

KARANAM YESWANTH TEJA

289224

1) PYTHON FUNCTIONS

1.1) Example Python Code for User-Defined function



The screenshot shows a Python IDE with a file named 'main.py'. The code defines a function 'square' that takes a number and returns its square. It then calls the function with the argument 6 and prints the result. The output window shows the result 'The square of the given number is: 36' and a success message.

```
1 # Example Python Code for User-Defined function
2 def square( num ):
3     """
4     This function computes the square of the number.
5     """
6     return num**2
7 object_ = square(6)
8 print( "The square of the given number is: ", object_ )
```

Output: The square of the given number is: 36
=== Code Execution Successful ===

1.2) Example Python Code for calling a function



The screenshot shows a Python IDE with a file named 'main.py'. The code defines a function 'a_function' that takes a string and returns its length. It then calls the function with the strings 'Functions' and 'Python' and prints the results. The output window shows the results 'Length of the string Functions is: 9' and 'Length of the string Python is: 6' and a success message.

```
1 # Example Python Code for calling a function
2 # Defining a function
3 def a_function( string ):
4     "This prints the value of length of string"
5     return len(string)
6
7 # Calling the function we defined
8 print( "Length of the string Functions is: ", a_function(
9     "Functions" ) )
9 print( "Length of the string Python is: ", a_function( "Python" )
    )
```

Output: 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

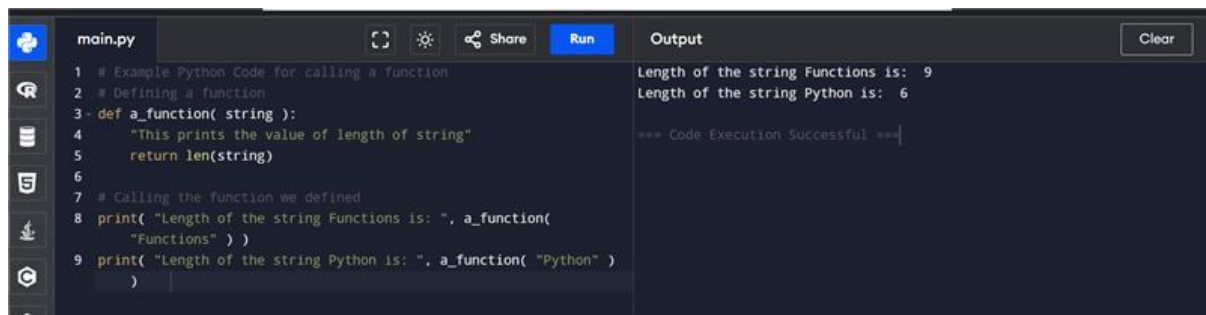


The screenshot shows a Python IDE with a file named 'main.py'. The code defines a function 'square' that takes a number and returns its square. It then calls the function with the argument 6 and prints the result. The output window shows the result 'The square of the given number is: 36' and a success message.

```
1 # Example Python Code for User-Defined function
2 def square( num ):
3     """
4     This function computes the square of the number.
5     """
6     return num**2
7 object_ = square(6)
8 print( "The square of the given number is: ", object_ )
```

Output: The square of the given number is: 36
=== Code Execution Successful ===

1.4) Example Python Code for calling a function



The screenshot shows a Python IDE with a file named 'main.py'. The code defines a function 'a_function' that takes a string and returns its length. It then calls this function twice: first with 'Functions' and then with 'Python'. The output pane shows the results of these calls: 'Length of the string Functions is: 9' and 'Length of the string Python is: 6'. A success message '=== Code Execution Successful ===' is also displayed.

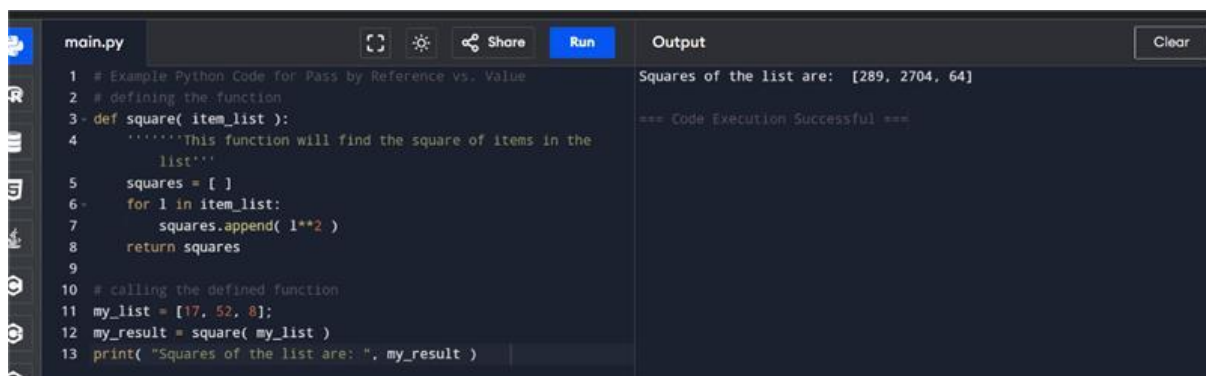
```
1 # Example Python Code for calling a function
2 # Defining a function
3 def a_function( string ):
4     "This prints the value of length of string"
5     return len(string)
6
7 # Calling the function we defined
8 print( "Length of the string Functions is: ", a_function(
9     "Functions" ) )
10 print( "Length of the string Python is: ", a_function( "Python" )
11     )
```

Output

```
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



The screenshot shows a Python IDE with a file named 'main.py'. The code defines a function 'square' that takes a list and returns a new list with the squares of its elements. It then calls this function with a list [17, 52, 8]. The output pane shows the result: 'Squares of the list are: [289, 2704, 64]'. A success message '=== Code Execution Successful ===' is also displayed.

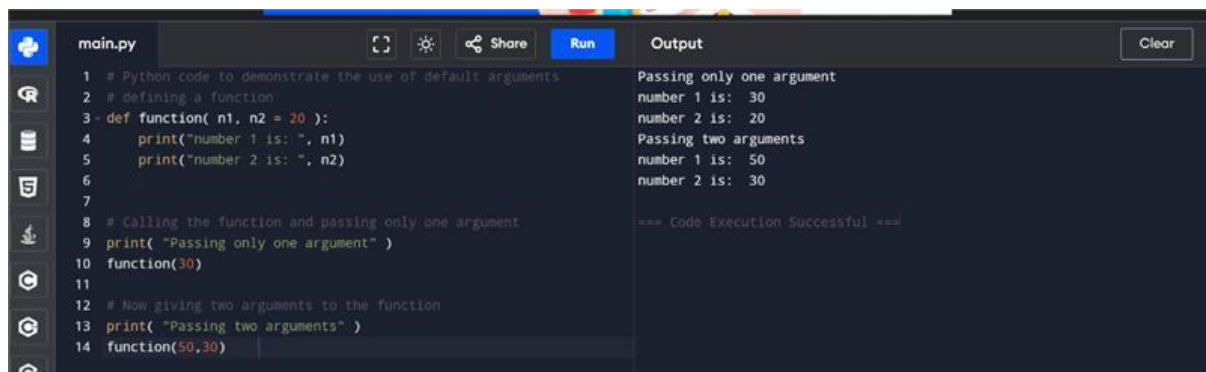
```
1 # Example Python Code for Pass by Reference vs. Value
2 # defining the function
3 def square( item_list ):
4     "This function will find the square of items in the
5     list"
6     squares = [ ]
7     for l in item_list:
8         squares.append( l**2 )
9     return squares
10
11 # calling the defined function
12 my_list = [17, 52, 8];
13 my_result = square( my_list )
14 print( "Squares of the list are: ", my_result )
```

Output

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

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

1.6) Default Arguments



The screenshot shows a Python IDE with a file named 'main.py'. The code defines a function 'function' with two arguments, 'n1' and 'n2', where 'n2' has a default value of 20. It then calls this function twice: first with only 'n1' (30) and then with both 'n1' (50) and 'n2' (30). The output pane shows the results of these calls: 'Passing only one argument' followed by 'number 1 is: 30' and 'number 2 is: 20', and 'Passing two arguments' followed by 'number 1 is: 50' and 'number 2 is: 30'. A success message '=== Code Execution Successful ===' is also displayed.

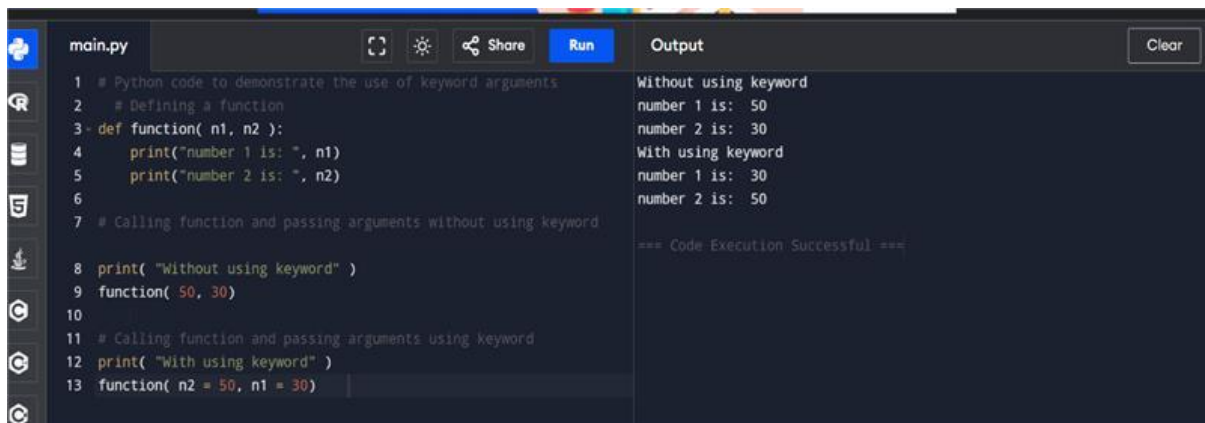
```
1 # Python code to demonstrate the use of default arguments
2 # defining a function
3 def function( n1, n2 = 20 ):
4     print("number 1 is: ", n1)
5     print("number 2 is: ", n2)
6
7
8 # Calling the function and passing only one argument
9 print( "Passing only one argument" )
10 function(30)
11
12 # Now giving two arguments to the function
13 print( "Passing two arguments" )
14 function(50,30)
```

Output

```
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



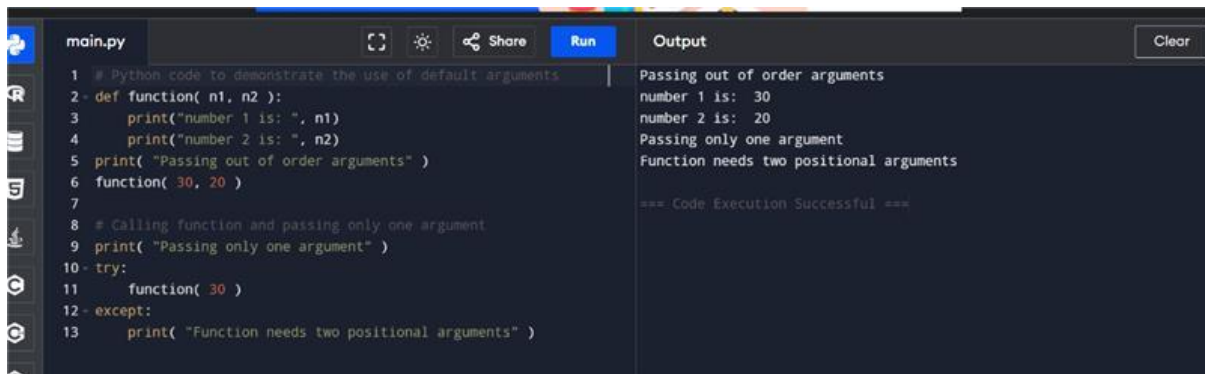
The screenshot shows a code editor with a file named `main.py`. The code defines a function `function(n1, n2)` that prints the values of `n1` and `n2`. It then demonstrates two ways to call the function: without keyword arguments and with keyword arguments. The output window shows the results of these calls.

```
1 # Python code to demonstrate the use of keyword arguments
2 # Defining a function
3 def function( n1, n2 ):
4     print("number 1 is: ", n1)
5     print("number 2 is: ", n2)
6
7 # Calling function and passing arguments without using keyword
8
9 print( "Without using keyword" )
10 function( 50, 30 )
11
12 # Calling function and passing arguments using keyword
13 print( "With using keyword" )
14 function( n2 = 50, n1 = 30 )
```

Output:

```
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



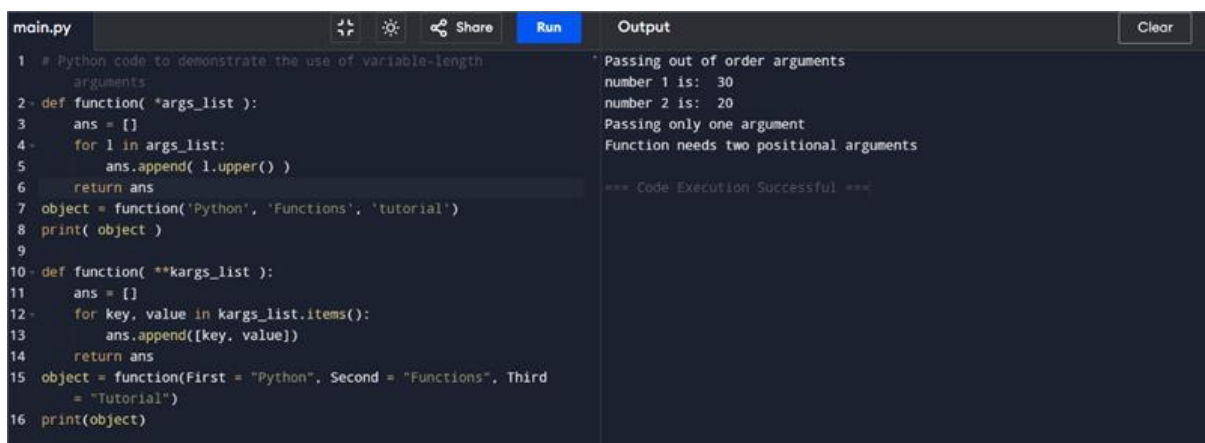
The screenshot shows a code editor with a file named `main.py`. The code defines a function `function(n1, n2)` that prints the values of `n1` and `n2`. It then demonstrates two ways to call the function: with two positional arguments and with one positional argument. The output window shows the results of these calls.

```
1 # Python code to demonstrate the use of default arguments
2 def function( n1, n2 ):
3     print("number 1 is: ", n1)
4     print("number 2 is: ", n2)
5
6 # Calling function and passing arguments without using keyword
7
8 print( "Passing out of order arguments" )
9 function( 30, 20 )
10
11 # Calling function and passing only one argument
12 print( "Passing only one argument" )
13 try:
14     function( 30 )
15 except:
16     print( "Function needs two positional arguments" )
```

Output:

```
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



The screenshot shows a code editor with a file named `main.py`. The code defines two functions: `function(*args_list)` and `function(**kargs_list)`. The first function takes a list of arguments and prints them. The second function takes a dictionary of keyword arguments and prints them. It then demonstrates two ways to call the functions: with a list of arguments and with a dictionary of keyword arguments. The output window shows the results of these calls.

```
1 # Python code to demonstrate the use of variable-length
2 # arguments
3 def function( *args_list ):
4     ans = []
5     for l in args_list:
6         ans.append( l.upper() )
7     return ans
8
9 object = function('Python', 'Functions', 'tutorial')
10 print( object )
11
12 # Calling function and passing arguments using keyword
13
14 def function( **kargs_list ):
15     ans = []
16     for key, value in kargs_list.items():
17         ans.append([key, value])
18     return ans
19
20 object = function(First = "Python", Second = "Functions", Third
21 = "Tutorial")
22 print(object)
```

Output:

```
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

```
main.py  [Icons] [Share] [Run] [Clear]
1 # Python code to demonstrate the use of return statements
2
3 def square( num ):
4     return num**2
5
6 # Calling function and passing arguments..
7 print( "With return statement" )
8 print( square( 52 ) )
9
10 # Defining a function without return statement
11 def square( num ):
12     num**2
13
14 # Calling function and passing arguments..
15 print( "Without return statement" )
16 print( square( 52 ) )
```

Output

```
With return statement
2704
Without return statement
None

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

1.11) The Anonymous Functions

```
main.py  [Icons] [Share] [Run] [Clear]
1 # Python code to demonstrate anonymous functions
2 # Defining a function
3 lambda_ = lambda argument1, argument2: argument1 + argument2;
4
5 # Calling the function and passing values
6 print( "Value of the function is : ", lambda_( 20, 30 ) )
7 print( "Value of the function is : ", lambda_( 40, 50 ) )
```

Output

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

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

1.12) Scope and Lifetime of Variables

```
main.py  [Icons] [Share] [Run] [Clear]
1 # Python code to demonstrate scope and lifetime of variables
2 #defining a function to print a number.
3 def number( ):
4     num = 50
5     print( "Value of num inside the function: ", num)
6
7 num = 10
8 number()
9 print( "Value of num outside the function:", num)
```

Output

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

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

1.13) Python abs() Function

```
main.py  [Icons] [Share] [Run] [Clear]
# 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))
```

Output

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

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

1.14) Python bin() Function

```
main.py  [Icons] [Share] [Run] [Clear]
1 x = 10
2 y = bin(x)
3 print(y)
```

0b1010
=== Code Execution Successful ===

1.15) Python all() Function

```
main.py  [Icons] [Share] [Run] [Clear]
1 # all values true
2 k = [1, 3, 4, 6]
3 print(all(k))
4
5 k = [0, False]
6 print(all(k))
7
8 k = [1, 3, 7, 0]
9 print(all(k))
10
11 k = [0, False, 5]
12 print(all(k))
13
14 # empty iterable
15 k = []
16 print(all(k))
```

True
False
False
False
True

=== Code Execution Successful ===

1.16) Python bool()

```
main.py  [Icons] [Share] [Run] [Clear]
1 test1 = []
2 print(test1, 'is', bool(test1))
3 test1 = [0]
4 print(test1, 'is', bool(test1))
5 test1 = 0.0
6 print(test1, 'is', bool(test1))
7 test1 = None
8 print(test1, 'is', bool(test1))
9 test1 = True
10 print(test1, 'is', bool(test1))
11 test1 = 'Easy string'
12 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()

```
main.py  [Icons] [Share] [Run] [Clear]
1 string = "Hello World."
2 array = bytes(string, 'utf-8')
3 print(array)
```

b'Hello World.'
=== Code Execution Successful ===

1.18) Python compile() Function

```
main.py  Run  Output
1 code_str = 'x=5\ny=10\nprint("sum =",x+y)'
2 code = compile(code_str, 'sum.py', 'exec')
3 print(type(code)) # <class 'code'>
4 exec(code)
5
6
```

<class 'code'>
sum = 15

=== Code Execution Successful ===

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

```
main.py  Run  Output  Clear
1 #exec
2 x = 8
3 exec('print(x==8)')
4 exec('print(x+4)')
5
6
7 #sum function
8 s = sum([1, 2, 4 ])
9 print(s)
10
11 s = sum([1, 2, 4], 10)
12 print(s)
```

True
12
7
17

=== Code Execution Successful ===

1.20) Any() function

```
main.py  Run  Output  Clear
1 l= [4, 3, 2, 0]
2 print(any(l))
3
4 l = [0, False]
5 print(any(l))
6
7 l = [0, False, 5]
8 print(any(l))
9
10 l = []
11 print(any(l))
```

True
False
True
False

=== Code Execution Successful ===

1.21) ASCII() function

```
Programiz Python Online Compiler  Programiz PRO >
main.py  Run  Output  Clear
1 normalText = 'Python is interesting'
2 print(ascii(normalText))
3
4 otherText = 'Pythøn is interesting'
5 print(ascii(otherText))
6
7 print('Pyth\xfdn is interesting')
```

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

=== Code Execution Successful ===

1.22) Byte array() and eval() functions

```
main.py  [Icons] Share Run Output Clear
1 string = "Python is a programming language."
2
3 # string with encoding 'utf-8'
4 arr = bytearray(string, 'utf-8')
5 print(arr)
6
7 #eval()
8 x = 8
9 print(eval('x + 1'))
```

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

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

1.23) Float() function

```
main.py  [Icons] Share Run Output Clear
1 # for integers
2 print(float(9))
3
4 # for floats
5 print(float(8.19))
6
7 # for string floats
8 print(float("-24.27"))
9
10 # for string floats with whitespaces
11 print(float("  -17.19\n"))
12
13 # string float error
14 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() function

```
main.py  [Icons] Share Run Output Clear
1 # d, f and b are a type
2
3 # integer
4 print(format(123, "d"))
5
6 # float arguments
7 print(format(123.4567898, "f"))
8
9 # binary format
10 print(format(12, "b"))
11
12 # tuple of letters
13 letters = ('m', 'r', 'o', 't', 's')
14
15 fSet = frozenset(letters)
16 print('Frozen set is:', fSet)
17 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

```
main.py  Run  Output
1 class Details:
2     age = 22
3     name = "Phill"
4
5 details = Details()
6 print('The age is:', getattr(details, "age"))
7 print('The age is:', details.age)
8
9 age = 22
10 globals()['age'] = 22
11 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

```
main.py  Run  Output  Clear
1 # list of numbers
2 list = [1,2,3,4,5]
3
4 listIter = iter(list)
5
6 # prints '1'
7 print(next(listIter))
8
9 # prints '2'
10 print(next(listIter))
11
12 # prints '3'
13 print(next(listIter))
14
15 # prints '4'
16 print(next(listIter))
17
18 # prints '5'
19 print(next(listIter))
```

1
2
3
4
5
=== Code Execution Successful ===

1.27) Python list()

```
main.py  Run  Output  Clear
1 # empty list
2 print(list())
3
4 # string
5 String = 'abcde'
6 print(list(String))
7
8 # tuple
9 Tuple = (1,2,3,4,5)
10 print(list(Tuple))
11 # list
12 List = [1,2,3,4,5]
13 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

```
main.py  [Icons] [Share] [Run] [Clear]
1- def localsAbsent():
2     return locals()
3
4- def localsPresent():
5     present = True
6     return locals()
7
8 print('localsNotPresent:', localsAbsent())
9 print('localsPresent:', localsPresent())
```

Output

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

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

1.29) Map() function

```
main.py  [Icons] [Share] [Run] [Clear]
1- def calculateAddition(n):
2     return n+n
3
4 numbers = (1, 2, 3, 4)
5 result = map(calculateAddition, numbers)
6 print(result)
7
8 # converting map object to set
9 numbersAddition = set(result)
10 print(numbersAddition)
```

Output

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

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

1.30) Python memoryview() Function

```
main.py  [Icons] [Share] [Run] [Clear]
1 # A random bytearray
2 randomByteArray = bytearray('ABC', 'utf-8')
3
4 mv = memoryview(randomByteArray)
5
6 # access the memory view's zeroth index
7 print(mv[0])
8
9 # It create byte from memory view
10 print(bytes(mv[0:2]))
11
12 # It create list from memory view
13 print(list(mv[0:3]))
```

Output

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

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

1.31) Python chr() Function

```
main.py  [Icons] [Share] [Run] [Clear]
1 # Calling function
2 result = chr(102) # It returns string representation of a char
3 result2 = chr(112)
4 # Displaying result
5 print(result)
6 print(result2)
7 # Verify, is it string type?
8 print("is it string type:", type(result) is str)
```

Output

```
f
p
is it string type: True

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

1.32) Python complex fun()

```
main.py  [Icons] [Share] [Run] [Output] [Clear]
1 # Python complex() function example
2 # Calling function
3 a = complex(1) # Passing single parameter
4 b = complex(1,2) # Passing both parameters
5 # Displaying result
6 print(a)
7 print(b)
```

(1+0j)
(1+2j)
=== Code Execution Successful ===

1.33) Python delattr() Function

```
main.py  [Icons] [Share] [Run] [Output] [Clear]
1 class Student:
2     id = 101
3     name = "Pranshu"
4     email = "pranshu@abc.com"
5 # Declaring function
6 def getinfo(self):
7     print(self.id, self.name, self.email)
8 s = Student()
9 s.getinfo()
10 delattr(Student, 'course') # Removing attribute which is not available
11 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()

```
main.py  [Icons] [Share] [Run] [Output] [Clear]
1 # Calling function
2 result = enumerate([1,2,3])
3 # Displaying result
4 print(result)
5 print(list(result))
```

<enumerate object at 0x7d5ae41d36a0>
[(0, 1), (1, 2), (2, 3)]
=== Code Execution Successful ===

1.35) Python dict()

```
main.py  [Icons] [Share] [Run] [Output] [Clear]
1 # Calling function
2 result = dict() # returns an empty dictionary
3 result2 = dict(a=1,b=2)
4 # Displaying result
5 print(result)
6 print(result2)
```

{}
{'a': 1, 'b': 2}
=== Code Execution Successful ===

1.36) Python filter ()

```
main.py  [Icons] [Share] [Run] [Clear]
1 # Python filter() function example
2 def filterdata(x):
3     if x>5:
4         return x
5 # Calling function
6 result = filter(filterdata,(1,2,6))
7 # Displaying result
8 print(list(result))
```

Output

```
[6]
=== Code Execution Successful ===
```

1.37) Python hash()

```
main.py  [Icons] [Share] [Run] [Clear]
1 result = hash(21) # integer value
2 result2 = hash(22.2) # decimal value
3 # Displaying result
4 print(result)
5 print(result2)
```

Output

```
21
461168601842737174
=== Code Execution Successful ===
```

1.38) Python min()

```
main.py  [Icons] [Share] [Run] [Clear]
1 # Calling function
2 small = min(2225,325,2025) # returns smallest element
3 small2 = min(1000.25,2025.35,5625.36,10052.50)
4 # Displaying result
5 print(small)
6 print(small2)
```

Output

```
325
1000.25
=== Code Execution Successful ===
```

1.39) Python hex() and set() function

```
main.py  [Icons] [Share] [Run] [Clear]
1 result = set() # empty set
2 result2 = set('12')
3 result3 = set('javatpoint')
4 # Displaying result
5 print(result)
6 print(result2)
7 print(result3)
8
9 # Calling function
10 result = hex(1)
11 # integer value
12 result2 = hex(342)
13 # Displaying result
14 print(result)
15 print(result2)
```

Output

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

1.40) Python Id()

```
main.py  Run Output Clear
# Calling function
val1 = id("Javatpoint") # string object
val2 = id(1200) # integer object
val3 = id([25,336,95,236,92,3225]) # List object
# Displaying result
print(val1)
print(val2)
print(val3)
```

```
137546140603312
137546141969936
137546143812032

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

1.41) Python setattr()

```
main.py  Run Output Clear
1 class Student:
2     id = 0
3     name = ""
4
5     def __init__(self, id, name):
6         self.id = id
7         self.name = name
8
9 student = Student(102,"Sohan")
10 print(student.id)
11 print(student.name)
12 #print(student.email) product error
13 setattr(student, 'email','sohan@abc.com') # adding new attribute
14 print(student.email)
```

```
102
Sohan
sohan@abc.com

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

1.42) Python slice() and sorted()

```
main.py  Run Output Clear
1 # Calling function
2 result = slice(5) # returns slice object
3 result2 = slice(0.5,3) # returns slice object
4 # Displaying result
5 print(result)
6 print(result2)
7
8 str = "javatpoint" # declaring string
9 # Calling function
10 sorted1 = sorted(str) # sorting string
11 # Displaying result
12 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()

```
main.py  [Icons] [Share] [Run] Output
1 number = iter([256, 32, 82]) # Creating iterator
2 # Calling function
3 item = next(number)
4 # Displaying result
5 print(item)
6 # second item
7 item = next(number)
8 print(item)
9 # third item
10 item = next(number)
11 print(item)
```

256
32
82
=== Code Execution Successful ===

1.44) Python input()

```
main.py  [Icons] [Share] [Run] Output
1 # Calling function
2 val = input("Enter a value: ")
3 # Displaying result
4 print("You entered:",val)
```

Enter a value:

1.45) Python int()

```
main.py  [Icons] [Share] [Run] Output
1 # Calling function
2 val = int(10) # integer value
3 val2 = int(10.52) # float value
4 val3 = int('10') # string value
5 # Displaying result
6 print("integer values :",val, val2, val3)
```

integer values : 10 10 10
=== Code Execution Successful ===

1.46) Python instance()

```
main.py  [Icons] [Share] [Run] Output
1 class Student:
2     id = 101
3     name = "John"
4     def __init__(self, id, name):
5         self.id=id
6         self.name=name
7
8 student = Student(1010,"John")
9 lst = [12,34,5,6,767]
10 # Calling function
11 print(isinstance(student, Student)) # isinstance of Student class
12 print(isinstance(lst, Student))
```

True
False
=== Code Execution Successful ===

1.47) Python ord() and pow() function

```
main.py  Run Output Clear
1
2 print(ord('8'))
3 # Code point of an alphabet
4 print(ord('R'))
5 # Code point of a character
6 print(ord('&'))
7
8 # positive x, positive y (x**y)
9 print(pow(4, 2))
10
11 # negative x, positive y
12 print(pow(-4, 2))
13
14 # positive x, negative y (x**-y)
15 print(pow(4, -2))
16
17 # negative x, negative y
18 print(pow(-4, -2))
```

56
82
38
16
16
0.0625
0.0625
=== Code Execution Successful ===

1.48) Python reversed()

```
main.py  Run Output Clear
1 # for string
2 String = 'Java'
3 print(list(reversed(String)))
4
5 # for tuple
6 Tuple = ('J', 'a', 'v', 'a')
7 print(list(reversed(Tuple)))
8
9 # for range
10 Range = range(8, 12)
11 print(list(reversed(Range)))
12
13 # for list
14 List = [1, 2, 7, 5]
15 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()

```
main.py  Run Output Clear
1 # for integers
2 print(round(10))
3
4 # for floating point
5 print(round(10.8))
6
7 # even choice
8 print(round(6.6))
9
10 class Python:
11     def __init__(self, x = 7, y = 9):
12         self.x = x
13         self.y = y
14
15 InstanceOfPython = Python()
16 print(vars(InstanceOfPython))
```

10
11
7
{'x': 7, 'y': 9}
=== Code Execution Successful ===

1.50) Python type() function

```
main.py  Run  Output
1 List = [4, 5]
2 print(type(List))
3
4 Dict = {'four': 'five'}
5 print(type(Dict))
6
7 class Python:
8     a = 0
9
10 InstanceOfPython = Python()
11 print(type(InstanceOfPython))
```

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

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

1.51) Python isinstance()

```
main.py  Run  Output
1 class Rectangle:
2     def __init__(rectangleType):
3         print('Rectangle is a ', rectangleType)
4
5 class Square(Rectangle):
6     def __init__(self):
7         Rectangle.__init__(self, 'square')
8
9 print(isinstance(Square, Rectangle))
10 print(isinstance(Square, list))
11 print(isinstance(Square, (list, Rectangle)))
12 print(isinstance(Rectangle, (list, Rectangle)))
```

```
True
False
True
True

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

1.52) Python zip()

```
main.py  Run  Output
1 numList = [4,5, 6]
2 strList = ['four', 'five', 'six']
3
4 # No iterables are passed
5 result = zip()
6
7 # Converting iterator to list
8 resultList = list(result)
9 print(resultList)
10
11 # Two iterables are passed
12 result = zip(numList, strList)
13
14 # Converting iterator to set
15 resultSet = set(result)
16 print(resultSet)
```

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

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


1.53) Python lambda()

```
main.py  Run  Output
1 add = lambda num: num + 4
2 print( add(6) )
3
4 def add( num ):
5     return num + 4
6 print( add(6) )
7
8 a = lambda x, y : (x * y)
9 print(a(4, 5))
10
11 a = lambda x, y, z : (x + y + z)
12 print(a(4, 5, 5))
```

10
10
20
14

=== Code Execution Successful ===

1.54) Def vs lambda difference

```
main.py  Run  Output  Clear
1 # Python code to show the reciprocal of the given number to highlight
  the difference between def() and lambda().
2 def reciprocal( num ):
3     return 1 / num
4
5 lambda_reciprocal = lambda num: 1 / num
6
7 # using the function defined by def keyword
8 print( "Def keyword: ", reciprocal(6) )
9
10 # using the function defined by lambda keyword
11 print( "Lambda keyword: ", lambda_reciprocal(6) )
```

Def keyword: 0.16666666666666666
Lambda keyword: 0.16666666666666666

=== Code Execution Successful ===

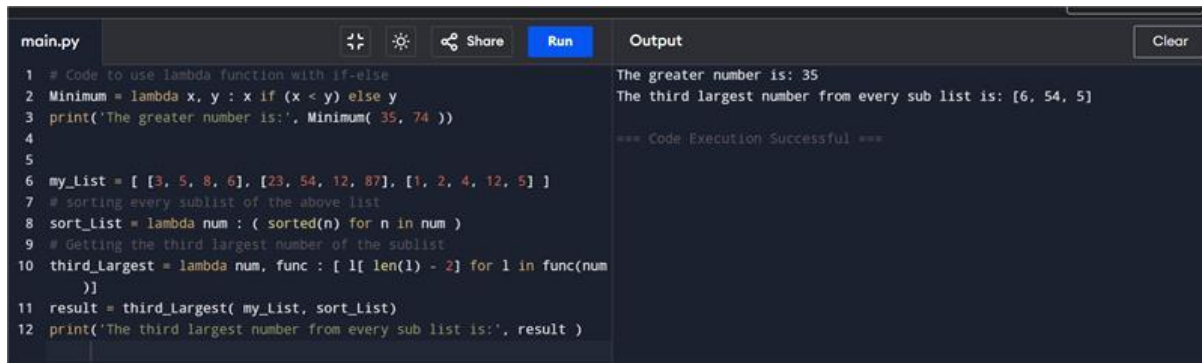
1.55) Using Lambda Function with filter(),map(),list comprehension()

```
main.py  Run  Output  Clear
1 # This code used to filter the odd numbers from the given list
2 list_ = [35, 12, 69, 55, 75, 14, 73]
3 odd_list = list(filter( lambda num: (num % 2 != 0) , list_ ))
4 print('The list of odd number is:',odd_list)
5
6 #Code to calculate the square of each number of a list using the map
  () function
7 numbers_list = [2, 4, 5, 1, 3, 7, 8, 9, 10]
8 squared_list = list(map( lambda num: num ** 2 , numbers_list ))
9 print( 'Square of each number in the given list:',squared_list )
10
11 squares = [lambda num = num: num ** 2 for num in range(0, 11)]
12 for square in squares:
13     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



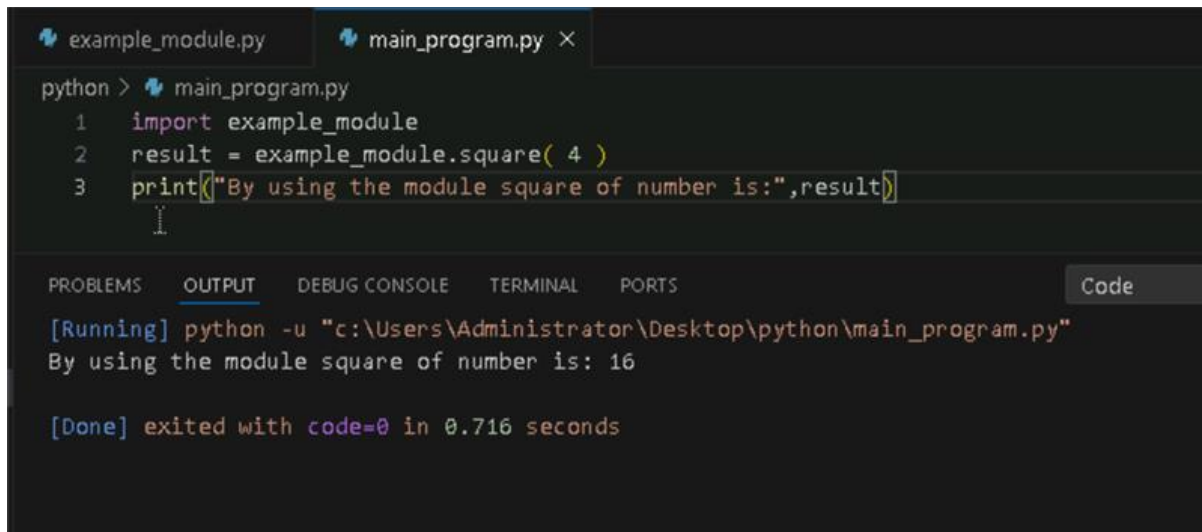
The screenshot shows a code editor with a file named `main.py`. The code defines a lambda function `Minimum` that takes two arguments `x` and `y` and returns the smaller one using a ternary operator. It then prints the result of `Minimum(35, 74)`. Next, it defines a list `my_list` containing three sublists. A lambda function `sort_list` is defined to sort each sublist. Another lambda function `third_largest` is defined to return the third largest element from a sorted list. Finally, it calls `third_largest` with `my_list` and `sort_list` and prints the result.

```
1 # Code to use lambda function with if-else
2 Minimum = lambda x, y : x if (x < y) else y
3 print('The greater number is:', Minimum( 35, 74 ))
4
5
6 my_list = [ [3, 5, 8, 6], [23, 54, 12, 87], [1, 2, 4, 12, 5] ]
7 # sorting every sublist of the above list
8 sort_list = lambda num : ( sorted(n) for n in num )
9 # Getting the third largest number of the sublist
10 third_largest = lambda num, func : [ l[ len(l) - 2 ] for l in func(num) ]
11 result = third_largest( my_list, sort_list )
12 print('The third largest number from every sub list is:', result )
```

The output shows: "The greater number is: 35" and "The third largest number from every sub list is: [6, 54, 5]". A message at the bottom indicates "Code Execution Successful".

2) MODULES

2.1.Importing modules

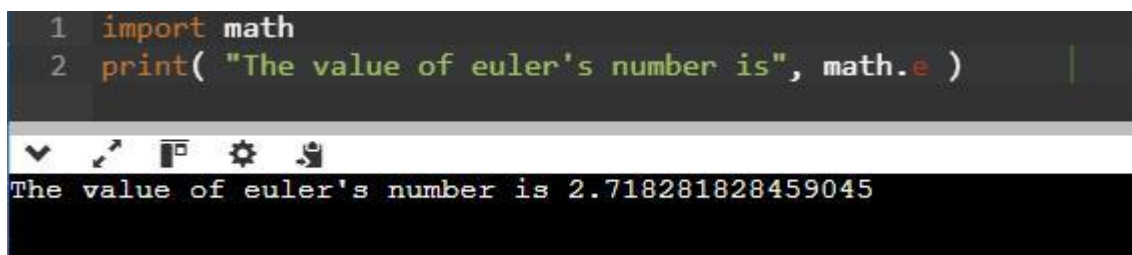


The screenshot shows a code editor with two files: `example_module.py` and `main_program.py`. The `main_program.py` file contains the following code:

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

The output shows the command `python -u "c:\Users\Administrator\Desktop\python\main_program.py"` being executed, followed by the output "By using the module square of number is: 16". A message at the bottom indicates "[Done] exited with code=0 in 0.716 seconds".

2.2. Importing and also Renaming:



The screenshot shows a code editor with the following code:

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

The output shows "The value of euler's number is 2.718281828459045".

2.3. Python from...import Statement:

```
1 from math import e, tau
2 print( "The value of tau constant is: ", tau )
3 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:

```
1 from math import *
2 # Here, we are accessing functions of math module without using the dot operator
3 print( "Calculating square root: ", sqrt(25) )
4 # here, we are getting the sqrt method and finding the square root of 25
5 print( "Calculating tangent of an angle: ", tan(pi/6) )
6
7
```

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

2.5. Locating Path of Modules:

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

Path of the sys module in the system is: ['/', '/usr/lib/python3.12.zip', '/usr/lib/python3.12', '/usr/lib/python3.12/lib-dynload', '/usr/local/lib/python3.12/dist-packages', '/usr/lib/python3/dist-packages']

2.6. The dir() Built-in Function:

```
1 print( "list of functions:\n", dir( str ), end=" " )
2
```

List of functions: ['_', 'add', 'class', 'contains', 'delattr', 'dir', 'doc', 'eq', 'format', 'ge', 'getattr', 'getitem', 'getnewargs', 'getstate', 'gt', 'hash', 'init', 'init_subclass', 'iter', 'le', 'len', 'lt', 'mod', 'mul', 'ne', 'new', 'reduce', 'reduce_ex', 'repr', 'rmul', 'rmod', 'setattr', 'sizeof', 'str', 'subclasshook', 'capitalize', 'casefold', 'center', 'count', 'encode', 'endwith', 'expandtabs', 'find', 'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isascii', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'removeprefix', 'removesuffix', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']

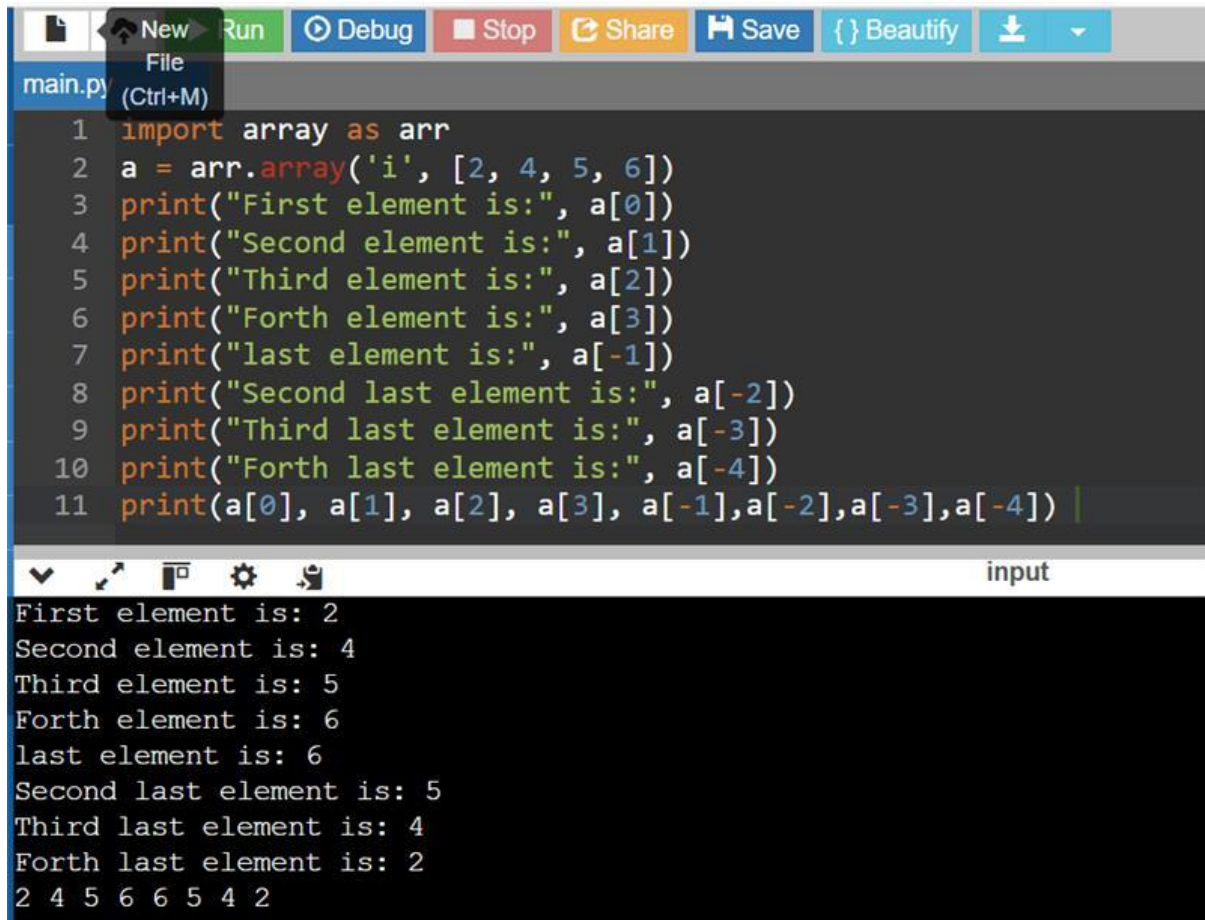
2.7. Namespaces and Scoping:

```
1 Number = 204
2 def AddNumber(): # here, we are defining a function with the name Add Number
3     # Here, we are accessing the global namespace
4     global Number
5     Number = Number + 200
6     print("The number is:", Number)
7 # here, we are printing the number after performing the addition
8 AddNumber() # here, we are calling the function
9 print("The number is:", Number)
```

The number is: 204
The number is: 404

3) PYTHON ARRAYS

3.1. Accessing array elements:



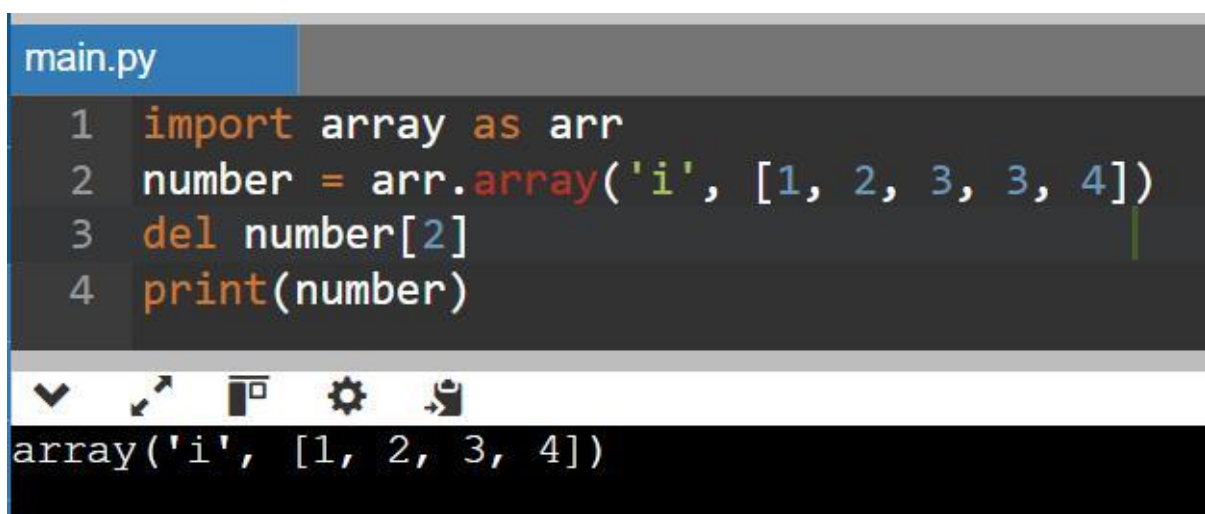
The screenshot shows a Python IDE with a toolbar at the top containing buttons for New, Run, Debug, Stop, Share, Save, Beautify, and a download icon. The file name 'main.py' is visible. The code in the editor is as follows:

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

The output window at the bottom shows the following results:

```
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



The screenshot shows a Python IDE with a toolbar at the top. The file name 'main.py' is visible. The code in the editor is as follows:

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

The output window at the bottom shows the following result:

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


3.3. Adding or changing the elements in Array

```
File
main.py (Ctrl+M)
1 import array as arr
2 numbers = arr.array('i', [1, 2, 3, 5, 7, 10])
3 numbers[0] = 0
4 print(numbers)
5 numbers[5] = 8
6 print(numbers)
7 numbers[2:5] = arr.array('i', [4, 6, 8])
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

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

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

4) PYTHON DECORATOR

4.1

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

input

Hii, welcome to function
Hii, welcome to function

2. Inner Function

```
main.py
1 def func():          # here, we are creating a function and passing the parameter
2     print("We are in first function") # Here, we are printing the data of function
3     def func1():      # here, we are creating a function and passing the parameter
4         print("This is first child function") # Here, we are printing the data of function 1
5     def func2():      # here, we are creating a function and passing the parameter
6         print("This is second child function") # Here, we are printing the data of
7     func1()
8     func2()
9 func()
```

input

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

4.3.

```
1 def add(x):          # he
2     return x+1        # he
3 def sub(x):           # he
4     return x-1        # h
5 def operator(func, x):
6     temp = func(x)
7     return temp
8 print(operator(sub,10))
9 print(operator(add,20))
```

9
21

4.4.

```
1 def hello():
2     def hi():
3         print("Hello")
4     return hi
5 new = hello()
6 new()
```

Hello

4.5. Decorating functions with parameters:

```
1 def divide(x,y):
2     print(x/y)
3 def outer_div(func):
4     def inner(x,y):
5         if x<y:
6             x,y = y,x
7         return func(x,y)
8     return inner
9 divide1 = outer_div(divide)
10 divide1(2,4)
```

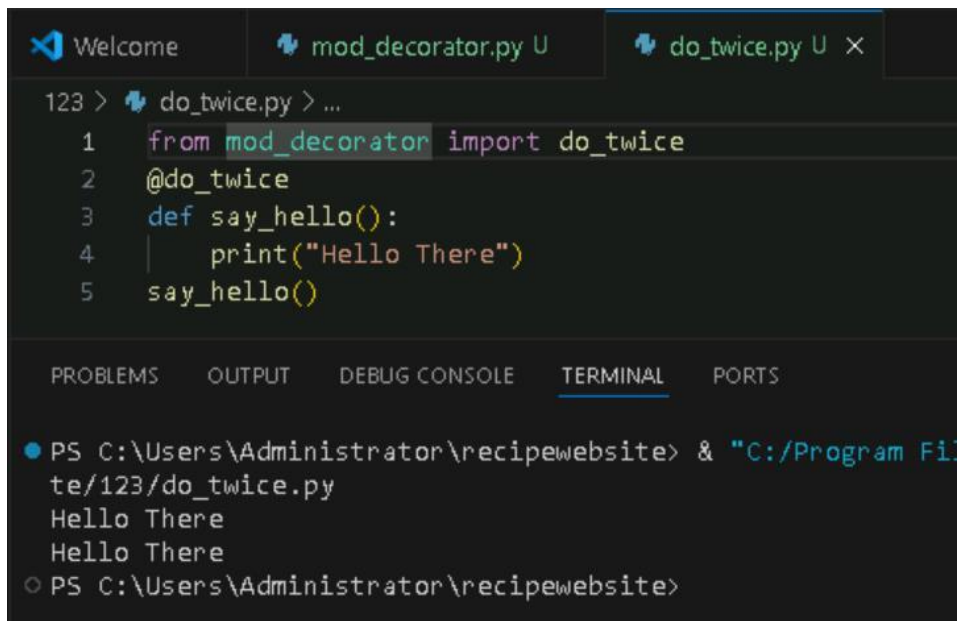
Hello

4.6. Syntactic Decorator:

```
1 def outer_div(func):
2     def inner(x, y):
3         if x < y:
4             x, y = y, x
5         return func(x, y)
6     return inner
7
8
9 @outer_div
10 def divide(x, y):
11     print(x / y)
12 divide(5, 10)
13
```

2.0

4.7.Reusing Decorator



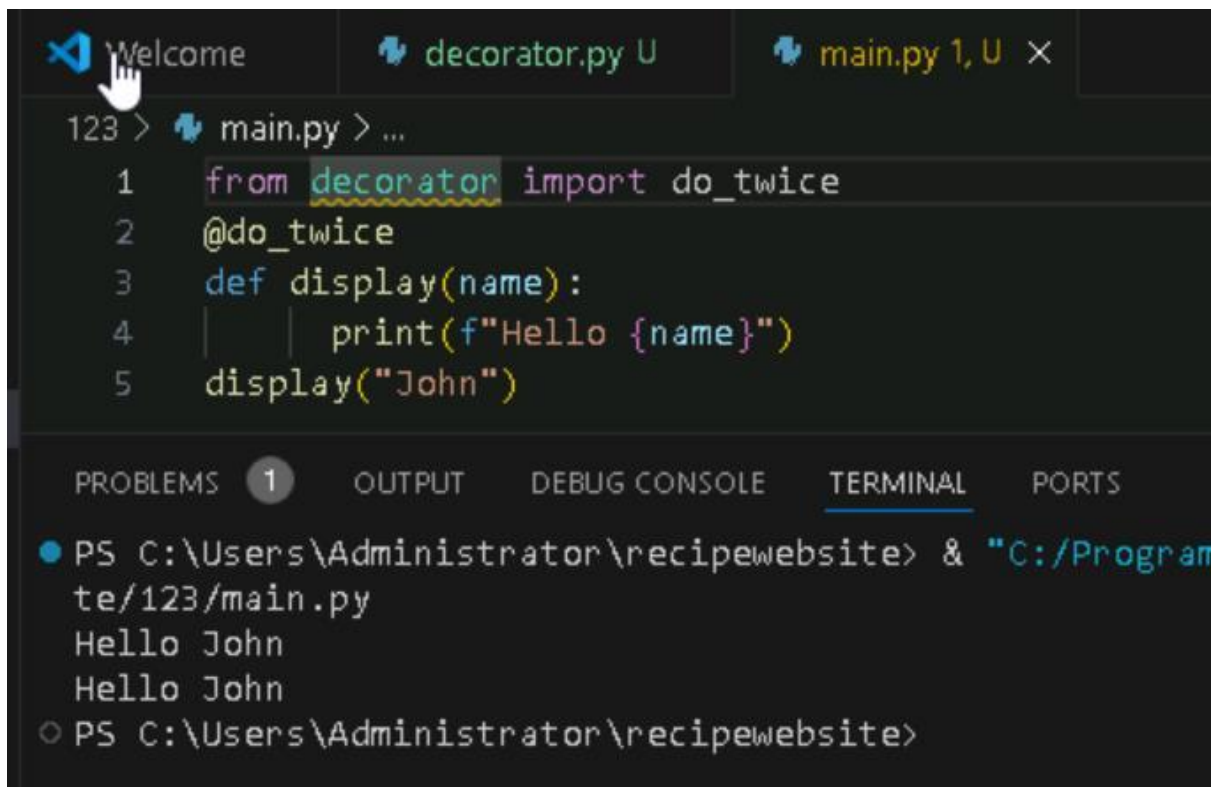
The screenshot shows the Visual Studio Code editor with two tabs: 'mod_decorator.py' and 'do_twice.py'. The 'do_twice.py' tab is active, displaying the following code:

```
1 from mod_decorator import do_twice
2 @do_twice
3 def say_hello():
4     print("Hello There")
5 say_hello()
```

Below the editor, the 'TERMINAL' tab is active, showing the command prompt output:

```
PS C:\Users\Administrator\recipewebsite> & "C:/Program Files/Python39/Python.exe" C:/Users/Administrator/recipewebsite/123/do_twice.py
Hello There
Hello There
PS C:\Users\Administrator\recipewebsite>
```

4.8.Python Decorator with Argument



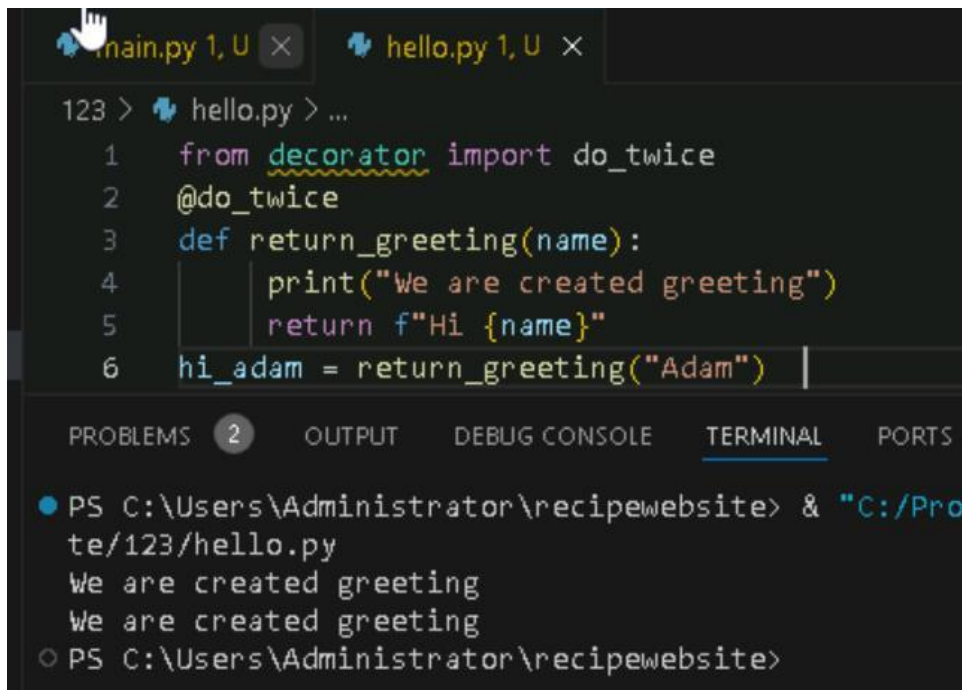
The screenshot shows the Visual Studio Code editor with three tabs: 'decorator.py', 'main.py', and 'main.py 1, U'. The 'main.py' tab is active, displaying the following code:

```
1 from decorator import do_twice
2 @do_twice
3 def display(name):
4     print(f"Hello {name}")
5 display("John")
```

Below the editor, the 'TERMINAL' tab is active, showing the command prompt output:

```
PS C:\Users\Administrator\recipewebsite> & "C:/Program Files/Python39/Python.exe" C:/Users/Administrator/recipewebsite/123/main.py
Hello John
Hello John
PS C:\Users\Administrator\recipewebsite>
```

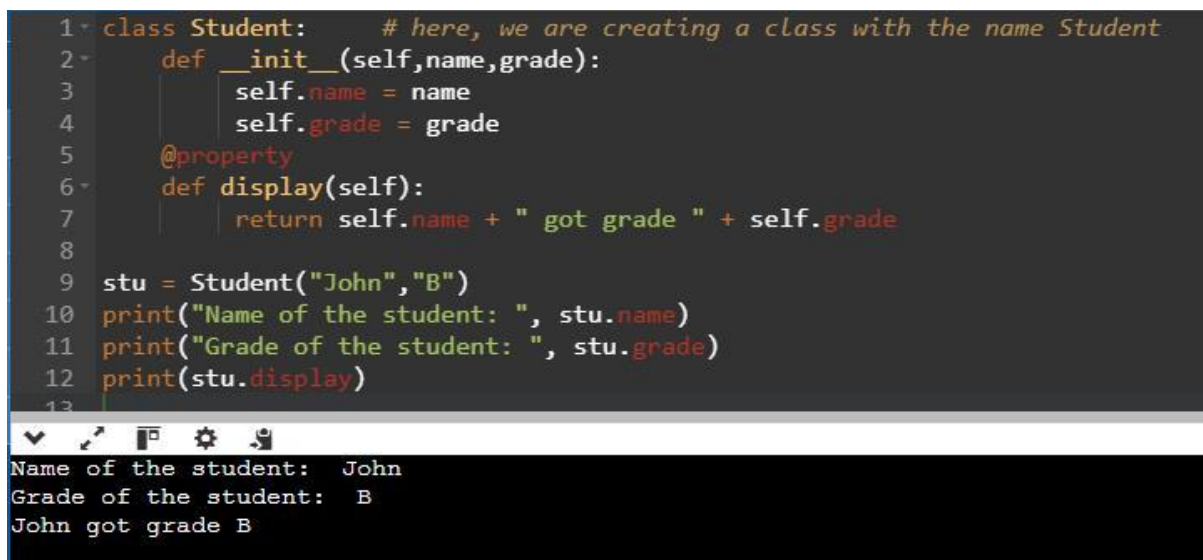
4.9. Returning Values from Decorated Functions



```
main.py 1, U x hello.py 1, U x
123 > hello.py > ...
1 from decorator import do_twice
2 @do_twice
3 def return_greeting(name):
4     print("We are created greeting")
5     return f"Hi {name}"
6 hi_adam = return_greeting("Adam")

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS
● 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



```
1 class Student: # here, we are creating a class with the name Student
2     def __init__(self, name, grade):
3         self.name = name
4         self.grade = grade
5     @property
6     def display(self):
7         return self.name + " got grade " + self.grade
8
9 stu = Student("John", "B")
10 print("Name of the student: ", stu.name)
11 print("Grade of the student: ", stu.grade)
12 print(stu.display)
13
Name of the student: John
Grade of the student: B
John got grade B
```



```
1 class Person: # here, we are creating a class with the name Student
2     @staticmethod
3     def hello(): # here, we are defining a function hello
4         print("Hello Peter")
5 per = Person()
6 per.hello()
7 Person.hello()
Hello Peter
Hello Peter
```

4.11.Decorator with Arguments

```
1 import functools # Importing functools into the program
2
3 def repeat(num): # Defining the repeat function that takes 'num'
4     # Creating and returning the decorator function
5     def decorator_repeat(func):
6         @functools.wraps(func) # Using functools.wraps to preserve
7         def wrapper(*args, **kwargs):
8             for _ in range(num): # Looping 'num' times to repeat
9                 value = func(*args, **kwargs) # Calling the original
10                return value # Returning the value after the loop
11            return wrapper # Returning the wrapper function
12
13    return decorator_repeat
14
15 @repeat(num=5)
16 def function1(name):
17     print(f"{name}")
18
19 function1("John")
20
```

John
John
John
John
John

4.12.Stateful Decorators

```
1 import functools # Importing functools into the program
2
3 def count_function(func):
4     # Defining the decorator function that counts the number of calls
5     @functools.wraps(func) # Preserving the metadata of the original function
6     def wrapper_count_calls(*args, **kwargs):
7         wrapper_count_calls.num_calls += 1 # Increment the call count
8         print(f"Call {wrapper_count_calls.num_calls} of {func.__name__}")
9         return func(*args, **kwargs) # Call the original function with the arguments
10
11    wrapper_count_calls.num_calls = 0 # Initialize the call counter
12    return wrapper_count_calls # Return the wrapper function
13
14 # Applying the decorator to the function say_hello
15 @count_function
16 def say_hello():
17     print("Say Hello")
18
19 # Calling the decorated function twice
20 say_hello() # First call
21 say_hello() # Second call
22
```

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

4.13. Classes as Decorators

```
1 import functools # Importing functools into the program
2
3 class Count_Calls:
4     # Class to count the number of times a function is called
5     def __init__(self, func):
6         functools.update_wrapper(self, func) # To update the wrapper with the original
7         self.func = func # Store the original function
8         self.num_calls = 0 # Initialize call counter
9
10    def __call__(self, *args, **kwargs):
11        # Increment the call counter each time the function is called
12        self.num_calls += 1
13        print(f"Call {self.num_calls} of {self.func.__name__!r}")
14        return self.func(*args, **kwargs) # Call the original function
15
16 # Applying the Count_Calls class as a decorator
17 @Count_Calls
18 def say_hello():
19     print("Say Hello")
20
21 # Calling the decorated function multiple times
22 say_hello() # First call
23 say_hello() # Second call
24 say_hello() # Third call
25
```

input

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

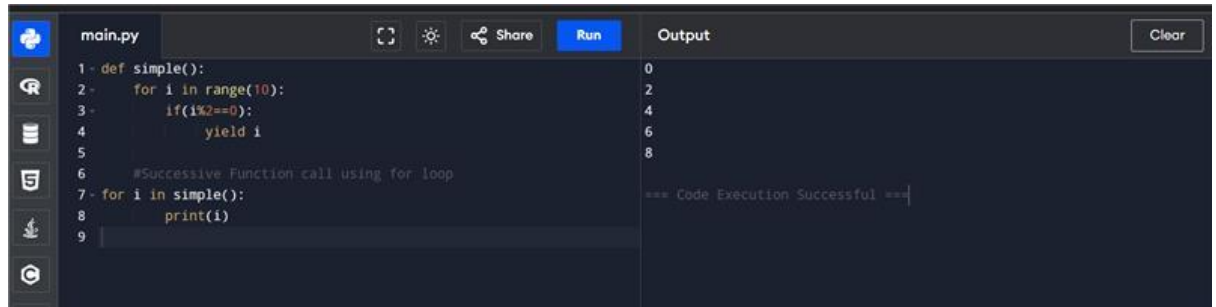
```
1 import functools # Importing functools into the program
2
3 class Count_Calls:
4     # Class to count the number of times a function is called
5     def __init__(self, func):
6         functools.update_wrapper(self, func) # To update the wrapper with the original
7         self.func = func # Store the original function
8         self.num_calls = 0 # Initialize call counter
9
10    def __call__(self, *args, **kwargs):
11        # Increment the call counter each time the function is called
12        self.num_calls += 1
13        print(f"Call {self.num_calls} of {self.func.__name__!r}")
14        return self.func(*args, **kwargs) # Call the original function
15
16 # Applying the Count_Calls class as a decorator
17 @Count_Calls
18 def say_hello():
19     print("Say Hello")
20
21 # Calling the decorated function multiple times
22 say_hello() # First call
23 say_hello() # Second call
24 say_hello() # Third call
25
```

input

```
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

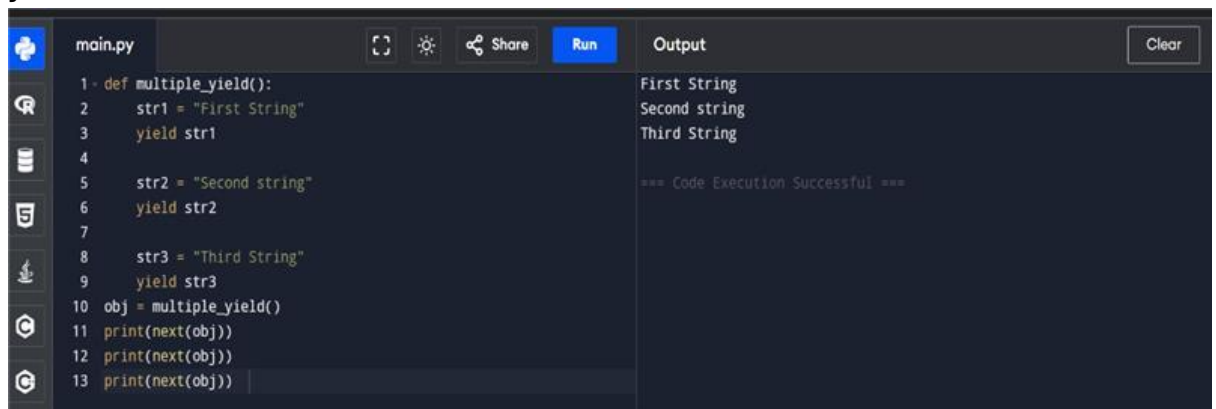


```
main.py
1 def simple():
2     for i in range(10):
3         if(i%2==0):
4             yield i
5
6     #Successive Function call using for loop
7 for i in simple():
8     print(i)
9
```

Output

```
0
2
4
6
8
=== Code Execution Successful ===
```

ii. yield vs return



```
main.py
1 def multiple_yield():
2     str1 = "First String"
3     yield str1
4
5     str2 = "Second string"
6     yield str2
7
8     str3 = "Third String"
9     yield str3
10 obj = multiple_yield()
11 print(next(obj))
12 print(next(obj))
13 print(next(obj))
```

Output

```
First String
Second string
Third String
=== Code Execution Successful ===
```

iii. Generator Expression

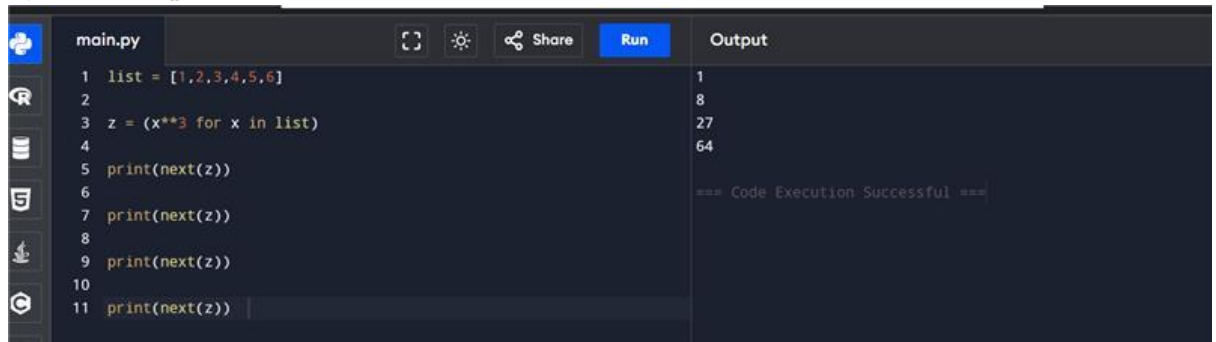


```
main.py
1 list = [1,2,3,4,5,6,7]
2
3 # List Comprehension
4 z = [x**3 for x in list]
5
6 # Generator expression
7 a = (x**3 for x in list)
8
9 print(a)
10 print(z)
```

Output

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

iv. Python next()

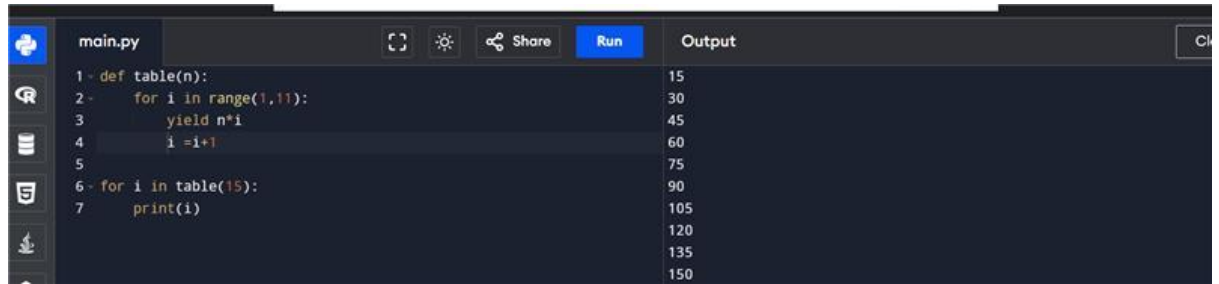


```
main.py
1 list = [1,2,3,4,5,6]
2
3 z = (x**3 for x in list)
4
5 print(next(z))
6
7 print(next(z))
8
9 print(next(z))
10
11 print(next(z))
```

Output

```
1
8
27
64
=== Code Execution Successful ===
```

v. Table program using generators

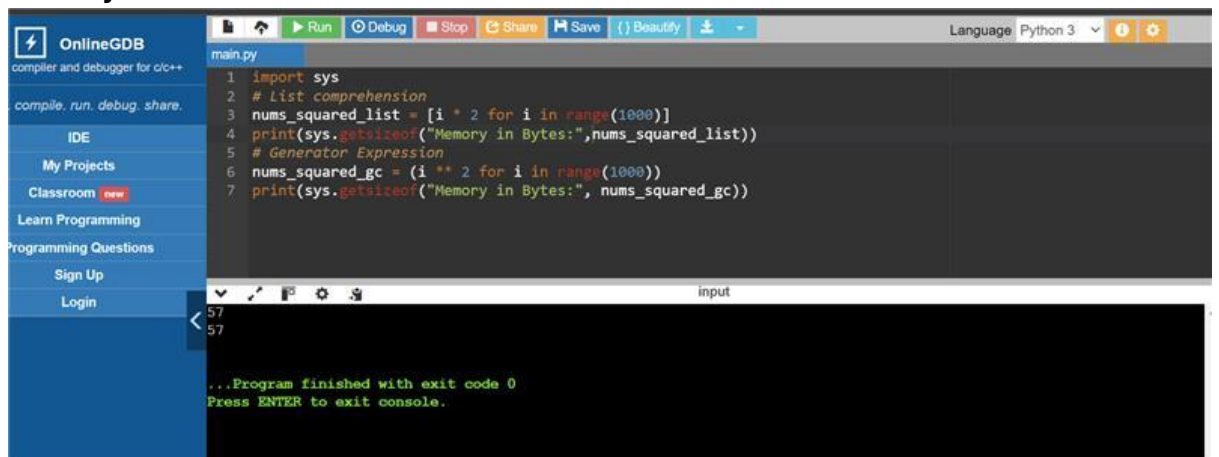


```
main.py
1 def table(n):
2     for i in range(1,11):
3         yield n*i
4         i = i+1
5
6 for i in table(15):
7     print(i)
```

Output

15
30
45
60
75
90
105
120
135
150

vi. Memory efficient



```
OnlineGDB
compiler and debugger for c/c++
compile, run, debug, share.
IDE
My Projects
Classroom new
Learn Programming
Programming Questions
Sign Up
Login
```

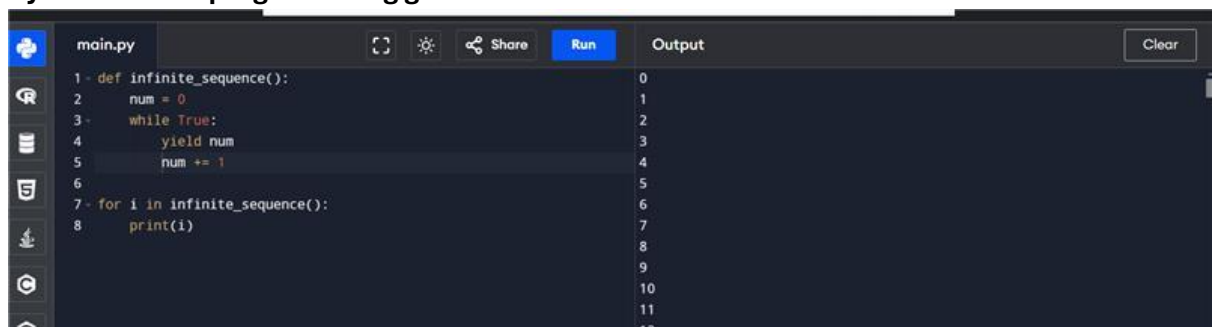
```
main.py
1 import sys
2 # List comprehension
3 nums_squared_list = [i * 2 for i in range(1000)]
4 print(sys.getsizeof("Memory in Bytes:", nums_squared_list))
5 # Generator Expression
6 nums_squared_gc = (i ** 2 for i in range(1000))
7 print(sys.getsizeof("Memory in Bytes:", nums_squared_gc))
```

input

57
57

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

vii. Python infinite program using generators



```
main.py
1 def infinite_sequence():
2     num = 0
3     while True:
4         yield num
5         num += 1
6
7 for i in infinite_sequence():
8     print(i)
```

Output

0
1
2
3
4
5
6
7
8
9
10
11
12