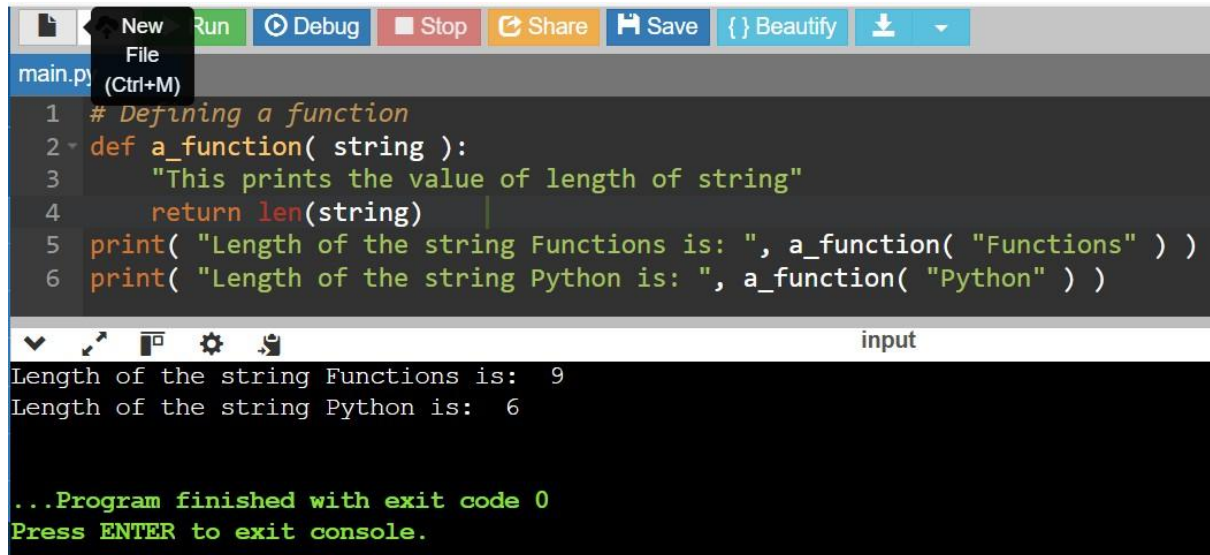


PYTHON FUNCTIONS

1. Calling a function



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. A dropdown menu is open under the 'New' button, showing 'File' and '(Ctrl+M)'. The main editor displays a Python script named 'main.py' with the following code:

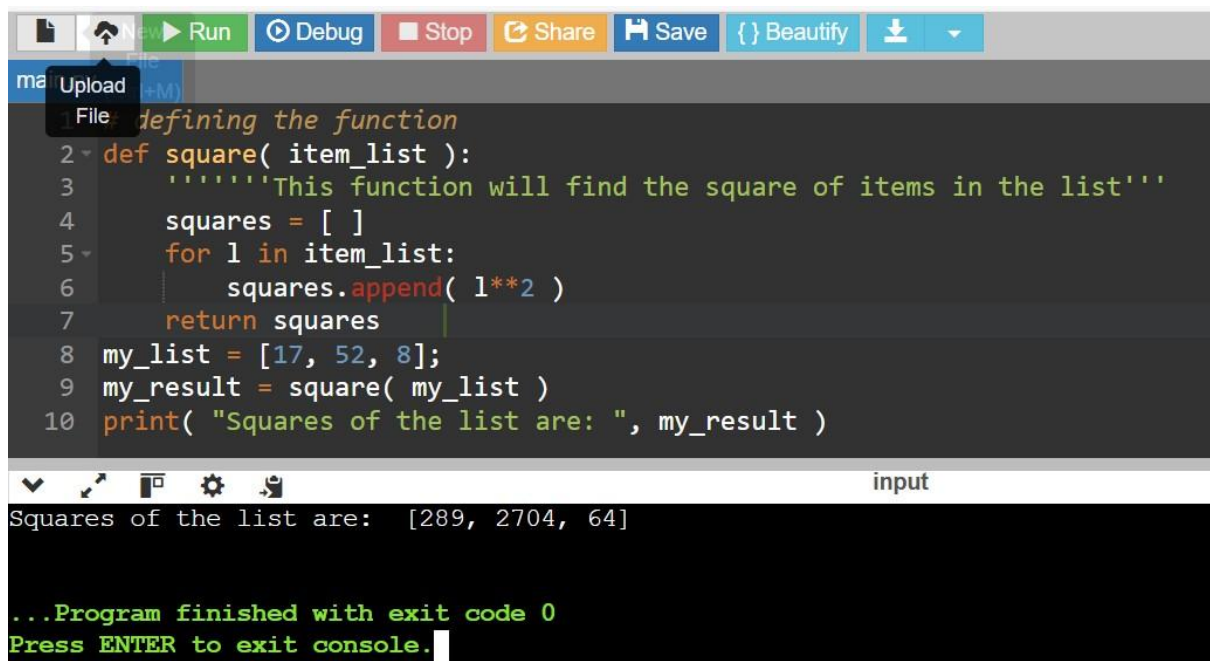
```
1 # Defining a function
2 def a_function( string ):
3     "This prints the value of length of string"
4     return len(string)
5 print( "Length of the string Functions is: ", a_function( "Functions" ) )
6 print( "Length of the string Python is: ", a_function( "Python" ) )
```

Below the editor, there is a console window with the output:

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

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

2. Pass by Reference Vs Pass by Value



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. A dropdown menu is open under the 'New' button, showing 'File' and '(Ctrl+M)'. The main editor displays a Python script named 'main.py' with the following code:

```
1 # defining the function
2 def square( item_list ):
3     '''This function will find the square of items in the list'''
4     squares = [ ]
5     for l in item_list:
6         squares.append( l**2 )
7     return squares
8 my_list = [17, 52, 8];
9 my_result = square( my_list )
10 print( "Squares of the list are: ", my_result )
```

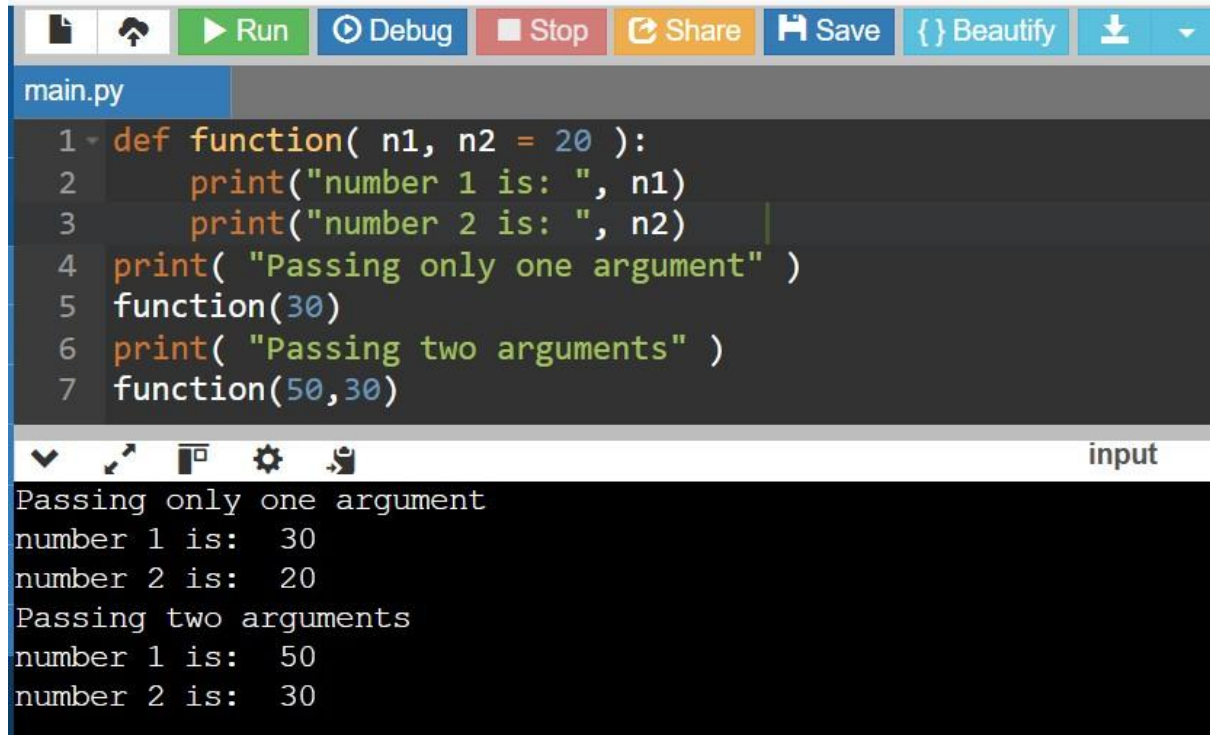
Below the editor, there is a console window with the output:

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

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

FUNCTION ARGUMENTS

1. Default arguments



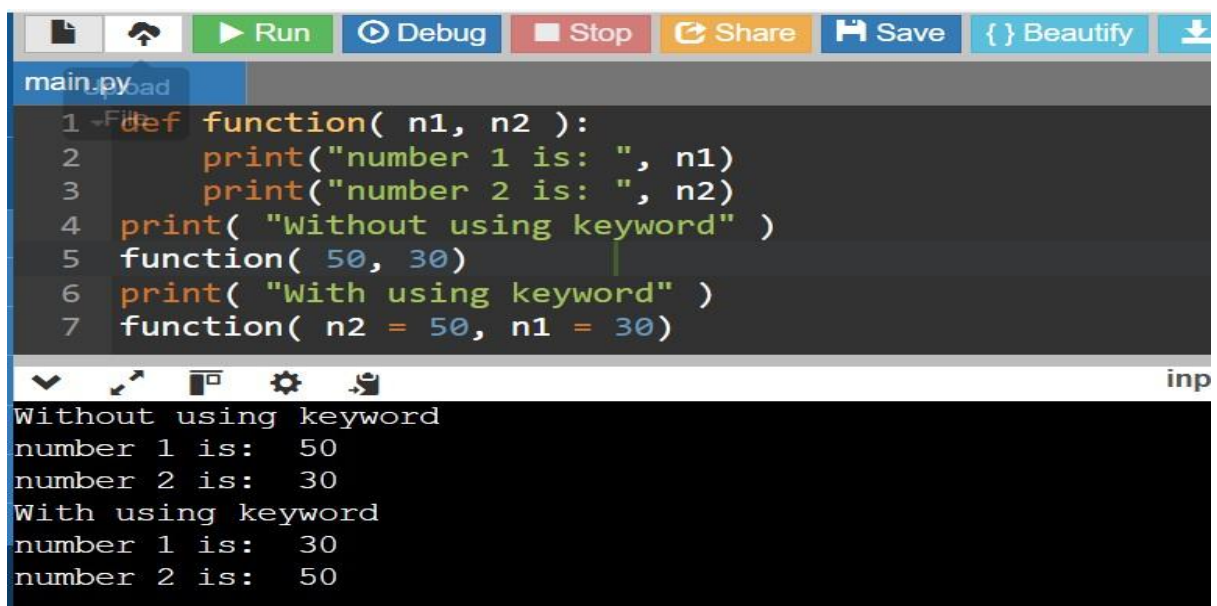
The screenshot shows a code editor with a toolbar at the top containing icons for file operations, a 'Run' button, 'Debug', 'Stop', 'Share', 'Save', 'Beautify', and a download icon. The file name 'main.py' is displayed. The code defines a function 'function' with two parameters, 'n1' and 'n2', where 'n2' has a default value of 20. The function prints the values of 'n1' and 'n2'. Below the code, the execution output is shown, demonstrating the function being called with one and two arguments.

```
1 def function( n1, n2 = 20 ):
2     print("number 1 is: ", n1)
3     print("number 2 is: ", n2)
4 print( "Passing only one argument" )
5 function(30)
6 print( "Passing two arguments" )
7 function(50,30)
```

input

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

2.Keyword arguments



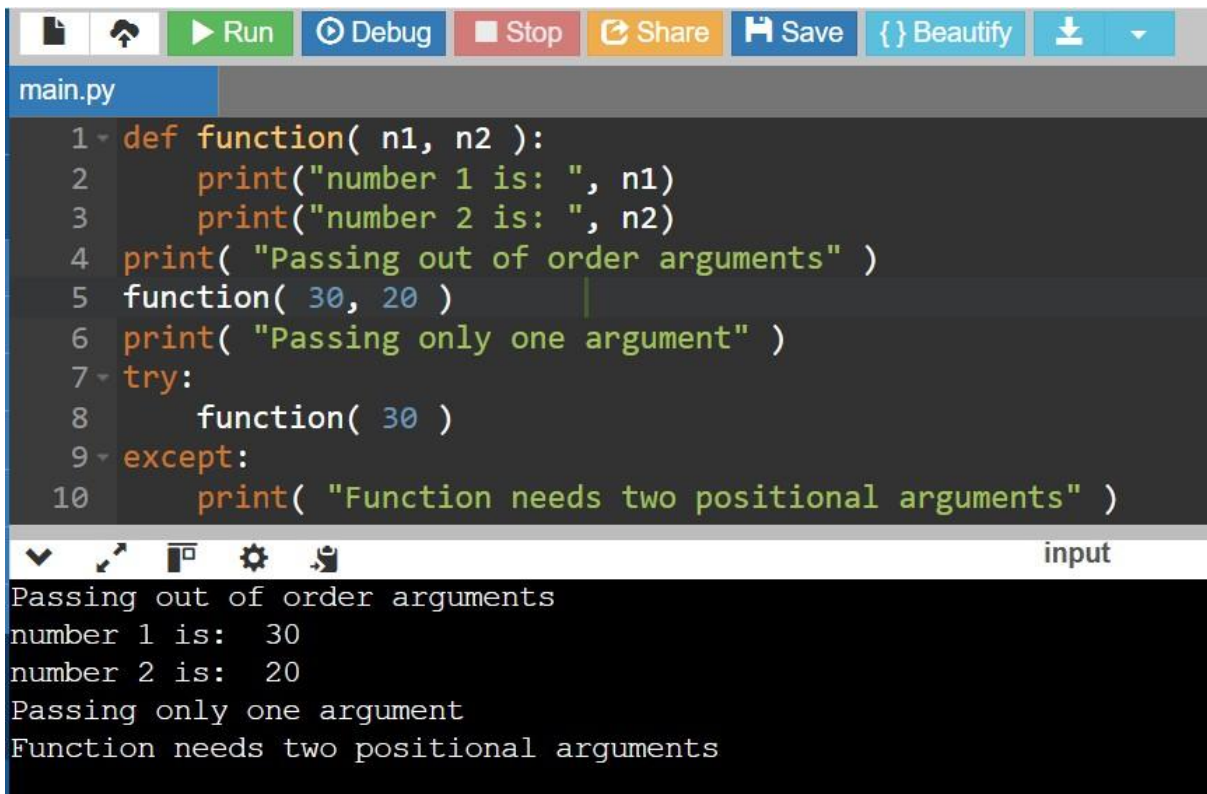
The screenshot shows a code editor with a toolbar at the top containing icons for file operations, a 'Run' button, 'Debug', 'Stop', 'Share', 'Save', 'Beautify', and a download icon. The file name 'main.py' is displayed. The code defines a function 'function' with two parameters, 'n1' and 'n2'. The function prints the values of 'n1' and 'n2'. Below the code, the execution output is shown, demonstrating the function being called with positional and keyword arguments.

```
1 def function( n1, n2 ):
2     print("number 1 is: ", n1)
3     print("number 2 is: ", n2)
4 print( "Without using keyword" )
5 function( 50, 30)
6 print( "With using keyword" )
7 function( n2 = 50, n1 = 30)
```

inp

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

3.Required arguments



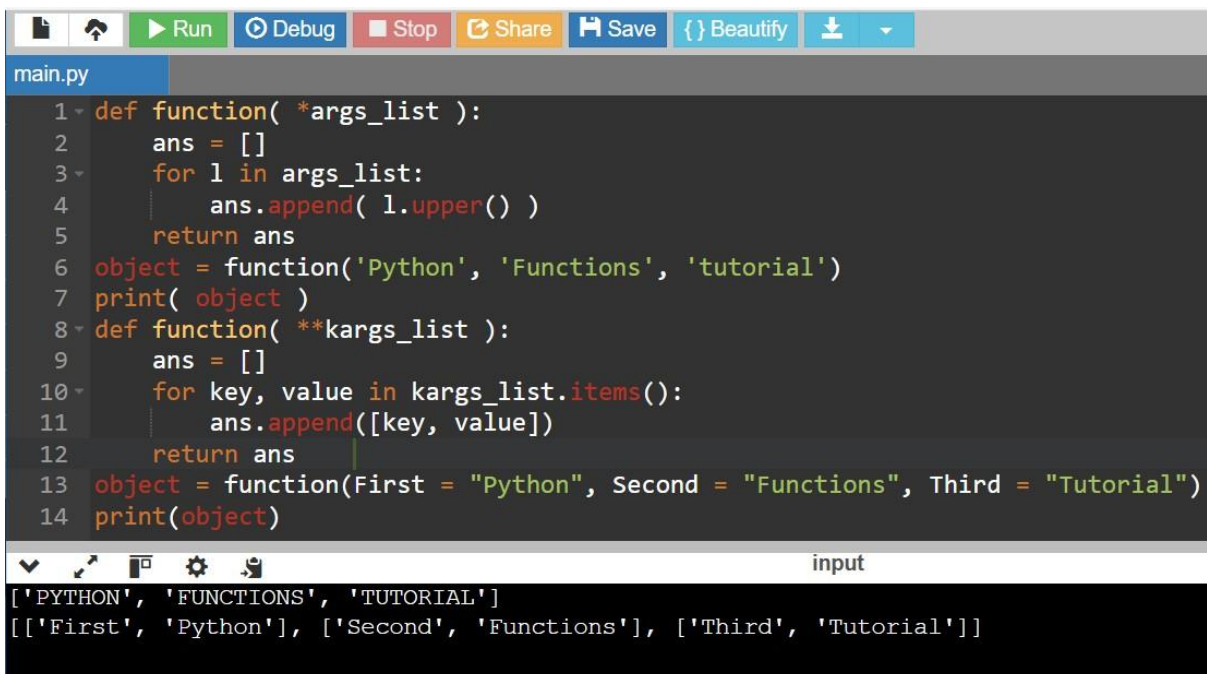
The screenshot shows a code editor with a toolbar at the top containing icons for file operations, a 'Run' button, 'Debug', 'Stop', 'Share', 'Save', 'Beautify', and a download icon. The file name 'main.py' is displayed. The code defines a function 'function' that takes two positional arguments, 'n1' and 'n2'. It prints the values of 'n1' and 'n2', then prints 'Passing out of order arguments'. It then calls 'function(30, 20)', prints 'Passing only one argument', and enters a 'try' block where it calls 'function(30)'. An 'except' block follows, printing 'Function needs two positional arguments'. The output window at the bottom shows the execution results: 'Passing out of order arguments', 'number 1 is: 30', 'number 2 is: 20', 'Passing only one argument', and 'Function needs two positional arguments'.

```
1 def function( n1, n2 ):
2     print("number 1 is: ", n1)
3     print("number 2 is: ", n2)
4     print( "Passing out of order arguments" )
5     function( 30, 20 )
6     print( "Passing only one argument" )
7     try:
8         function( 30 )
9     except:
10        print( "Function needs two positional arguments" )
```

input

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

4.Variable-length arguments



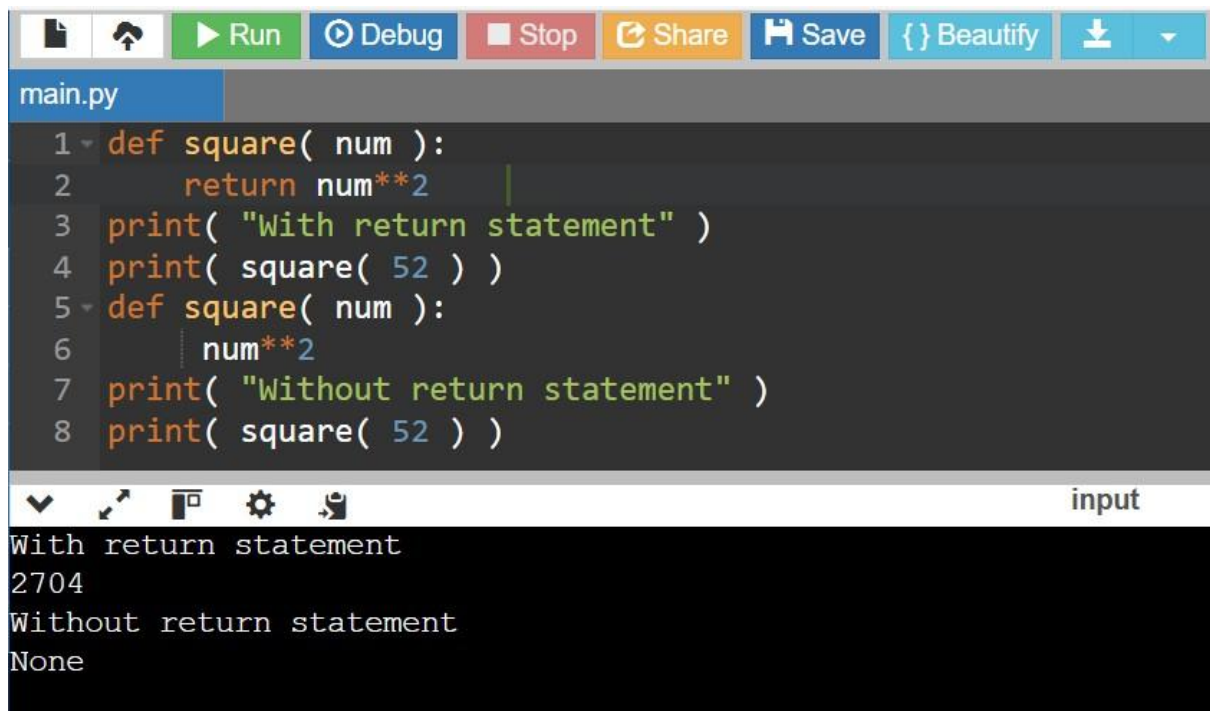
The screenshot shows a code editor with a toolbar at the top containing icons for file operations, a 'Run' button, 'Debug', 'Stop', 'Share', 'Save', 'Beautify', and a download icon. The file name 'main.py' is displayed. The code defines two functions. The first function, 'function', takes a variable-length argument '*args_list' and iterates over it to append the upper case of each element to a list 'ans', which is then returned. It calls this function with 'Python', 'Functions', and 'tutorial'. The second function, 'function', takes a variable-length keyword argument '**kwargs_list' and iterates over its items to append each key-value pair to a list 'ans', which is then returned. It calls this function with 'First = "Python"', 'Second = "Functions"', and 'Third = "Tutorial"'. The output window at the bottom shows the execution results: ['PYTHON', 'FUNCTIONS', 'TUTORIAL'] and [['First', 'Python'], ['Second', 'Functions'], ['Third', 'Tutorial']].

```
1 def function( *args_list ):
2     ans = []
3     for l in args_list:
4         ans.append( l.upper() )
5     return ans
6 object = function('Python', 'Functions', 'tutorial')
7 print( object )
8 def function( **kwargs_list ):
9     ans = []
10    for key, value in kwargs_list.items():
11        ans.append([key, value])
12    return ans
13 object = function(First = "Python", Second = "Functions", Third = "Tutorial")
14 print(object)
```

input

['PYTHON', 'FUNCTIONS', 'TUTORIAL']
[['First', 'Python'], ['Second', 'Functions'], ['Third', 'Tutorial']]

RETURN STATEMENT



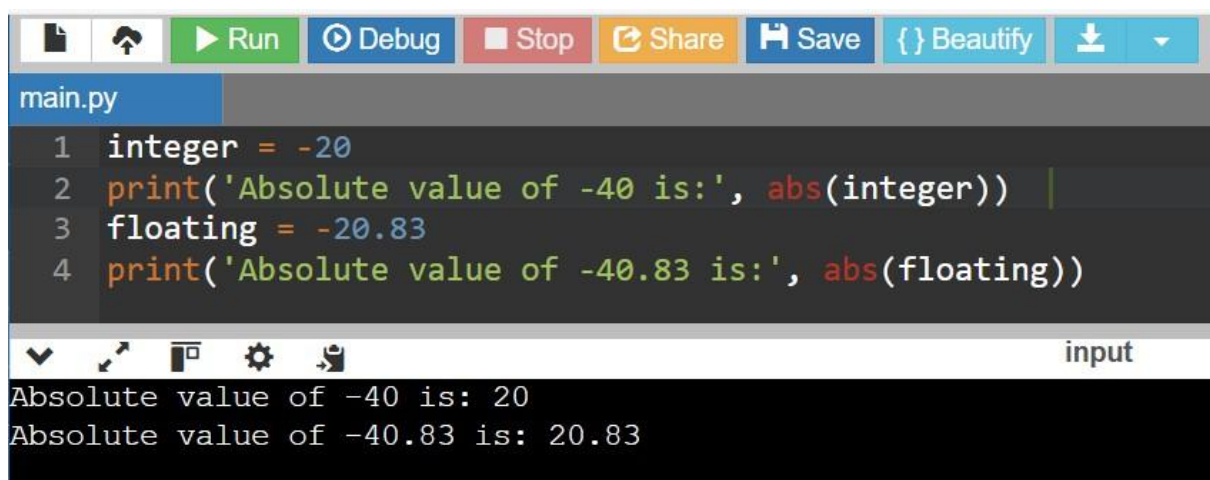
```
main.py
1 def square( num ):
2     return num**2
3 print( "With return statement" )
4 print( square( 52 ) )
5 def square( num ):
6     num**2
7 print( "Without return statement" )
8 print( square( 52 ) )
```

input

With return statement
2704
Without return statement
None

PYTHON BUILT-IN FUNCTIONS

1. Abs () function

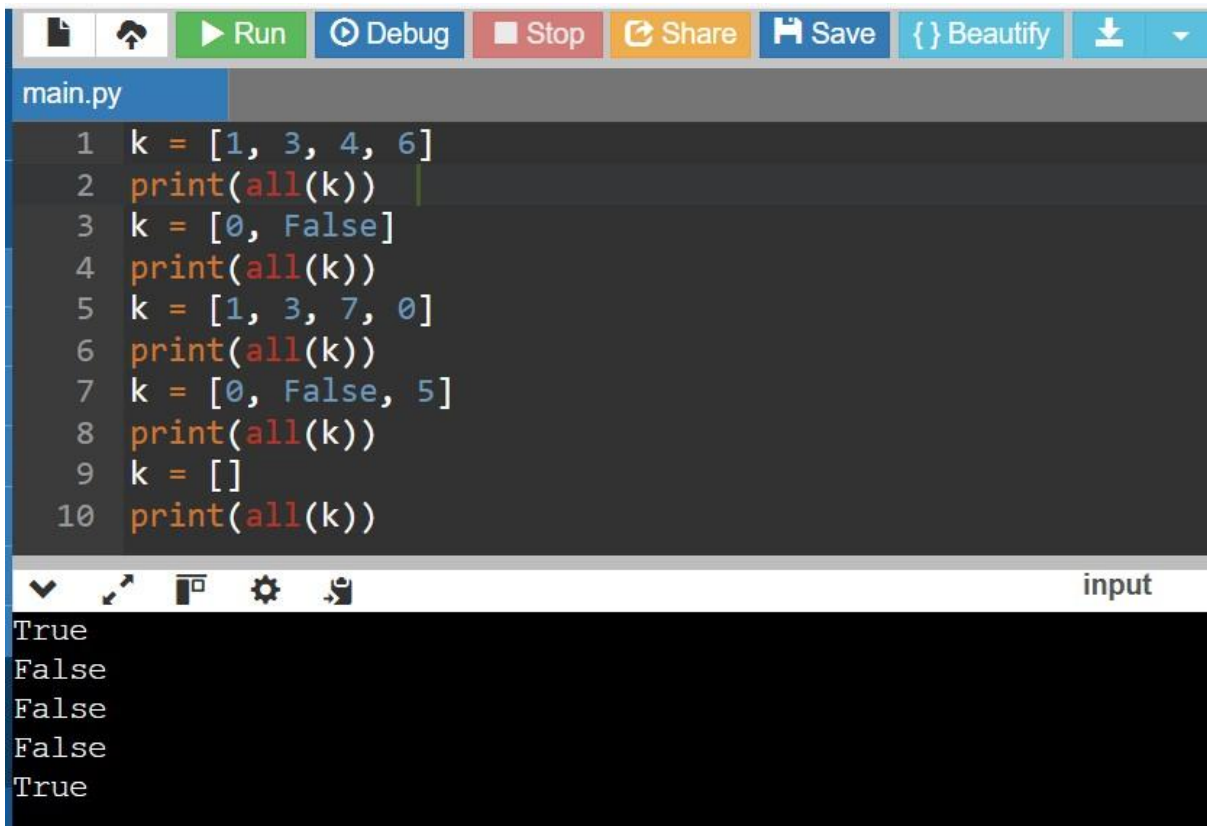


```
main.py
1 integer = -20
2 print('Absolute value of -40 is:', abs(integer))
3 floating = -20.83
4 print('Absolute value of -40.83 is:', abs(floating))
```

input

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

2. All () function



```
main.py
1 k = [1, 3, 4, 6]
2 print(all(k))
3 k = [0, False]
4 print(all(k))
5 k = [1, 3, 7, 0]
6 print(all(k))
7 k = [0, False, 5]
8 print(all(k))
9 k = []
10 print(all(k))
```

True
False
False
False
True


3.Bool () function



```
main.py
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

4.Sum () Function

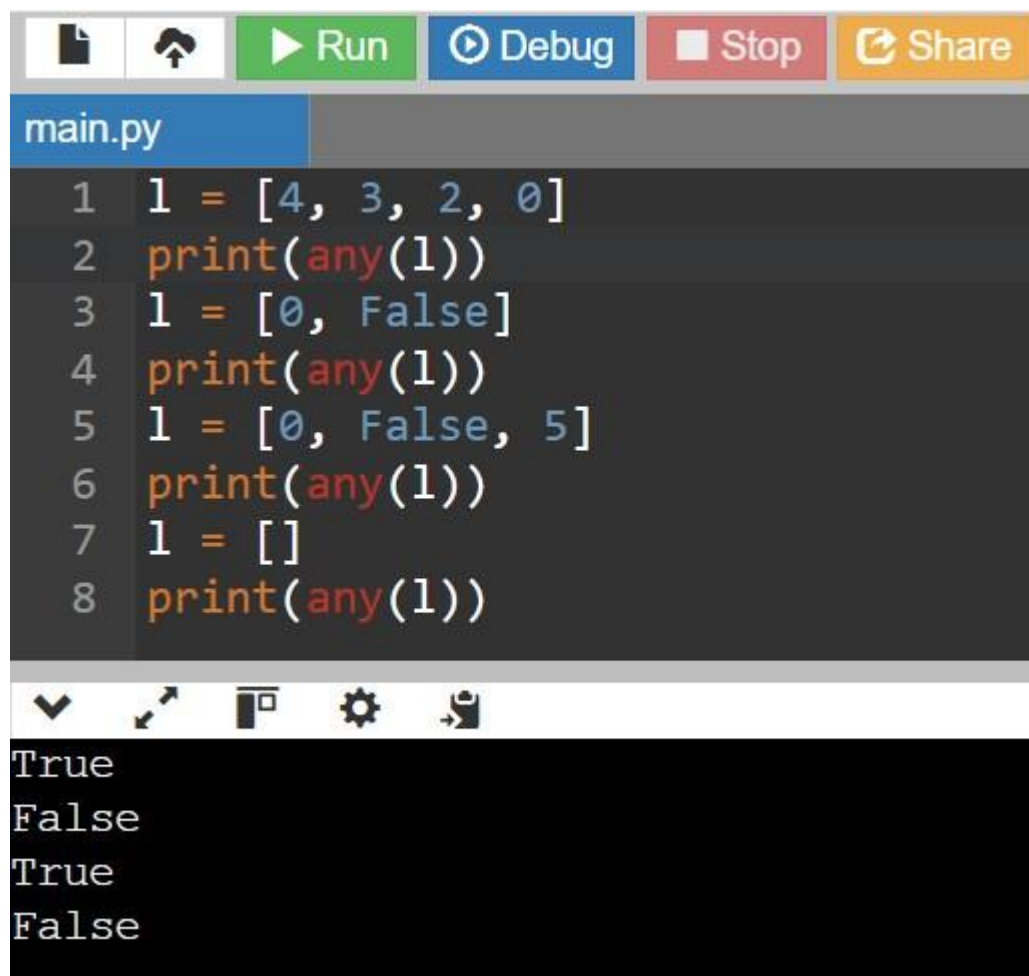


The screenshot shows a code editor with a toolbar at the top containing icons for file operations, a 'Run' button, 'Debug', 'Stop', 'Share', 'Save', 'Beautify', and a download icon. The file name 'main.py' is displayed. The code in the editor is as follows:

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

Below the code editor, the output is shown in a black box with the text 'input' on the right. The output consists of two lines: '7' and '17'.

5.Any () function



The screenshot shows a code editor with a toolbar at the top containing icons for file operations, a 'Run' button, 'Debug', 'Stop', and 'Share'. The file name 'main.py' is displayed. The code in the editor is as follows:

```
1 l = [4, 3, 2, 0]
2 print(any(l))
3 l = [0, False]
4 print(any(l))
5 l = [0, False, 5]
6 print(any(l))
7 l = []
8 print(any(l))
```

Below the code editor, the output is shown in a black box. The output consists of four lines: 'True', 'False', 'True', and 'False'.

PYTHON LAMBDA FUNCTION

1. Lambda function example

```
main.py
1 add = lambda num: num + 4
2 print( add(6) )
```

10

2. Distinction between Lambda and Def Function

```
main.py
1 def reciprocal( num ):
2     return 1 / num
3 lambda_reciprocal = lambda num: 1 / num
4 print( "Def keyword: ", reciprocal(6) )
5 print( "Lambda keyword: ", lambda_reciprocal(6) )
```

input

Def keyword: 0.16666666666666666
Lambda keyword: 0.16666666666666666

3. Using Lambda Function with map ()

```
main.py
1 numbers_list = [2, 4, 5, 1, 3, 7, 8, 9, 10]
2 squared_list = list(map( lambda num: num ** 2 , numbers_list ))
3 print( 'Square of each number in the given list:' ,squared_list )
```

input

Square of each number in the given list: [4, 16, 25, 1, 9, 49, 64, 81, 100]

4. Using Lambda Function with List

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

input
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
```

5. Using Lambda Function with Multiple Statements

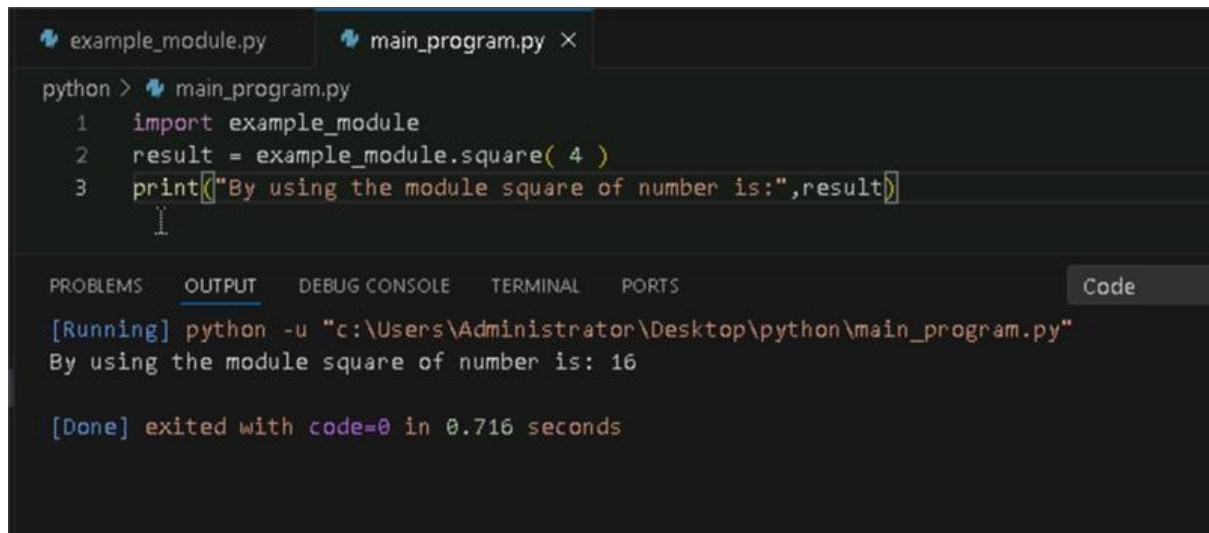
```
main.py
1 my_List = [ [3, 5, 8, 6], [23, 54, 12, 87], [1, 2, 4, 12, 5] ]
2 sort_List = lambda num : ( sorted(n) for n in num )
3 third_Largest = lambda num, func : [ l[ len(l) - 2] for l in func(num)]
4 result = third_Largest( my_List, sort_List)
5 print('The third largest number from every sub list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

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

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


MODULES

1. Python Modules



The screenshot shows an IDE with two tabs: `example_module.py` and `main_program.py`. The `main_program.py` tab is active, showing the following code:

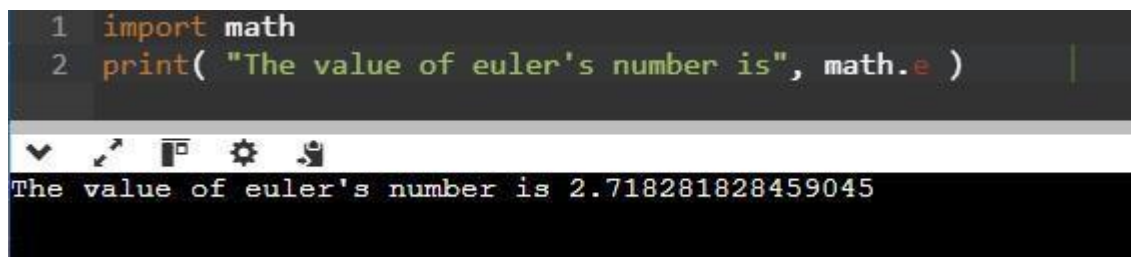
```
python > main_program.py
1 import example_module
2 result = example_module.square( 4 )
3 print("By using the module square of number is:",result)
```

Below the code editor, the **OUTPUT** pane shows the execution results:

```
[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. Importing and also Renaming



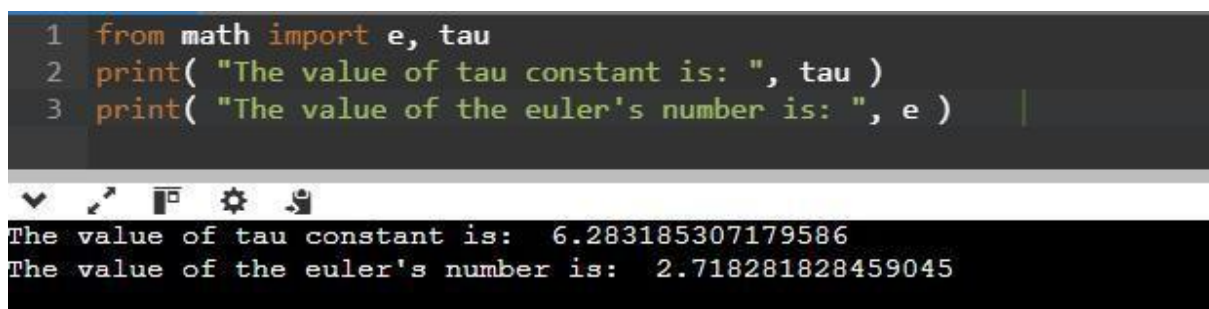
The screenshot shows a Python script with the following code:

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

The output of the script is:

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

3. Python from...import Statement



The screenshot shows a Python script with the following code:

```
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 output of the script is:

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

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
```

input

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

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)
```

input

Path of the sys module in the system is: ['/home', '/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']

6.The dir() Built-in Function

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

input

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_', '_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_', '_isnumeric_', '_isprintable_', '_isspace_', '_istitle_', '_isupper_', '_join_', '_ljust_', '_lower_', '_lstrip_', '_maketrans_', '_partition_', '_removeprefix_', '_removesuffix_', '_replace_', '_find_', '_rindex_', '_rjust_', '_rpartition_', '_rsplit_', '_rstrip_', '_split_', '_splitlines_', '_startswith_', '_strip_', '_swapcase_', '_title_', '_translate_', '_upper_', '_zfill_']

7.Namespaces and Scoping

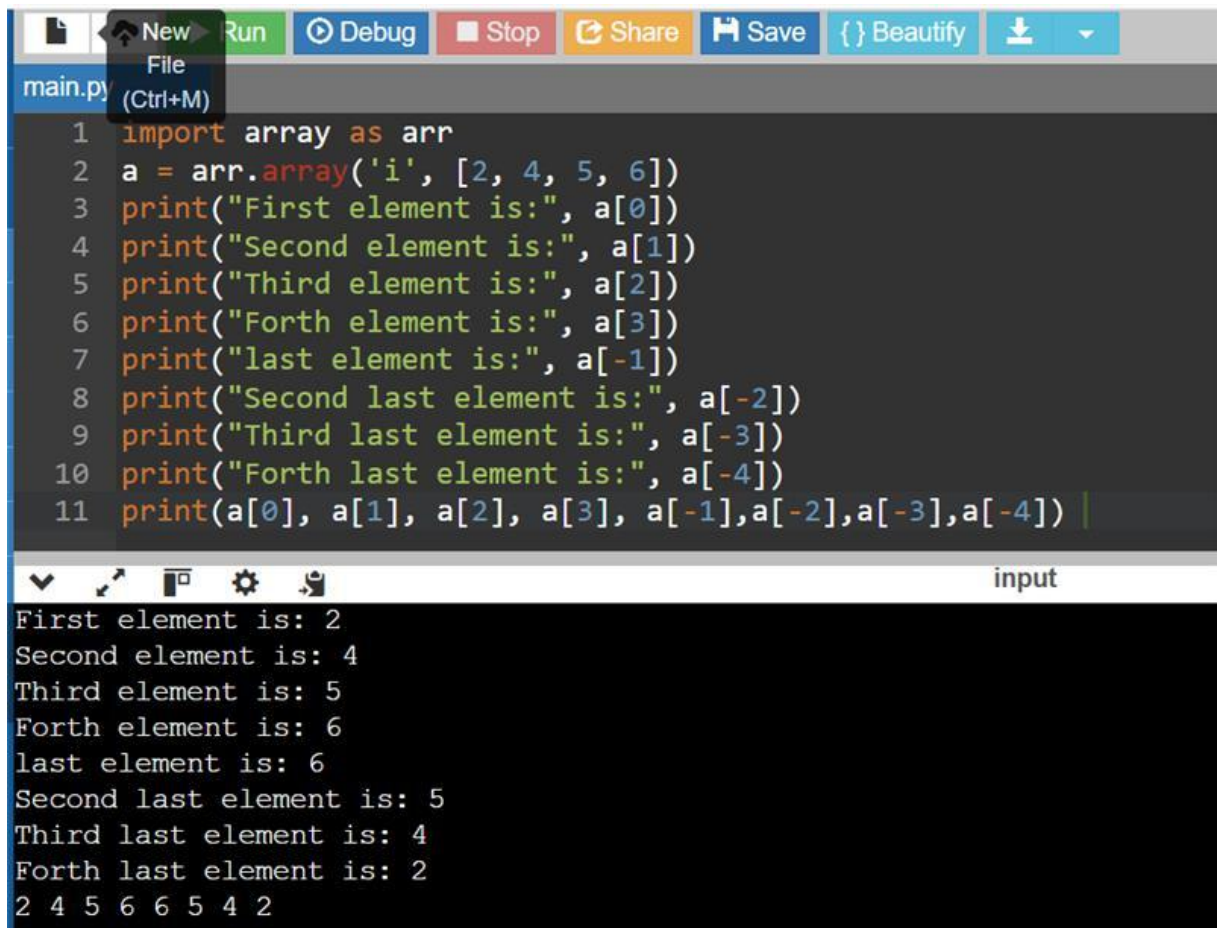
```
1 Number = 204
2 def AddNumber(): # here, we are defining a function with the name Add Number
3     # Here, we are accessing the global namespace
4     global Number
5     Number = Number + 200
6 print("The number is:", Number)
7 # here, we are printing the number after performing the addition
8 AddNumber() # here, we are calling the function
9 print("The number is:", Number)
10
```

input

The number is: 204
The number is: 404

PYTHON ARRAYS

1. Accessing array elements



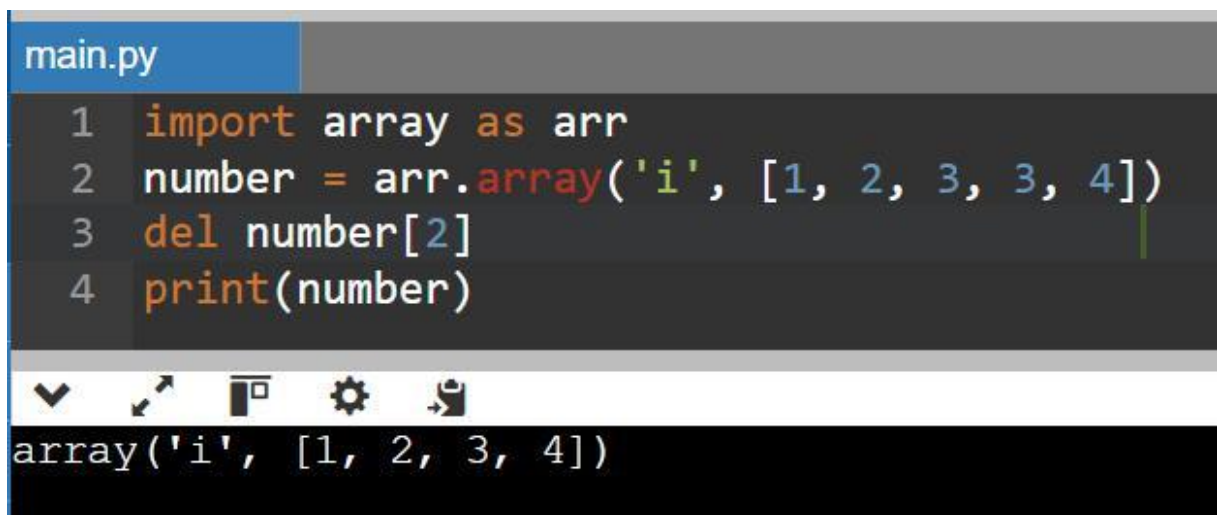
The screenshot shows a Python IDE with a toolbar at the top containing buttons for New, Run, Debug, Stop, Share, Save, Beautify, and a download icon. The file name 'main.py' is visible. The code in the editor is as follows:

```
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 in the console is:

```
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
```

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 in the console is:

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

3. Adding or changing the elements in Array

```
main.py File (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.
```

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
```


PYTHON DECORATOR

```
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

1.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

```
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


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

▼ ↗ 📄 ⚙️ 🐞

Hello

2. 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)
```

▼ ↗ 📄 ⚙️ 🐞

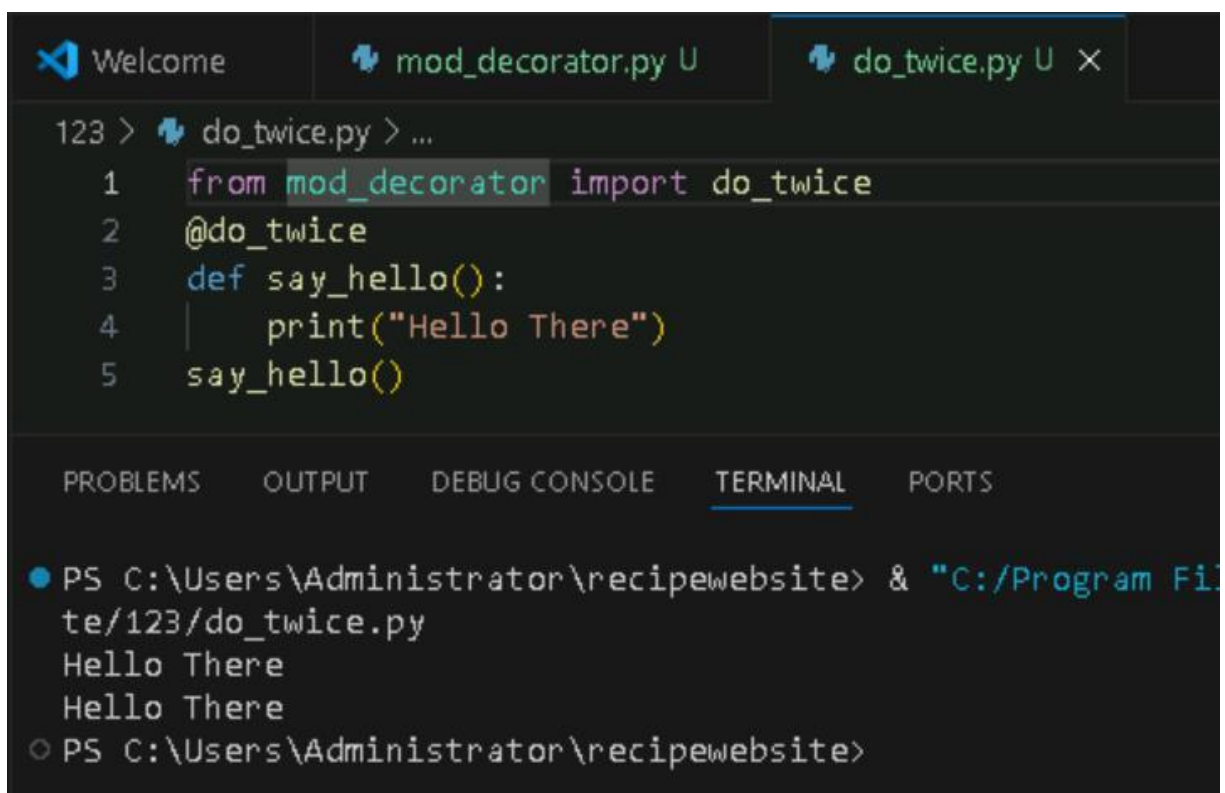
Hello

3.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.Reusing Decorator



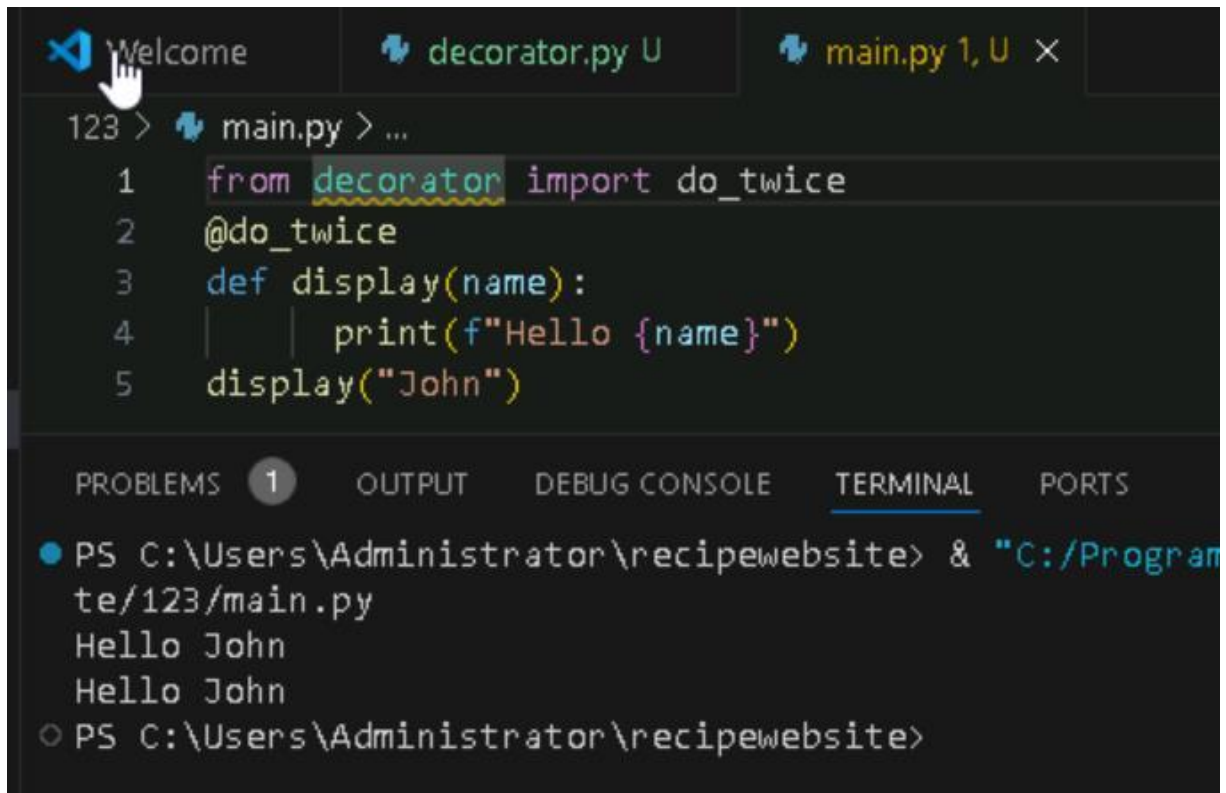
The screenshot shows a Visual Studio Code editor with two open files: `mod_decorator.py` and `do_twice.py`. The `do_twice.py` file contains the following code:

```
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** panel shows the command prompt output:

```
PS C:\Users\Administrator\recipewebsite> & "C:/Program Files/Python311/Python.exe" C:/Users/Administrator/recipewebsite/123/do_twice.py
Hello There
Hello There
PS C:\Users\Administrator\recipewebsite>
```

5. Python Decorator with Argument



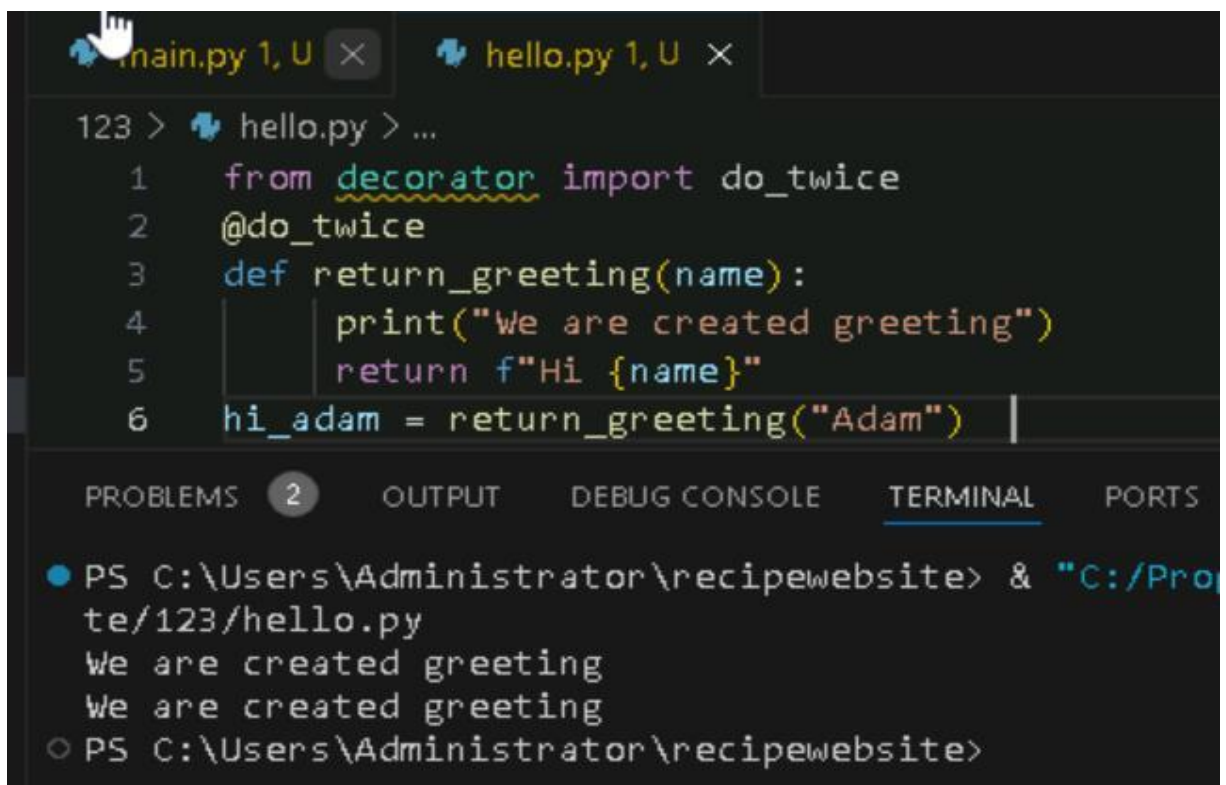
The screenshot shows the Visual Studio Code editor with two tabs: 'Welcome' and 'decorator.py U'. A third tab, 'main.py 1, U', is active. The code in 'main.py' is as follows:

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

The bottom panel shows the 'TERMINAL' tab with the following output:

```
PS C:\Users\Administrator\recipewebsite> & "C:/Program Files/Python311/Python.exe" C:/Program Files/Python311/Scripts/python.exe C:/Users/Administrator/recipewebsite/123/main.py
Hello John
Hello John
PS C:\Users\Administrator\recipewebsite>
```

6. Returning Values from Decorated Functions



The screenshot shows the Visual Studio Code editor with two tabs: 'main.py 1, U' and 'hello.py 1, U'. The code in 'hello.py' is as follows:

```
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")
```

The bottom panel shows the 'TERMINAL' tab with the following output:

```
PS C:\Users\Administrator\recipewebsite> & "C:/Program Files/Python311/Python.exe" C:/Program Files/Python311/Scripts/python.exe C:/Users/Administrator/recipewebsite/123/hello.py
We are created greeting
We are created greeting
PS C:\Users\Administrator\recipewebsite>
```

7.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

8.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 metadata
7         def wrapper(*args, **kwargs):
8             for _ in range(num): # Looping 'num' times to repeat the function
9                 value = func(*args, **kwargs) # Calling the original function
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

9.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__!r}")
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

10.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
```