

Python Functions:

Illustration of a User-Defined Function

main.py Output

```
1 def square( num ):  
2     """  
3     This function computes the square of the number.  
4     """  
5     return num**2  
6 object_ = square(6)  
7 print( "The square of the given number is: ", object_ )
```

main.py Output

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

Calling a Function

main.py Output

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




main.py Output

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

Pass by Reference vs. Pass by Value

main.py

Output



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

main.py

Output




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

Function Arguments

1. Default arguments

main.py

Output



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

main.py




Output

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

main.py

Output



```
1 def function( n1, n2 ):
2     print("number 1 is: ", n1)
3     print("number 2 is: ", n2)
4
5     # Calling function and passing arguments without using keyword
6     print( "Without using keyword" )
7     function( 50, 30)
8
9     # Calling function and passing arguments using keyword
10    print( "With using keyword" )
11    function( n2 = 50, n1 = 30)
```

main.py




Output

```
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

main.py

Output



```
1 def function( n1, n2 ):
2     print("number 1 is: ", n1)
3     print("number 2 is: ", n2)
4
5     # Calling function and passing two arguments out of order, we need
6     # num1 to be 20 and num2 to be 30
7     print( "Passing out of order arguments" )
8     function( 30, 20 )
9
10    # Calling function and passing only one argument
11    print( "Passing only one argument" )
12    try:
13        function( 30 )
14    except:
15        print( "Function needs two positional arguments" )
```

main.py




Output

```
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

main.py




Output



```
1 def function( *args_list ):
2     ans = []
3     for l in args_list:
4         ans.append( l.upper() )
5     return ans
6 # Passing args arguments
7 object = function('Python', 'Functions', 'tutorial')
8 print( object )
9
10 # defining a function
11 def function( **kargs_list ):
12     ans = []
13     for key, value in kargs_list.items():
14         ans.append([key, value])
15     return ans
16 # Paasing kwargs arguments
17 object = function(First = "Python", Second = "Functions", Third =
    "Tutorial")
18 print(object)
```

main.py

Output





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

return Statement

main.py

Output



```
1 def square( num ):
2     return num**2
3
4 # Calling function and passing arguments.
5 print( "With return statement" )
6 print( square( 52 ) )
7
8 # Defining a function without return statement
9 def square( num ):
10     num**2
11
12 # Calling function and passing arguments.
13 print( "Without return statement" )
14 print( square( 52 ) )
```

main.py

Output

```
With return statement
2704
Without return statement
None
```

The Anonymous Functions

```
main.py Output
1 lambda_ = lambda argument1, argument2: argument1 + argument2;
2
3 # Calling the function and passing values
4 print( "Value of the function is : ", lambda_( 20, 30 ) )
5 print( "Value of the function is : ", lambda_( 40, 50 ) )
```

```
main.py Output
Value of the function is : 50
Value of the function is : 90
```

Scope and Lifetime of Variables

```
main.py Output
1 def number( ):
2     num = 50
3     print( "Value of num inside the function: ", num)
4 num = 10
5 number()
6 print( "Value of num outside the function:", num)
```

```
main.py Output
Value of num inside the function: 50
Value of num outside the function: 10
```

Python Capability inside Another Capability

```
main.py Output
1 def word():
2     string = 'Python functions tutorial'
3     x = 5
4     def number():
5         print( string )
6         print( x )
7     number()
8 word()
```

main.py

Output

Python functions tutorial
5

Python abs() Function

main.py

Output

```
1 # integer number
2 integer = -20
3 print('Absolute value of -40 is:', abs(integer))
4
5 # floating number
6 floating = -20.83
7 print('Absolute value of -40.83 is:', abs(floating))
```

main.py

Output

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

Python all() Function

main.py

Output



```
1 # all values true
2 k = [1, 3, 4, 6]
3 print(all(k))
4
5 # all values false
6 k = [0, False]
7 print(all(k))
8
9 # one false value
10 k = [1, 3, 7, 0]
11 print(all(k))
12
13 # one true value
14 k = [0, False, 5]
15 print(all(k))
16
17 # empty iterable
18 k = []
19 print(all(k))
```

main.py

Output



True
False
False
False
True

Python bin() Function

main.py

Output

```
1 x = 10
2 y = bin(x)
3 print(y)
```

main.py




Output

0b1010

Python bool()

main.py




Output



```
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))
```

main.py

Output



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

Python bytes()

main.py

Output



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

main.py

Output

b'Hello World.'

Python callable() Function

main.py	Run	Output
<pre>1 x=8 2 print(callable(x))</pre>		<pre>False === Code Execution Successful ===</pre>

Python compile() Function

main.py	Share	Run
<pre>1 #compile string source to code 2 code_str = 'x=5\ny=10\nprint("sum=",x+y)' 3 code = compile(code_str, 'sum.py', 'exec') 4 print(type(code)) 5 exec(code) 6 exec(x)</pre>		

Output
<pre><class 'code'> sum= 15 ERROR!</pre>

Python exec() Function

main.py	Run	Output
<pre>1 x=8 2 exec('print(x==8)') 3 exec('print(x+4)')</pre>		<pre>True 12</pre>

Python sum() Function

main.py	Run	Output
<pre>1 s=sum([1,2,4]) 2 print(s) 3 s=sum([1,2,4],10) 4 print(s)</pre>		<pre>7 17 === Code Execution Successful ===</pre>

Python any() Function

main.py		Output
<pre>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))</pre>	<div>☐ ☾ 🔗 Share Run</div>	<pre>True False True False === Code Execution Successful ===</pre>

Python ascii() Function

main.py		Output
<pre>1 normalText = 'Python is interesting' 2 print(ascii(normalText)) 3 otherText = 'Pythøn is interesting' 4 print(ascii(otherText)) 5 print('Pyth\xuf6n is interesting')</pre>	<div>☐ ☾ 🔗 Run</div>	<pre>'Python is interesting' 'Pyth\ufffdn is interesting' Pythön is interesting === Code Execution Successful ===</pre>

Python bytearray()

main.py		Output
<pre>1 string = "Python is a programming language." # string with encoding 'utf-8' 2 arr = bytearray(string, 'utf-8') 3 print(arr)</pre>	<div>☐ ☾ 🔗 Run</div>	<pre>bytearray(b'Python is a programming language.') === Code Execution Successful ===</pre>

Python eval() Function

main.py		Output
<pre>1 x = 8 2 print(eval('x + 1'))</pre>	<div>☐ ☾ 🔗 Run</div>	<pre>9 === Code Execution Successful ===</pre>

Python float()

main.py		Output
<pre>1 # for integers 2 print(float(9)) 3 # for floats 4 print(float(8.19)) 5 # for string floats 6 print(float("-24.27")) 7 # for string floats with whitespaces 8 print(float(" -17.19\n")) 9 # string float error 10 print(float("xyz"))</pre>	<div>☐ ☾ 🔗 Run</div>	

Output
9.0
8.19
-24.27
-17.19
ERROR!

Python format() Function

main.py	Output
<pre> 1 # d, f and b are a type 2 # integer 3 print(format(123, "d")) 4 # float arguments 5 print(format(123.4567898, "f")) 6 # binary format 7 print(format(12, "b")) </pre>	<pre> 123 123.456790 1100 === Code Execution Successful === </pre>

Python frozenset()

main.py	Output
<pre> 1 # tuple of letters 2 letters = ('m', 'r', 'o', 't', 's') 3 fSet = frozenset(letters) 4 print('Frozen set is:', fSet) 5 print('Empty frozen set is:', frozenset()) </pre>	<pre> Frozen set is: frozenset({'m', 't', 'o', 's', 'r'}) Empty frozen set is: frozenset() === Code Execution Successful === </pre>

Python getattr() Function

main.py	Output
<pre> 1 class Details: 2 age = 22 3 name = "Phill" 4 details = Details() 5 print('The age is:', getattr(details, "age")) 6 print('The age is:', details.age) </pre>	<pre> The age is: 22 The age is: 22 === Code Execution Successful === </pre>

Python globals() Function

main.py	Output
<pre> 1 age = 22 2 globals()['age'] = 22 3 print('The age is:', age) </pre>	<pre> The age is: 22 === Code Execution Successful === </pre>

Python hasattr() Function

main.py	Output
<pre>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))</pre>	<pre>True False True False === Code Execution Successful ===</pre>

Python iter() Function

main.py	Output
<pre>1 # list of numbers 2 list = [1,2,3,4,5] 3 listIter = iter(list) 4 # prints '1' 5 print(next(listIter)) 6 # prints '2' 7 print(next(listIter)) 8 # prints '3' 9 print(next(listIter)) 10 # prints '4' 11 print(next(listIter)) 12 # prints '5' 13 print(next(listIter))</pre>	<pre>1 2 3 4 5 === Code Execution Successful ===</pre>

Python len() Function

main.py	Output
<pre>1 strA = 'Python' 2 print(len(strA))</pre>	<pre>6 === Code Execution Successful ===</pre>

Python list()

main.py	Output
<pre>1 # empty list 2 print(list()) 3 # string 4 String = 'abcde' 5 print(list(String)) 6 # tuple 7 Tuple = (1,2,3,4,5) 8 print(list(Tuple)) 9 # list 10 List = [1,2,3,4,5] 11 print(list(List))</pre>	<pre>[] ['a', 'b', 'c', 'd', 'e'] [1, 2, 3, 4, 5] [1, 2, 3, 4, 5] === Code Execution Successful ===</pre>

Python locals() Function

main.py	Output
<pre>1 def localsAbsent(): 2 return locals() 3 def localsPresent(): 4 present = True 5 return locals() 6 print('localsNotPresent:', localsAbsent()) 7 print('localsPresent:', localsPresent())</pre>	<pre>localsNotPresent: {} localsPresent: {'present': True} === Code Execution Successful ===</pre>

Python map() Function

main.py	Output
<pre>1 def calculateAddition(n): 2 return n+n 3 numbers = (1, 2, 3, 4) 4 result = map(calculateAddition, numbers) 5 print(result) 6 # converting map object to set 7 numbersAddition = set(result) 8 print(numbersAddition)</pre>	<pre><map object at 0x7f9c03f3d450> {8, 2, 4, 6} === Code Execution Successful ===</pre>




Python memoryview() Function

main.py	Output
<pre>1 # A random bytearray 2 randomByteArray = bytearray('ABC', 'utf-8') 3 4 # Create a memory view 5 mv = memoryview(randomByteArray) 6 7 # Access the memory view's zeroth index 8 print(mv[0]) # Output: 65 (ASCII code for 'A') 9 10 # Create bytes from memory view 11 print(bytes(mv[0:2])) # Output: b'AB' 12 13 # Create list from memory view 14 print(list(mv[0:3])) # Output: [65, 66, 67]</pre>	<pre>65 b'AB' [65, 66, 67] === Code Execution Successful ===</pre>




Python object()

main.py	Output
<pre>1 python = object() 2 print(type(python)) 3 print(dir(python)) 4</pre>	<pre><class 'object'> ['_class__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__getattribute__', '__getstate__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__'] === Code Execution Successful ===</pre>





Python chr() Function

main.py	  	Run	Output
<pre>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)</pre>			<pre>f p is it string type: True === Code Execution Successful ===</pre>




Python complex()

main.py	  	Run	Output
<pre>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)</pre>			<pre>(1+0j) (1+2j) === Code Execution Successful ===</pre>





Python dir() Function

main.py	  	Run	Output	
<pre>1 # Calling function 2 att = dir() 3 # Displaying result 4 print(att)</pre>			<pre>['__annotations__', '__builtins__', '__doc__', '__loader__', '__name__', '__package__', '__spec__', 'traceback'] === Code Execution Successful ===</pre>	




Python divmod() Function

main.py	  	Run	Output
<pre>1 # Python divmod() function example 2 # Calling function 3 result = divmod(10,2) 4 # Displaying result 5 print(result)</pre>			<pre>(5, 0) === Code Execution Successful ===</pre>






Python enumerate() Function

main.py	  	Run	Output	
<pre>1 # Calling function 2 result = enumerate([1,2,3]) 3 # Displaying result 4 print(result) 5 print(list(result))</pre>			<pre><enumerate object at 0x7adcf65276a0> [(0, 1), (1, 2), (2, 3)] === Code Execution Successful ===</pre>	






Python hash() Function

main.py	  	Run	Output
<pre>1 # Calling function 2 result = hash(21) # integer value 3 result2 = hash(22.2) # decimal value 4 # Displaying result 5 print(result) 6 print(result2)</pre>			<pre>21 461168601842737174 === Code Execution Successful ===</pre>




Python help() Function

main.py	   	Output	
<pre>1 # Calling function 2 info = help() # No argument 3 # Displaying result 4 print(info)</pre>		<p>Welcome to Python 3.12's help utility! If this is your first time using Python, you should definitely check out the tutorial at https://docs.python.org/3.12/tutorial/.</p> <p>Enter the name of any module, keyword, or topic to get help on writing Python programs and using Python modules. To get a list of available modules, keywords, symbols, or topics, enter "modules", "keywords", "symbols", or "topics".</p> <p>Each module also comes with a one-line summary of what it does; to list the modules whose name or summary contain a given string such as "spam", enter "modules spam".</p> <p>To quit this help utility and return to the interpreter, enter "q" or "quit".</p> <p>help> == Session Ended. Please Run the code again ==</p>	






Python min() Function

main.py	   	Output	
<pre>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)</pre>		<p>325 1000.25</p> <p>=== Code Execution Successful ===</p>	

Python set() Function

main.py	   	Output	
<pre>1 # Calling function 2 result = set() # empty set 3 result2 = set('12') 4 result3 = set('javatpoint') 5 # Displaying result 6 print(result) 7 print(result2) 8 print(result3)</pre>		<p>set() {'2', '1'} {'a', 'v', 'o', 'n', 'j', 'i', 't', 'p'}</p> <p>=== Code Execution Successful ===</p>	

Python hex() Function

main.py	   	Output	
<pre>1 # Calling function 2 result = hex(1) 3 # integer value 4 result2 = hex(342) 5 # Displaying result 6 print(result) 7 print(result2)</pre>		<p>0x1 0x156</p> <p>=== Code Execution Successful ===</p>	






Python id() Function

main.py	Output
<pre>1 val = id("Javatpoint") # string object 2 val2 = id(1200) # integer object 3 val3 = id([25,336,95,236,92,3225]) # List object 4 # Displaying result 5 print(val) 6 print(val2) 7 print(val3)</pre>	<pre>138688839734192 138688841100816 138688842942912 === Code Execution Successful ===</pre>






Python setattr() Function

main.py	Output
<pre>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,"meghana") 10 print(student.id) 11 print(student.name) 12 #print(student.email) product error 13 setattr(student, 'email', 'meghana@gmail.com') # adding new attribute 14 print(student.email)</pre>	<pre>102 meghana meghana@gmail.com === Code Execution Successful ===</pre>





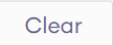
Python slice() Function

main.py	   	Output 
<pre> 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)</pre>	<pre> slice(None, 5, None) slice(0, 5, 3) === Code Execution Successful ===</pre>	





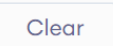
Python sorted() Function

main.py	   	Output 
<pre> 1 str = "javatpoint" # declaring string 2 # Calling function 3 sorted1 = sorted(str) # sorting string 4 # Displaying result 5 print(sorted1)</pre>	<pre> ['a', 'a', 'i', 'j', 'n', 'o', 'p', 't', 't', 'v'] === Code Execution Successful ===</pre>	




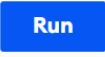
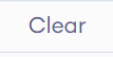
Python next() Function

main.py	   	Output 
<pre> 1 number = iter([256, 32, 82]) 2 item = next(number) 3 print(item) 4 item = next(number) 5 print(item) 6 item = next(number) 7 print(item)</pre>	<pre> 256 32 82 === Code Execution Successful ===</pre>	



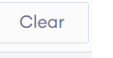
Python input() Function

main.py	   	Output 
<pre> 1 # Calling function 2 val = input("Enter a value: ") 3 # Displaying result 4 print("You entered:",val)</pre>	<pre> Enter a value: 44 You entered: 44 === Code Execution Successful ===</pre>	





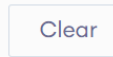
Python int() Function

main.py	   	Output	
<pre>1 val = int(10) # integer value 2 val2 = int(10.52) # float value 3 val3 = int('10') # string value 4 # Displaying result 5 print("integer values :",val, val2, val3)</pre>		<pre>integer values : 10 10 10 === Code Execution Successful ===</pre>	






Python isinstance() Function

main.py	   	Output	
<pre>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)) # instance of Student class 12 print(isinstance(lst, Student))</pre>		<pre>True False === Code Execution Successful ===</pre>	

Python oct() Function

main.py	   	Output	
<pre>1 # Calling function 2 val = oct(10) 3 # Displaying result 4 print("Octal value of 10:",val)</pre>		<pre>Octal value of 10: 0o12 === Code Execution Successful ===</pre>	






Python ord() Function

main.py	   	Output	
<pre>1 # Code point of an integer 2 print(ord('8')) 3 4 # Code point of an alphabet 5 print(ord('R')) 6 7 # Code point of a character 8 print(ord('&'))</pre>		<pre>56 82 38 === Code Execution Successful ===</pre>	





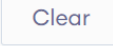
Python pow() Function

main.py	   	Output	
<pre>1 # positive x, positive y (x**y) 2 print(pow(4, 2)) 3 4 # negative x, positive y 5 print(pow(-4, 2)) 6 7 # positive x, negative y (x**-y) 8 print(pow(4, -2)) 9 10 # negative x, negative y 11 print(pow(-4, -2))</pre>		<pre>16 16 0.0625 0.0625 === Code Execution Successful ===</pre>	

Python print() Function

main.py	   	Output	
<pre>1 print("Python is programming language." 2) 3 x = 7 4 # Two objects passed 5 print("x =", x) 6 7 y = x 8 # Three objects passed 9 print('x =', x, '= y')</pre>		<pre>Python is programming language. x = 7 x = 7 = y === Code Execution Successful ===</pre>	

Python range() Function

main.py	   	Output	
<pre>1 # empty range 2 print(list(range(0))) 3 4 # using the range(stop) 5 print(list(range(4))) 6 7 # using the range(start, stop) 8 print(list(range(1,7)))</pre>		<pre>[] [0, 1, 2, 3] [1, 2, 3, 4, 5, 6] === Code Execution Successful ===</pre>	

Python reversed() Function

main.py	Output
<pre>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)))</pre>	<pre>['a', 'v', 'a', 'J'] ['a', 'v', 'a', 'J'] [11, 10, 9, 8] [5, 7, 2, 1] === Code Execution Successful ===</pre>

Python round() Function

main.py	Output
<pre>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))</pre>	<pre>10 11 7 === Code Execution Successful ===</pre>

Python isinstance() Function

main.py	Output
<pre>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__('square') 8 9 print(isinstance(Square, Rectangle)) 10 print(isinstance(Square, list)) 11 print(isinstance(Square, (list, Rectangle))) 12 print(isinstance(Rectangle, (list, Rectangle)))</pre>	<pre>True False True True === Code Execution Successful ===</pre>

Python tuple() Function

main.py	Run	Output
<pre>1 t1 = tuple() 2 print('t1=', t1) 3 4 # creating a tuple from a list 5 t2 = tuple([1, 6, 9]) 6 print('t2=', t2) 7 8 # creating a tuple from a string 9 t1 = tuple('Java') 10 print('t1=',t1) 11 12 # creating a tuple from a dictionary 13 t1 = tuple({4: 'four', 5: 'five'}) 14 print('t1=',t1)</pre>		<pre>t1= () t2= (1, 6, 9) t1= ('J', 'a', 'v', 'a') t1= (4, 5) === Code Execution Successful ===</pre>




Python type()

main.py	Run	Output
<pre>1 List = [4, 5] 2 print(type(List)) 3 4 Dict = {4: 'four', 5: 'five'} 5 print(type(Dict)) 6 7 class Python: 8 a = 0 9 10 InstanceOfPython = Python() 11 print(type(InstanceOfPython))</pre>		<pre><class 'list'> <class 'dict'> <class '__main__.Python'> === Code Execution Successful ===</pre>

Python vars() function






main.py	Run	Output
<pre>1 class Python: 2 def __init__(self, x = 7, y = 9): 3 self.x = x 4 self.y = y 5 6 InstanceOfPython = Python() 7 print(vars(InstanceOfPython))</pre>		<pre>{'x': 7, 'y': 9} === Code Execution Successful ===</pre>

Python zip() Function





main.py	   	Output	
<pre>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)</pre>		<pre>[] {(5, 'five'), (6, 'six'), (4, 'four')} === Code Execution Successful ===</pre>	

Python Lambda Functions






Example 1:

main.py	   	Output	
<pre>1 add = lambda num: num + 4 2 print(add(6))</pre>		<pre>10 === Code Execution Successful ===</pre>	





Example 2:

main.py	   	Output	
<pre>1 def add(num): 2 return num + 4 3 print(add(6))</pre>		<pre>10 === Code Execution Successful ===</pre>	






Example 3:

main.py	   	Output	
<pre>1 a = lambda x, y : (x * y) 2 print(a(4, 5))</pre>		<pre>20 === Code Execution Successful ===</pre>	






Example 4:

main.py	   	Output	
<pre>1 a = lambda x, y, z : (x + y + z) 2 print(a(4, 5, 5))</pre>		<pre>14 === Code Execution Successful ===</pre>	






What's the Distinction Between Lambda and Def Functions?

main.py	   	Output	
<pre>1 def reciprocal(num) : 2 return 1 / num 3 4 lambda_reciprocal = lambda num: 1 / num 5 print("Def keyword: ", reciprocal(6)) 6 print("Lambda keyword: ", lambda_reciprocal(6))</pre>		<pre>Def keyword: 0.16666666666666666 Lambda keyword: 0.16666666666666666 === Code Execution Successful ===</pre>	






Using Lambda Function with filter()

main.py	   	Output	
<pre>1 list_ = [35, 12, 69, 55, 75, 14, 73] 2 odd_list = list(filter(lambda num: (num % 2 != 0) , list_)) 3 print('The list of odd number is:' ,odd_list)</pre>		<pre>The list of odd number is: [35, 69, 55, 75, 73] === Code Execution Successful ===</pre>	

Using Lambda Function with map()

main.py	   	Output	
<pre>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)</pre>		<pre>Square of each number in the given list: [4, 16, 25, 1, 9, 49, 64, 81, 100] === Code Execution Successful ===</pre>	

Using Lambda Function with if-else

main.py	   	Output	
<pre>1 Minimum = lambda x, y : x if (x < y) else y 2 print('The greater number is:', Minimum(35, 74))</pre>		<pre>The greater number is: 35 === Code Execution Successful ===</pre>	

Using Lambda with Multiple Statements

main.py	Output
<pre>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)</pre>	<p>The third largest number from every sub list is: [6, 54, 5]</p> <p>=== Code Execution Successful ===</p>

Python Modules

The screenshot shows a Python 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 `Code` tab is selected, showing the execution output:

```
[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
```

Importing and also Renaming

The screenshot shows a Python IDE with the following code in the editor:

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

Below the code editor, the output is displayed:

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

Python from...import Statement

The screenshot shows a Python IDE with the following code in the editor:

```
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 )
```

Below the code editor, the output is displayed:

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

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

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)
4
```

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']

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_', '_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_', '_rfind_', '_rindex_', '_rjust_', '_rpartition_', '_rsplit_', '_rstrip_', '_split_', '_splitlines_', '_startswith_', '_strip_', '_swapcase_', '_title_', '_translate_', '_upper_', '_zfill_', ...]

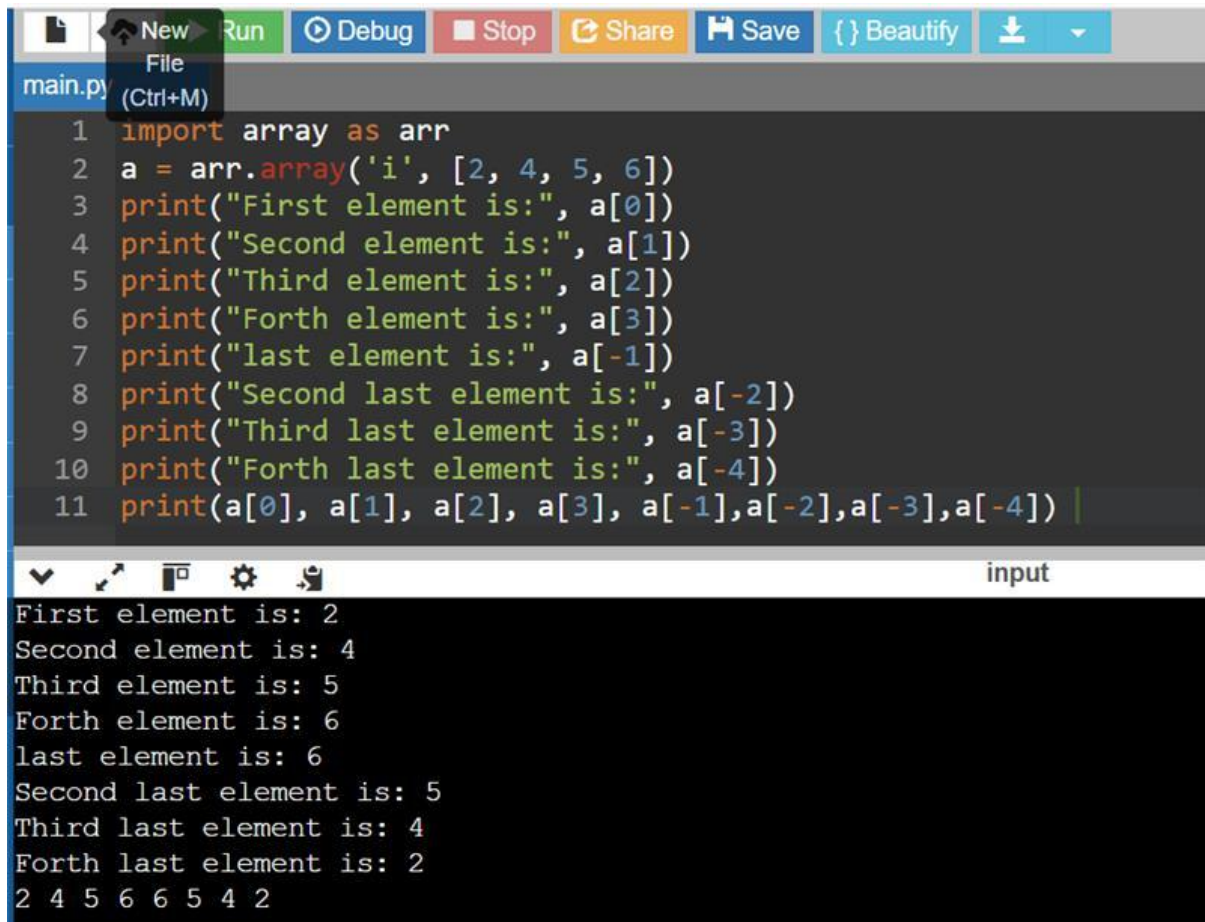
Namespaces and Scoping

```
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
```

The number is: 204
The number is: 404

PYTHON ARRAYS

1. Accessing array elements



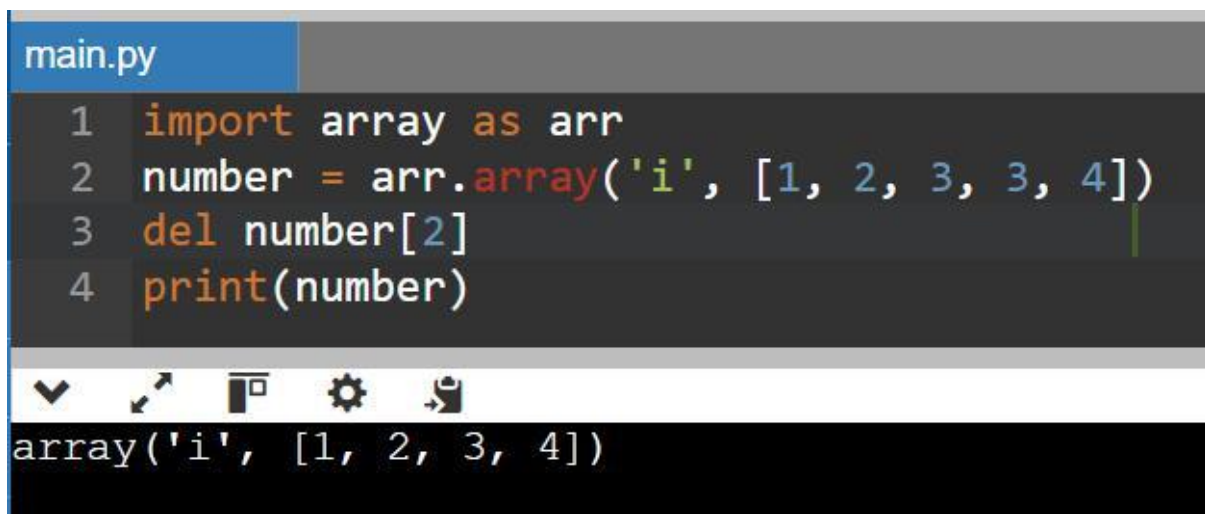
The screenshot shows a Python IDE with a toolbar at the top containing buttons for New, Run, Debug, Stop, Share, Save, Beautify, and a download icon. The file name 'main.py' is visible. The code in the editor is as follows:

```
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 containing buttons for 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 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

```
File  
(Ctrl+M)  
main.py  
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

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

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

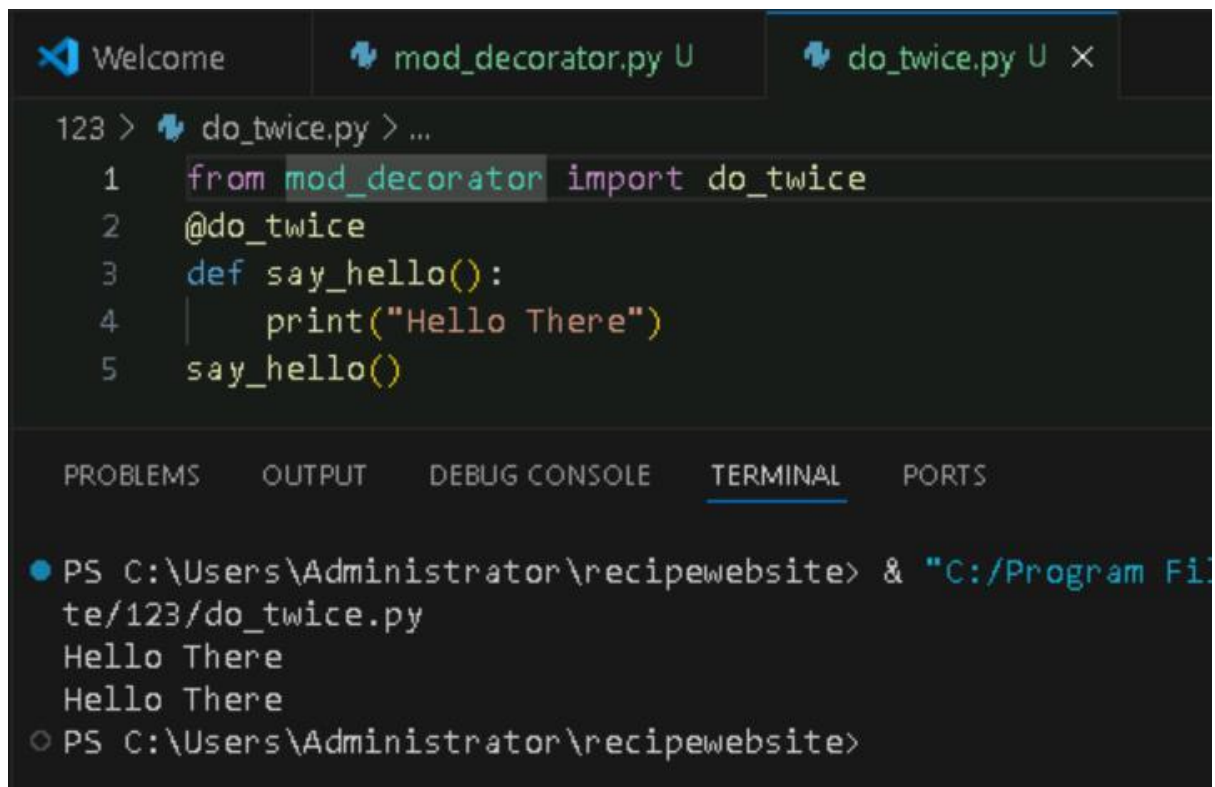
Hello

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

Reusing Decorator

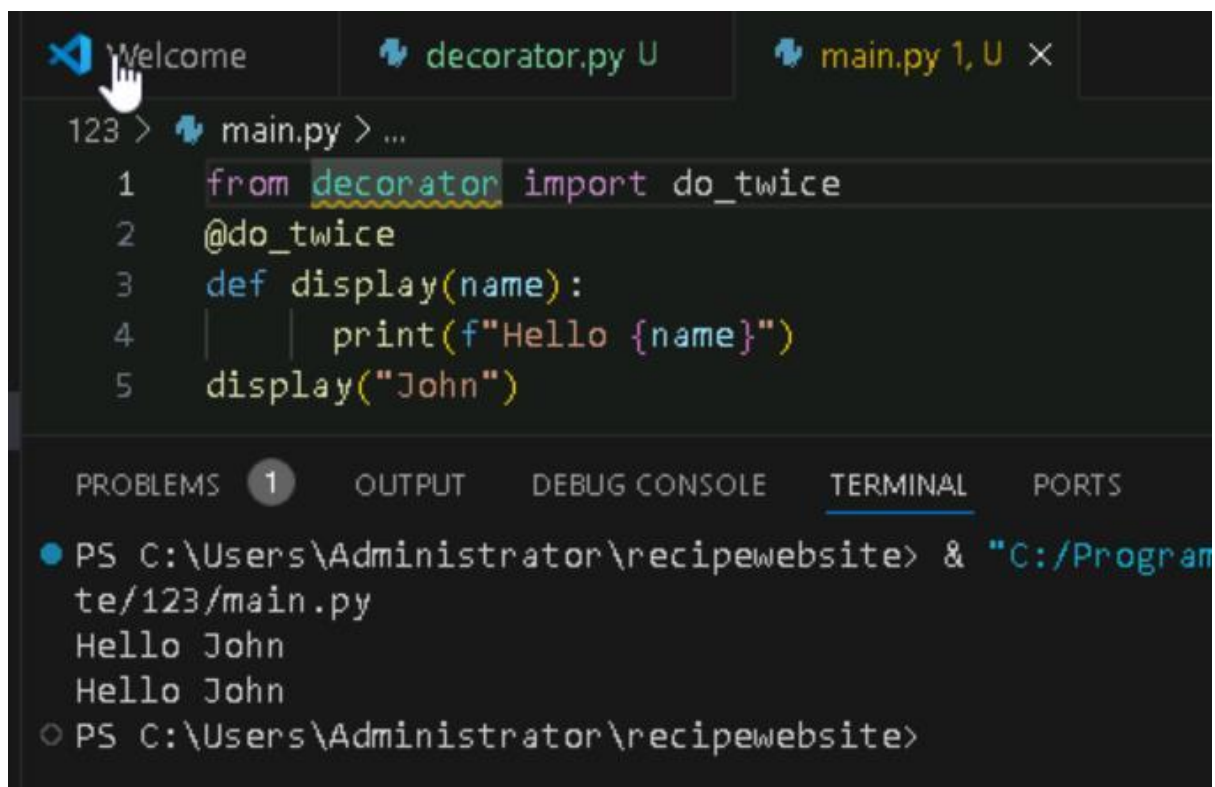


```
123 > do_twice.py > ...
1 from mod_decorator import do_twice
2 @do_twice
3 def say_hello():
4     print("Hello There")
5 say_hello()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

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

Python Decorator with Argument

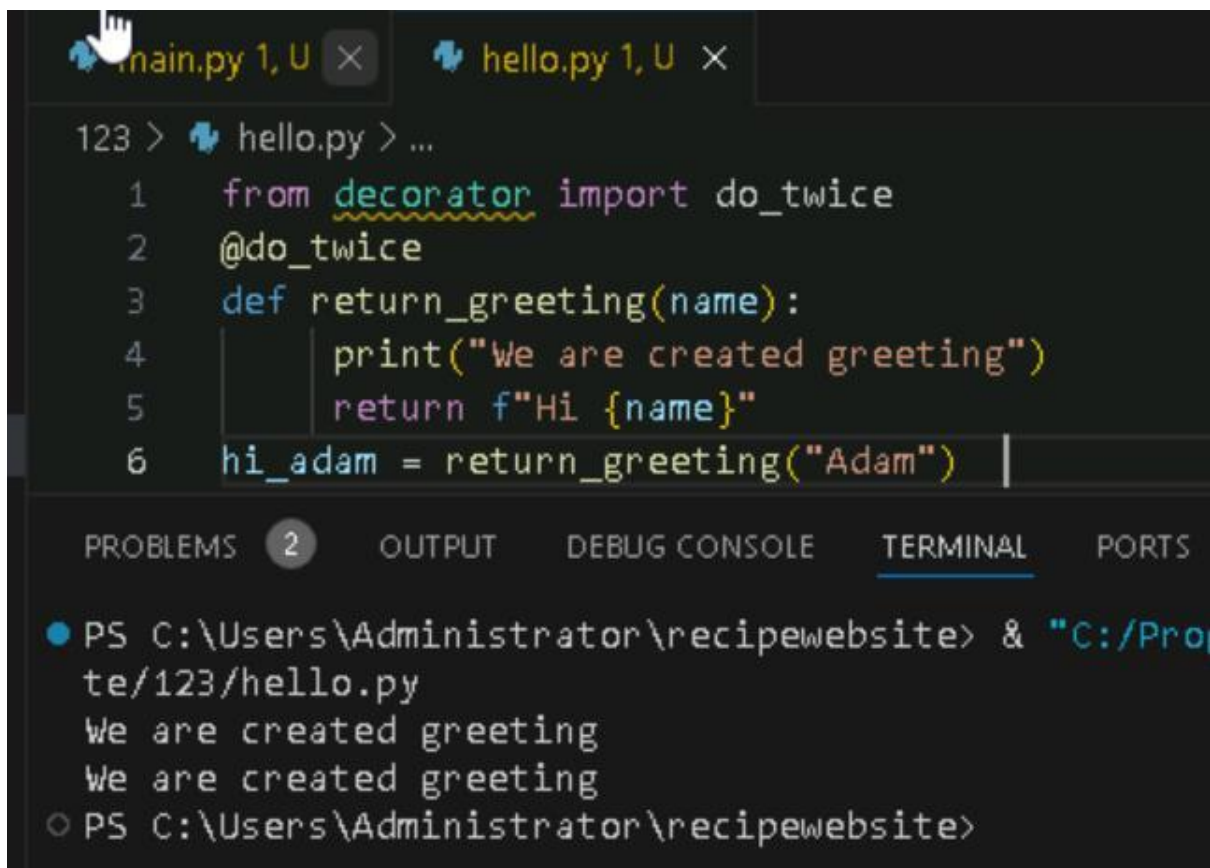


```
123 > main.py > ...
1 from decorator import do_twice
2 @do_twice
3 def display(name):
4     print(f"Hello {name}")
5 display("John")
```

PROBLEMS **1** OUTPUT DEBUG CONSOLE TERMINAL PORTS

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

Returning Values from Decorated Functions



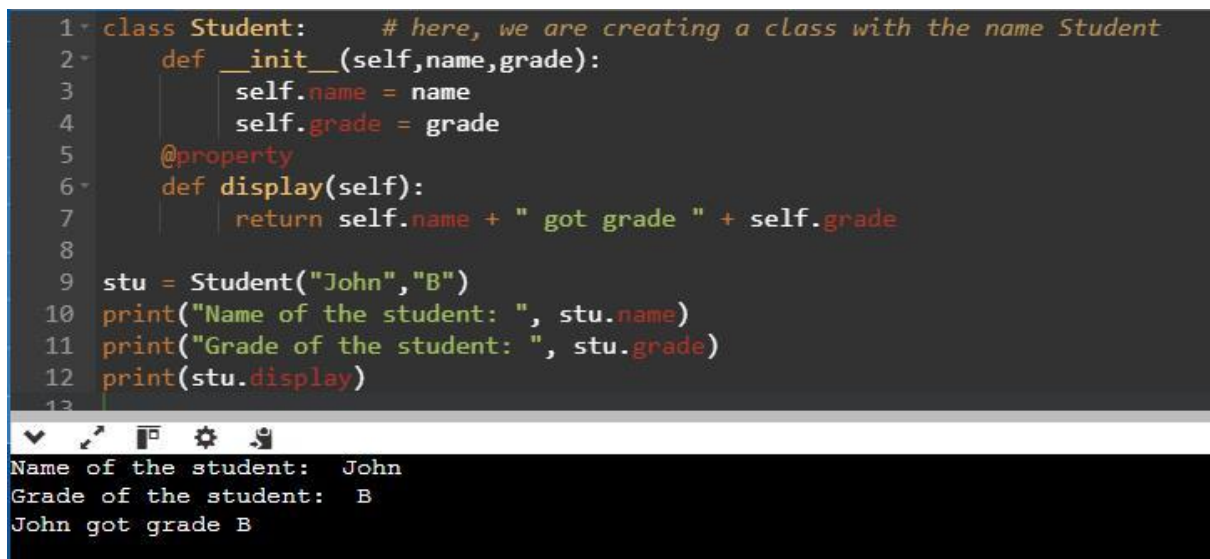
The screenshot shows a code editor with two tabs: 'main.py 1, U' and 'hello.py 1, U'. The 'hello.py' tab is active, displaying the following Python code:

```
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")
```

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

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

Fancy Decorators



The screenshot shows a code editor with a Python script defining a class and using a decorator. The code is as follows:

```
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
```

Below the code editor, the terminal shows the output of the script:

```
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

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 decorated
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

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 argument
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

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
```

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

