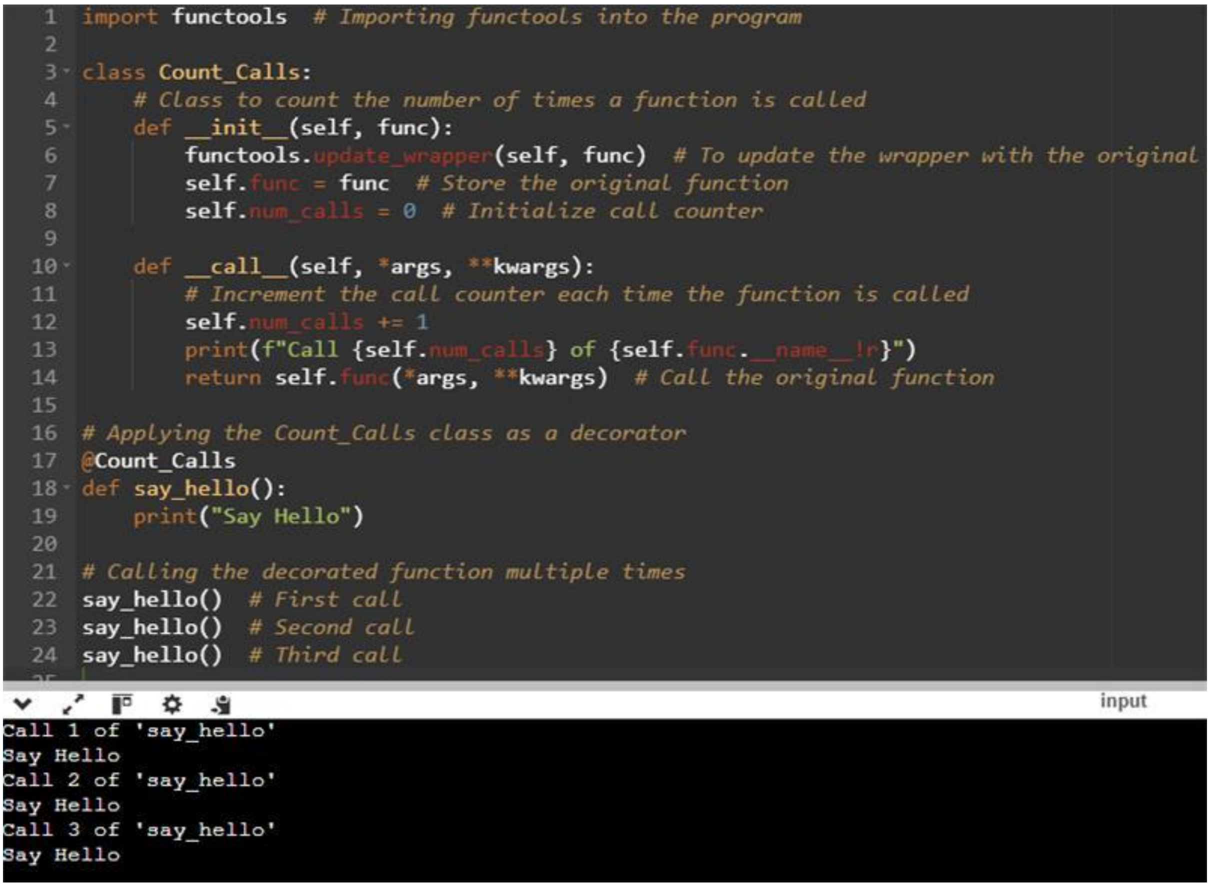
```
Call 1 of 'say_hello'
Say Hello
Call 2 of 'say_hello'
Say Hello
```

4.13.Classes as Decorators



```python
import functools  # Importing functools into the program

class Count_Calls:
    # Class to count the number of times a function is called
    def __init__(self, func):
        functools.update_wrapper(self, func)  # To update the wrapper with the original
        self.func = func  # Store the original function
        self.num_calls = 0  # Initialize call counter

    def __call__(self, *args, **kwargs):
        # Increment the call counter each time the function is called
        self.num_calls += 1
        print(f"Call {self.num_calls} of {self.func.__name__!r}")
        return self.func(*args, **kwargs)  # Call the original function

# Applying the Count_Calls class as a decorator
@Count_Calls
def say_hello():
    print("Say Hello")

# Calling the decorated function multiple times
say_hello()  # First call
say_hello()  # Second call
say_hello()  # Third call
```

```
Call 1 of 'say_hello'
Say Hello
Call 2 of 'say_hello'
Say Hello
Call 3 of 'say_hello'
Say Hello
```

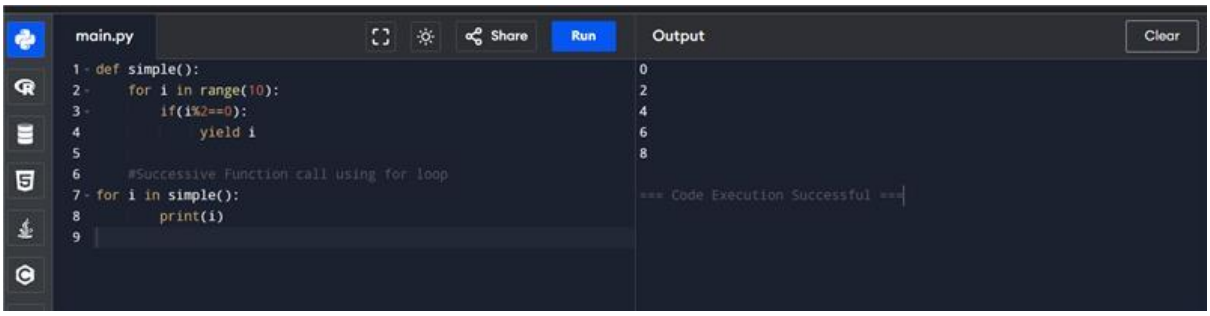


```python
import functools  # Importing functools into the program

class Count_Calls:
    # Class to count the number of times a function is called
    def __init__(self, func):
        functools.update_wrapper(self, func)  # To update the wrapper with the original
        self.func = func  # Store the original function
        self.num_calls = 0  # Initialize call counter

    def __call__(self, *args, **kwargs):
        # Increment the call counter each time the function is called
        self.num_calls += 1
        print(f"Call {self.num_calls} of {self.func.__name__!r}")
        return self.func(*args, **kwargs)  # Call the original function

# Applying the Count_Calls class as a decorator
@Count_Calls
def say_hello():
    print("Say Hello")

# Calling the decorated function multiple times
say_hello()  # First call
say_hello()  # Second call
say_hello()  # Third call
```

```
Call 1 of 'say_hello'
Say Hello
Call 2 of 'say_hello'
Say Hello
Call 3 of 'say_hello'
Say Hello
```

5) Python Generators

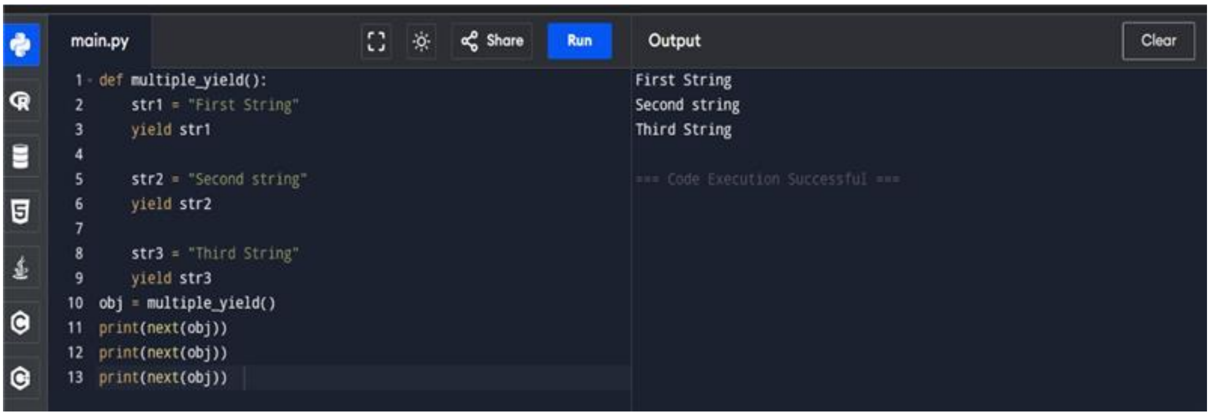
i. Create Generator function in Python



```python
def simple():
    for i in range(10):
        if(i%2==0):
            yield i

    #Successive Function call using for loop
for i in simple():
        print(i)

```

```
0
2
4
6
8

=== Code Execution Successful ===
```

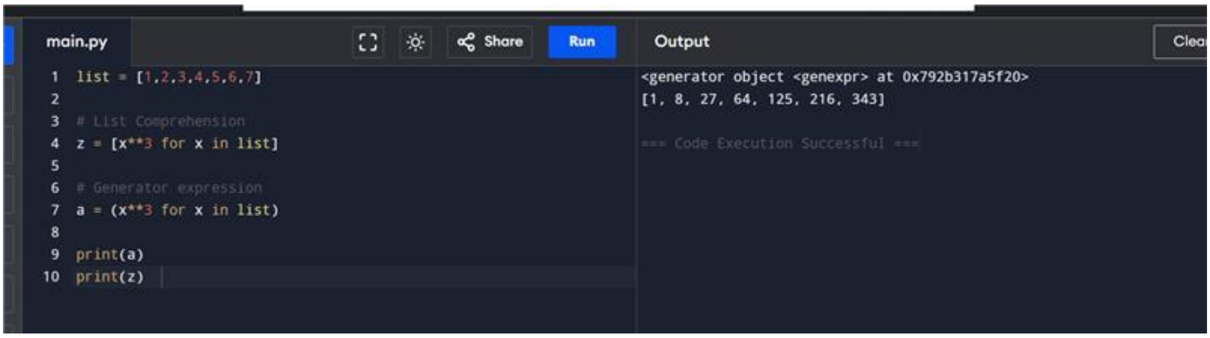
ii. yield vs return



```python
def multiple_yield():
    str1 = "First String"
    yield str1

    str2 = "Second string"
    yield str2

    str3 = "Third String"
    yield str3
obj = multiple_yield()
print(next(obj))
print(next(obj))
print(next(obj))
```

```
First String
Second string
Third String

=== Code Execution Successful ===
```

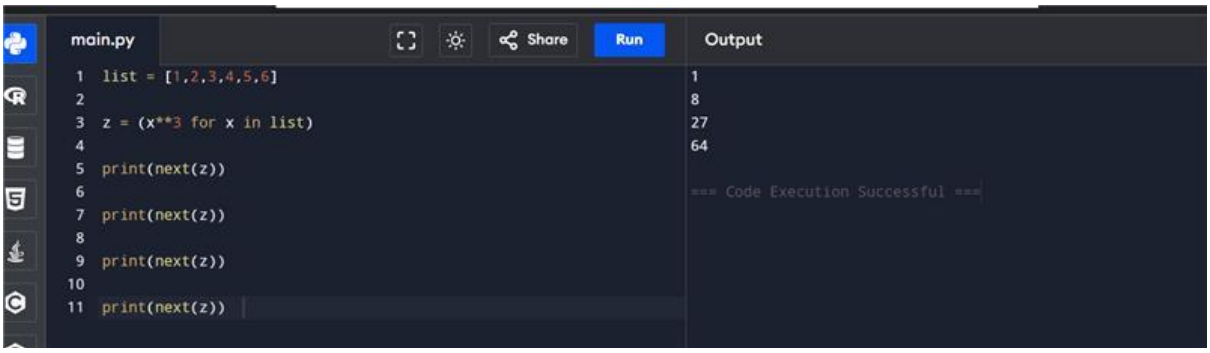
iii. Generator Expression



```python
list = [1,2,3,4,5,6,7]

# List Comprehension
z = [x**3 for x in list]

# Generator expression
a = (x**3 for x in list)

print(a)
print(z)
```

```
<generator object <genexpr> at 0x792b317a5f20>
[1, 8, 27, 64, 125, 216, 343]

=== Code Execution Successful ===
```

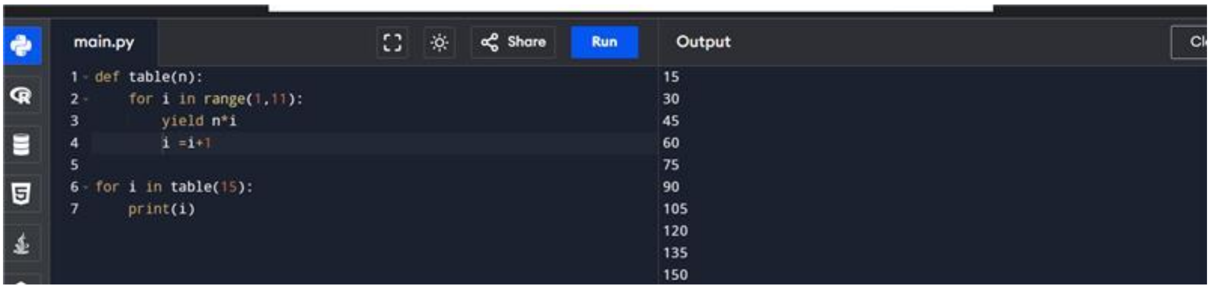
iv. Python next()



```python
list = [1,2,3,4,5,6]

z = (x**3 for x in list)

print(next(z))

print(next(z))

print(next(z))

print(next(z))
```

```
1
8
27
64

=== Code Execution Successful ===
```

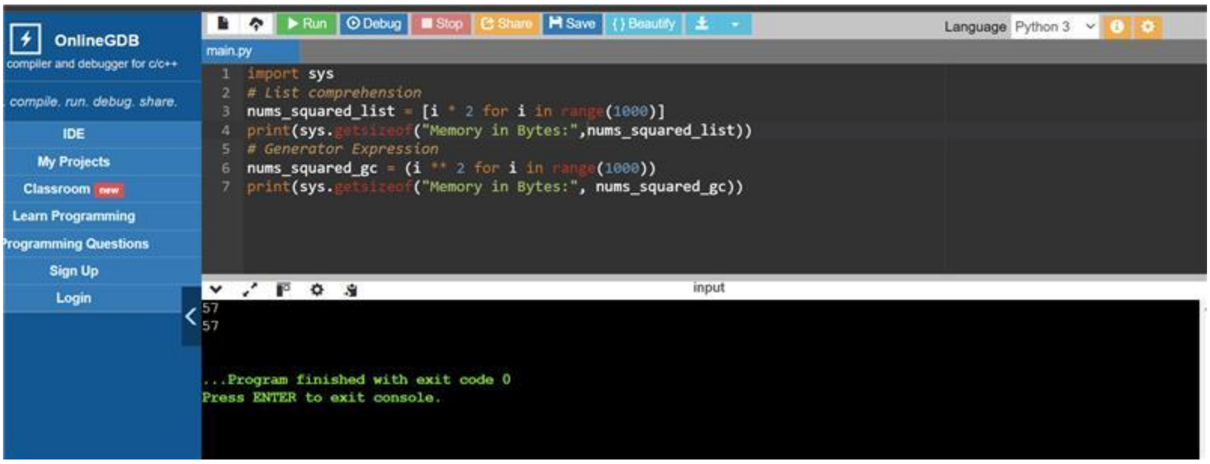
v. **Table program using generators**



```python
def table(n):
    for i in range(1,11):
        yield n*i
        i =i+1

for i in table(15):
    print(i)
```

```
15
30
45
60
75
90
105
120
135
150
```

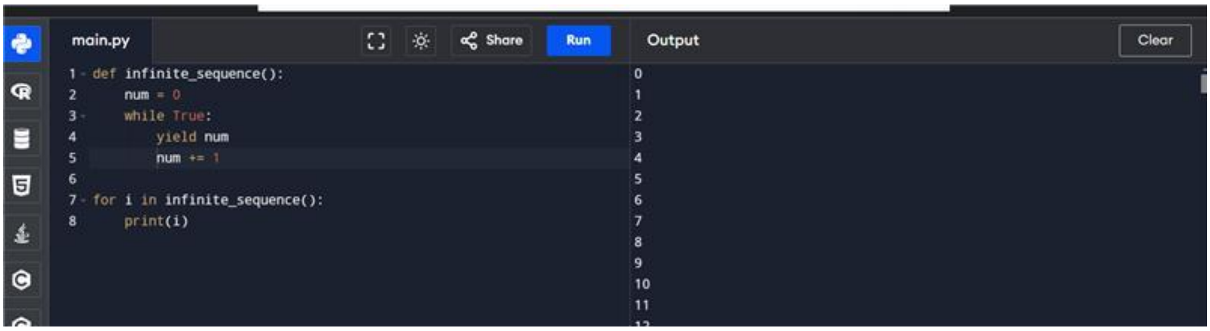
vi. **Memory efficient**



```python
import sys
# List comprehension
nums_squared_list = [i * 2 for i in range(1000)]
print(sys.getsizeof("Memory in Bytes:",nums_squared_list))
# Generator Expression
nums_squared_gc = (i ** 2 for i in range(1000))
print(sys.getsizeof("Memory in Bytes:", nums_squared_gc))
```

```
57
57


...Program finished with exit code 0
Press ENTER to exit console.
```

vii. **Python infinite program using generators**



```python
def infinite_sequence():
    num = 0
    while True:
        yield num
        num += 1

for i in infinite_sequence():
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
10
11
```