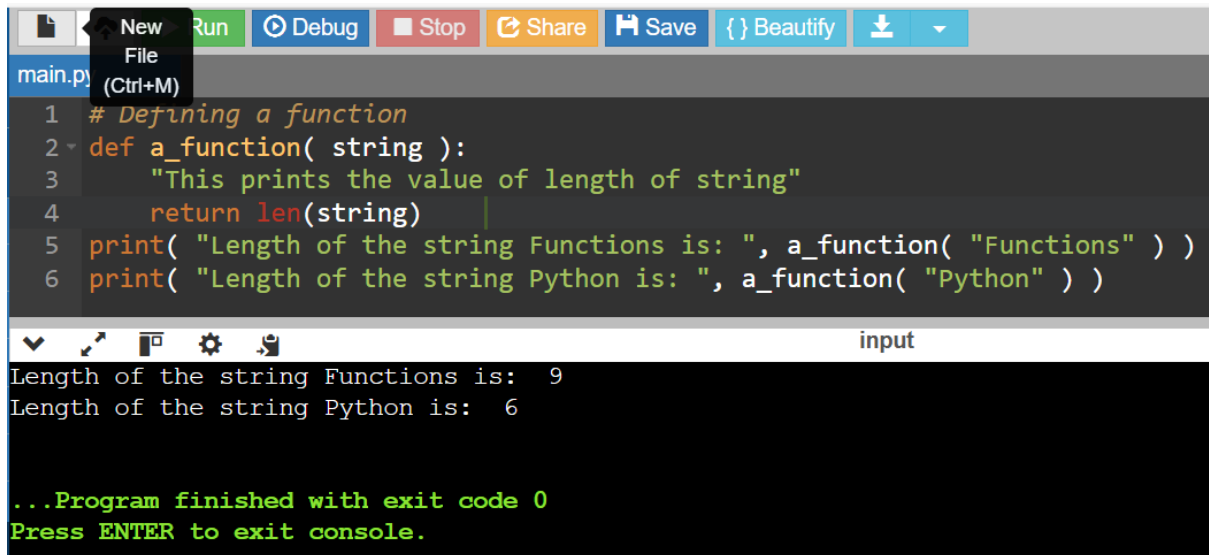


PYTHON PROJECT

PYTHON FUNCTIONS

1. 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 with the strings 'Functions' and 'Python'. The output shows the lengths 9 and 6 respectively. The interface includes a toolbar with buttons for New File, Run, Debug, Stop, Share, Save, Beautify, and a download icon. A dropdown menu is open for the 'New File' button.

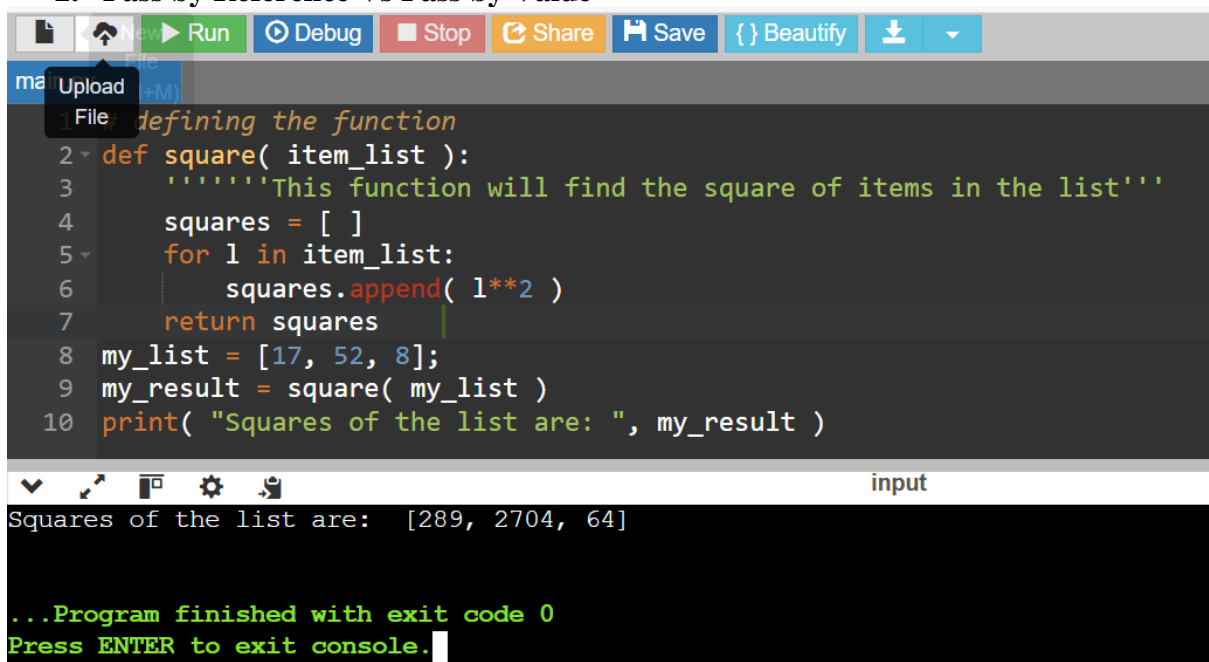
```
1 # Defining a function
2 def a_function( string ):
3     "This prints the value of length of string"
4     return len(string)
5 print( "Length of the string Functions is: ", a_function( "Functions" ) )
6 print( "Length of the string Python is: ", a_function( "Python" ) )
```

input

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 file named 'main.py'. The code defines a function 'square' that takes a list and appends the square of each element. It then calls this function with a list [17, 52, 8]. The output shows the modified list [289, 2704, 64]. The interface includes a toolbar with buttons for New File, Run, Debug, Stop, Share, Save, Beautify, and a download icon. A dropdown menu is open for the 'New File' button.

```
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 )
```

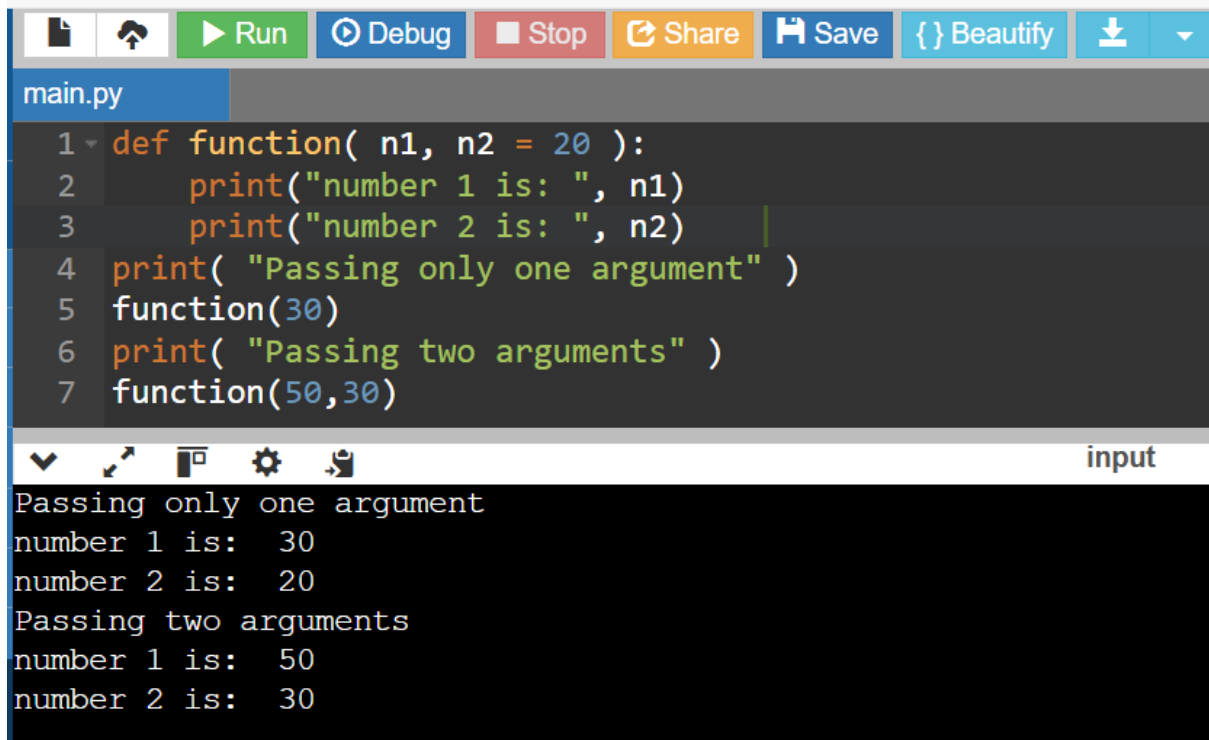
input

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

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

FUNCTION ARGUMENTS

1. Default arguments



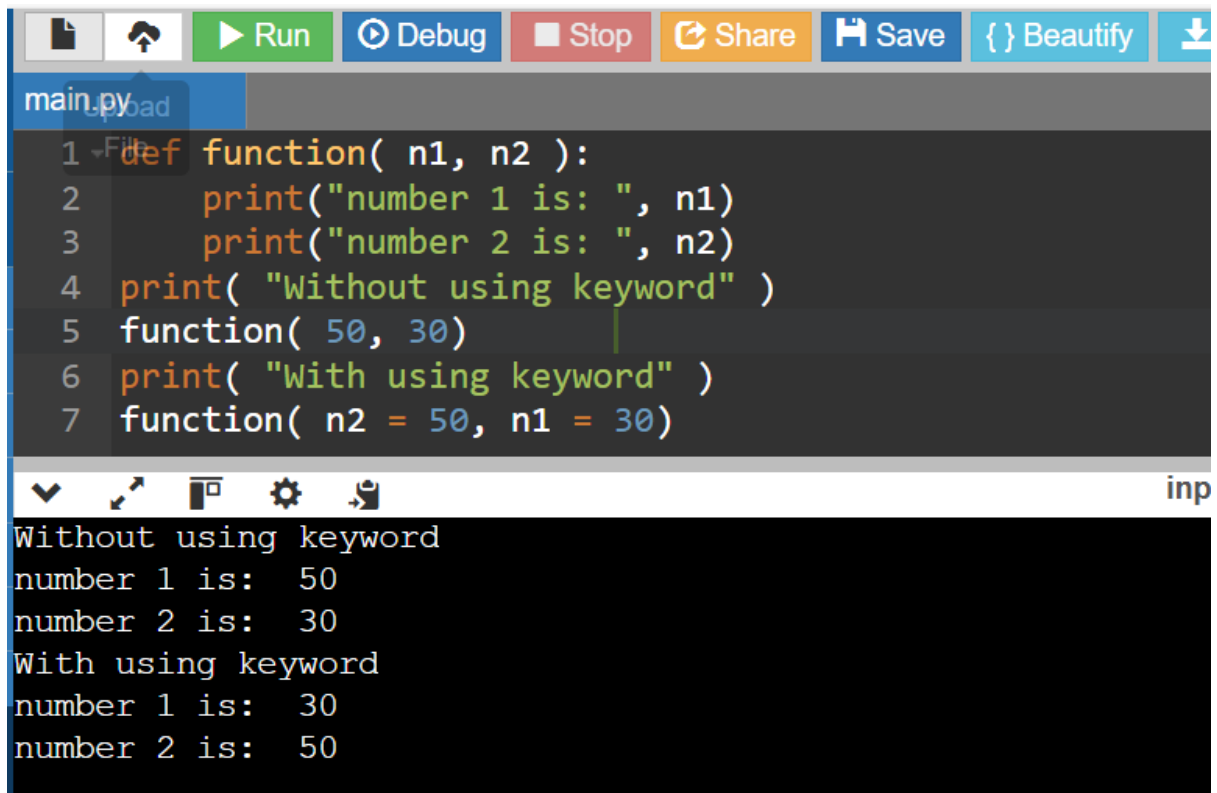
The screenshot shows a code editor with a toolbar at the top containing icons for file operations, a 'Run' button, 'Debug', 'Stop', 'Share', 'Save', 'Beautify', and a download icon. The file name 'main.py' is visible in the tab. The code defines a function 'function' with two parameters: 'n1' and 'n2 = 20'. The function prints the values of 'n1' and 'n2'. It then calls the function three times: first with one argument (30), then with two arguments (50, 30), and finally with two arguments (50, 30). The output shows that when only one argument is passed, it defaults to 20 for 'n2'. When two arguments are passed, it uses the provided values.

```
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



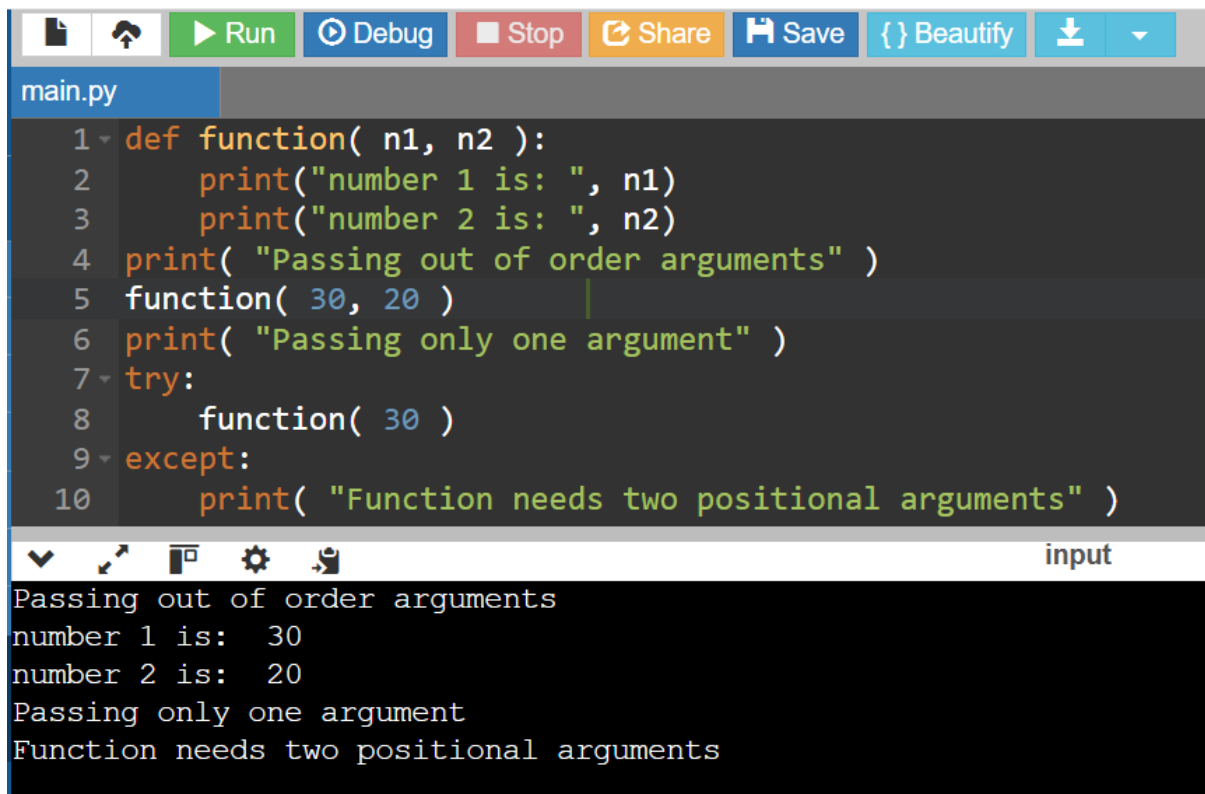
The screenshot shows a code editor with a toolbar at the top containing icons for file operations, a 'Run' button, 'Debug', 'Stop', 'Share', 'Save', 'Beautify', and a download icon. The file name 'main.py' is visible in the tab. The code defines a function 'function' with two parameters: 'n1' and 'n2'. The function prints the values of 'n1' and 'n2'. It then calls the function twice: first with two arguments (50, 30) without keywords, and then with two arguments (n2 = 50, n1 = 30) using keyword arguments. The output shows that when keywords are used, the arguments are assigned to the parameters in the order specified in the function call, regardless of their position in the parameter list.

```
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

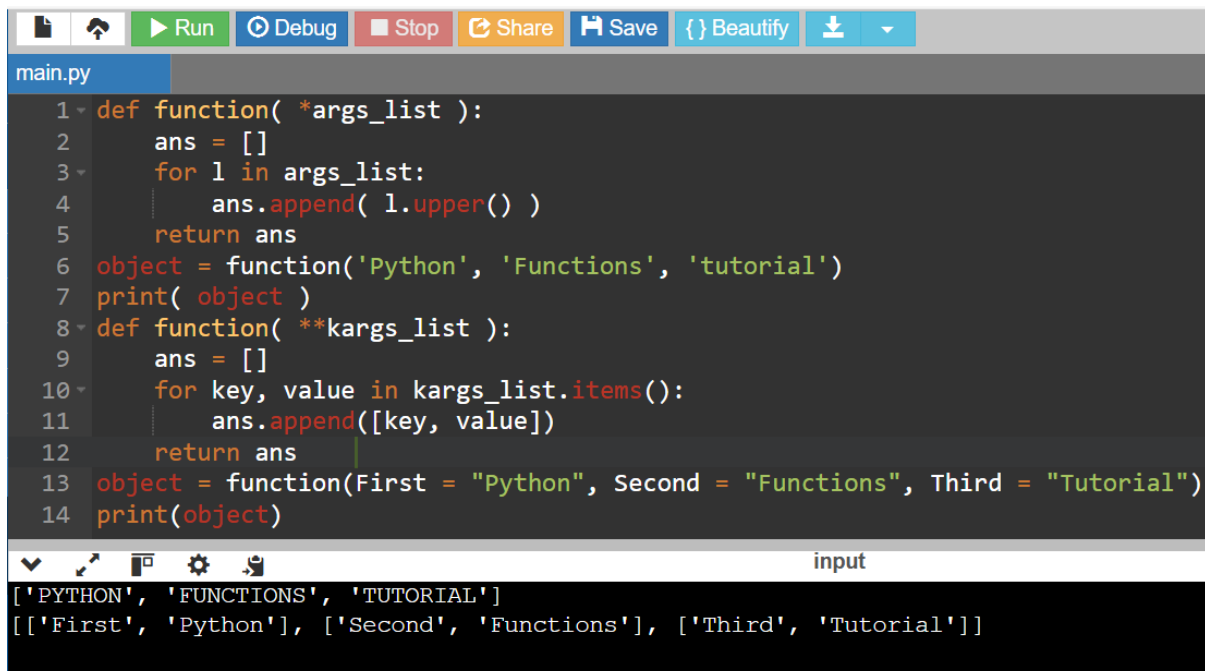


The screenshot shows a code editor with a toolbar at the top containing icons for file operations, a 'Run' button, 'Debug', 'Stop', 'Share', 'Save', 'Beautify', and download/upload options. The file is named 'main.py'. The code defines a function 'function' that takes two positional arguments, 'n1' and 'n2'. It prints the values of 'n1' and 'n2', then prints 'Passing out of order arguments'. Below this, it 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'.

```
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" )
```

The output in the console shows the execution of the code: 'Passing out of order arguments', 'number 1 is: 30', 'number 2 is: 20', 'Passing only one argument', and 'Function needs two positional arguments'.

4. Variable-length arguments

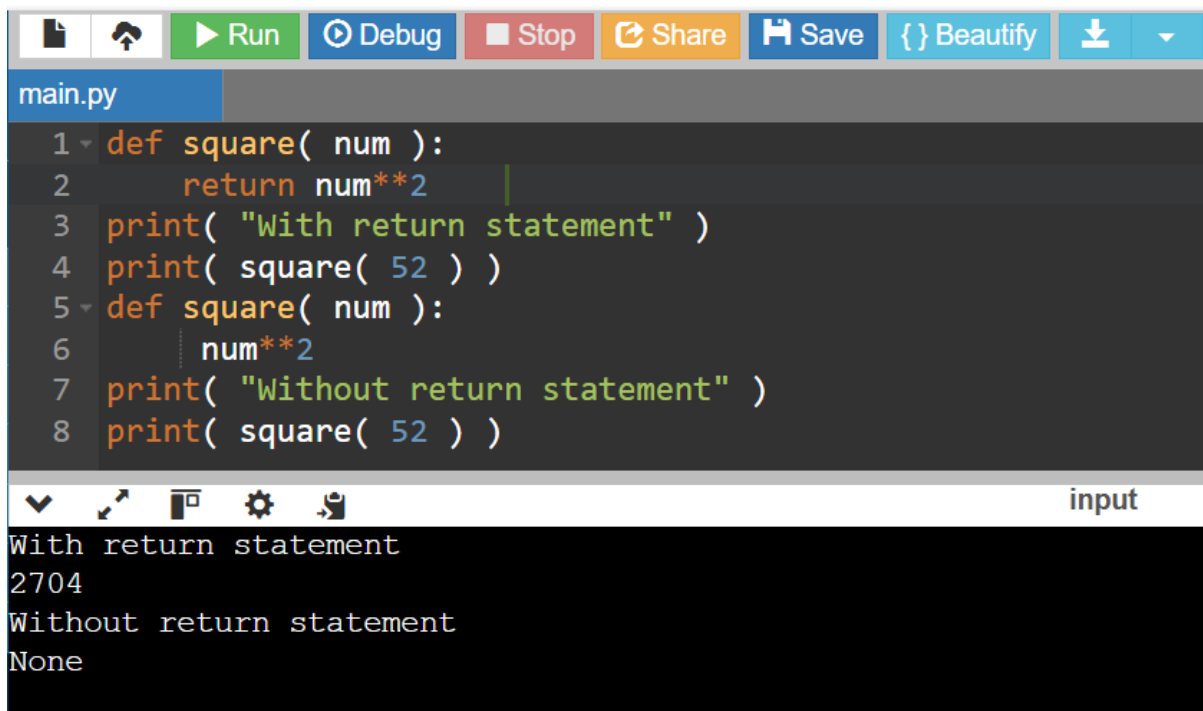


The screenshot shows a code editor with a toolbar at the top containing icons for file operations, a 'Run' button, 'Debug', 'Stop', 'Share', 'Save', 'Beautify', and download/upload options. The file is named 'main.py'. The code defines two functions. The first function, 'function', takes a variable-length argument '*args_list' and returns a list of the arguments converted to uppercase. The second function, 'function', takes a variable-length argument '**kargs_list' and returns a list of the arguments as key-value pairs. The code calls the first function with 'Python', 'Functions', and 'tutorial', and prints the result. It then calls the second function with keyword arguments 'First = "Python"', 'Second = "Functions"', and 'Third = "Tutorial"', and prints the result.

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

The output in the console shows the execution of the code: ['PYTHON', 'FUNCTIONS', 'TUTORIAL'] and [['First', 'Python'], ['Second', 'Functions'], ['Third', 'Tutorial']].

RETURN STATEMENT



The screenshot shows a Python IDE with a toolbar at the top containing icons for file operations, a 'Run' button, 'Debug', 'Stop', 'Share', 'Save', 'Beautify', and a download icon. The editor window, titled 'main.py', contains the following code:

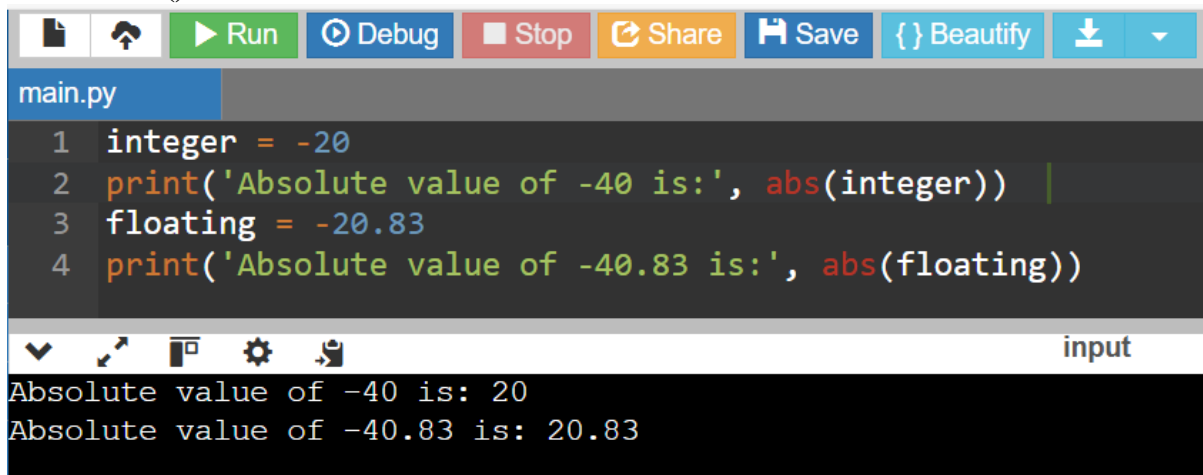
```
1 def square( num ):
2     return num**2
3 print( "With return statement" )
4 print( square( 52 ) )
5 def square( num ):
6     num**2
7 print( "Without return statement" )
8 print( square( 52 ) )
```

Below the editor is a console window labeled 'input' showing the output of the program:

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

PYTHON BUILT-IN FUNCTIONS

1. Abs () function



The screenshot shows a Python IDE with a toolbar at the top containing icons for file operations, a 'Run' button, 'Debug', 'Stop', 'Share', 'Save', 'Beautify', and a download icon. The editor window, titled 'main.py', contains the following code:

```
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))
```

Below the editor is a console window labeled 'input' showing the output of the program:

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

2. All () function



The screenshot shows a code editor with a toolbar at the top containing icons for file operations, a 'Run' button, a 'Debug' button, a 'Stop' button, a 'Share' button, a 'Save' button, a 'Beautify' button, and a download icon. The file name 'main.py' is displayed. The code consists of ten lines: 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))`. Below the code editor, the output is displayed: 'True', 'False', 'False', 'False', 'True'.

```
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

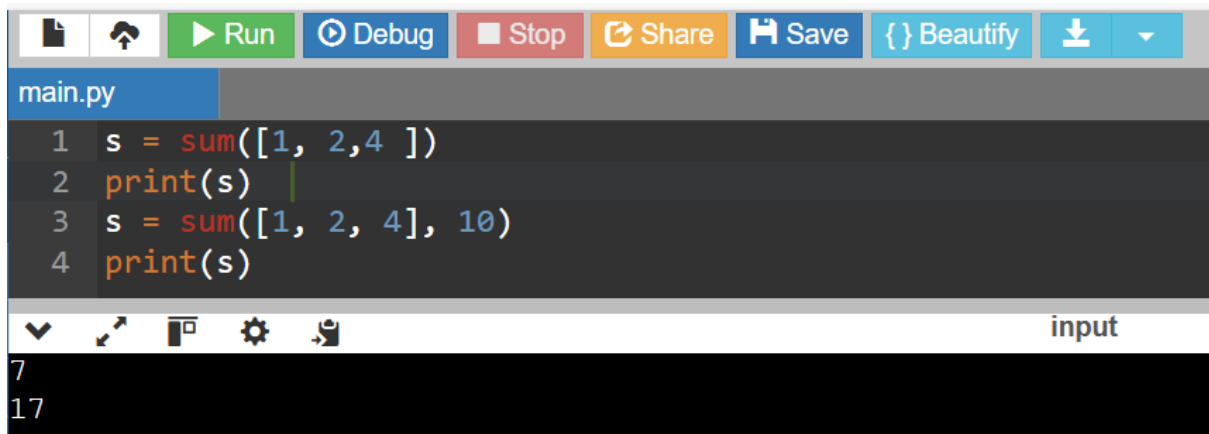


The screenshot shows a code editor with a toolbar at the top containing icons for file operations, a 'Run' button, a 'Debug' button, a 'Stop' button, a 'Share' button, a 'Save' button, and a download icon. The file name 'main.py' is displayed. The code consists of twelve lines: 1. `test1 = []`, 2. `print(test1, '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))`. Below the code editor, the output is displayed: '[] is False', '[0] is True', '0.0 is False', 'None is False', 'True is True', 'Easy string is True'.

```
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



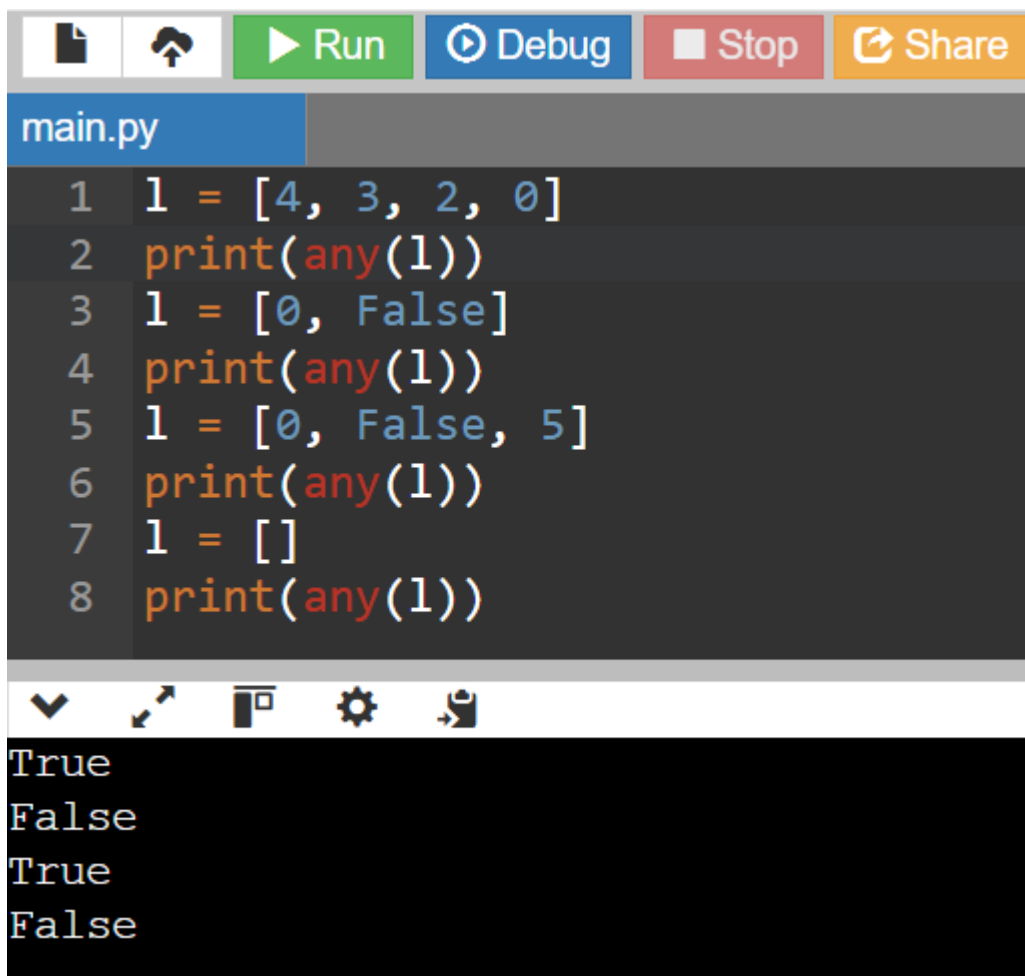
The screenshot shows a code editor with a toolbar at the top containing icons for file operations, a 'Run' button, 'Debug', 'Stop', 'Share', 'Save', 'Beautify', and a download icon. The file name 'main.py' is displayed. The code in the editor is as follows:

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

Below the code editor, there is a terminal window with the word 'input' on the right. The terminal output shows the results of the code execution:

```
7
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, there is a terminal window showing the output of the code execution:

```
True
False
True
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 : [ 1[ 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.
```

PYTHON MODULES

```
example_module.py  main_program.py x
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)

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  Code
[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
```

1. 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

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']
```


2. Importing and Renaming

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

The value of euler's number is 2.718281828459045

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

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

5. 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_',]

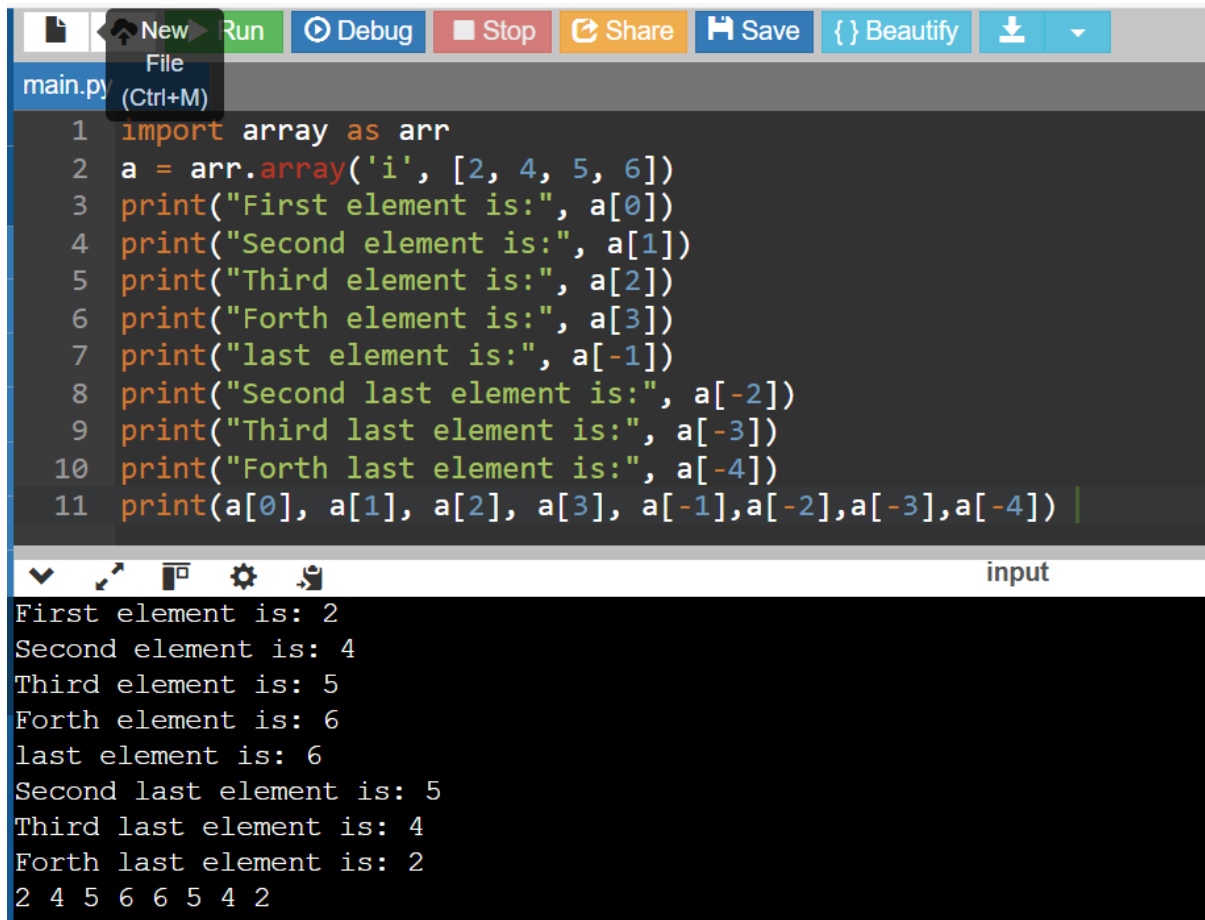
6. Namespaces and Scoping

```
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



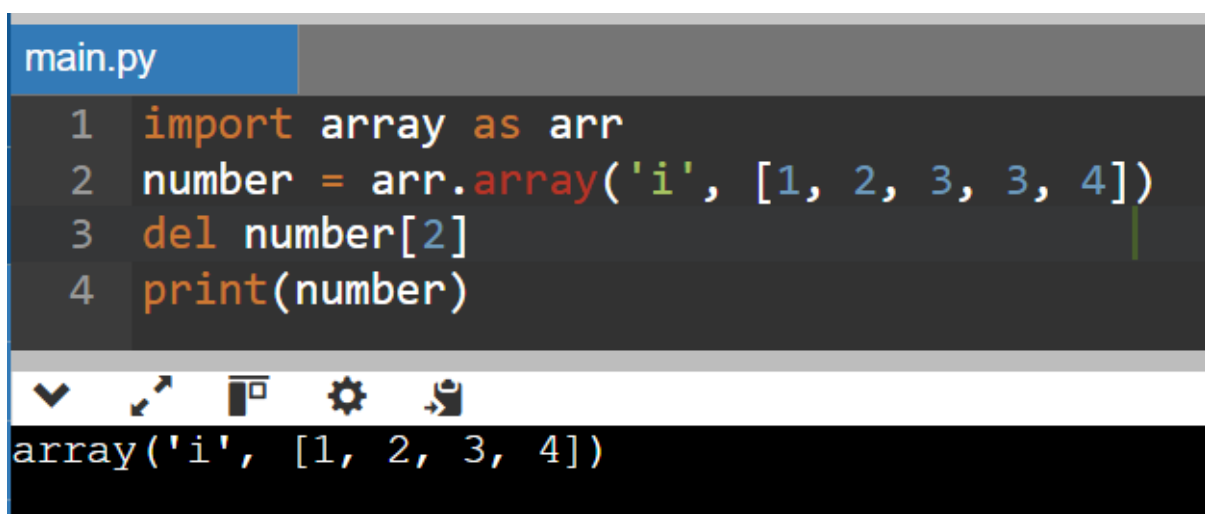
The screenshot shows a Python IDE with a toolbar at the top containing buttons for New, Run, Debug, Stop, Share, Save, Beautify, and a download icon. The file explorer on the left shows 'main.py'. The code editor contains the following Python code:

```
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 console at the bottom shows the following output:

```
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 explorer on the left shows 'main.py'. The code editor contains the following Python code:

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

The output console at the bottom shows the following output:

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

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.

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):          # here, we are creating a function and passing the parameter
2     return x+1        # here, we are returning the data of function
3 def sub(x):           # here, we are creating a function and passing the parameter
4     return x-1        # here, we are returning the data of function
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     return inner
10 divide1 = outer_div(divide)
11 divide1(2,4)
```

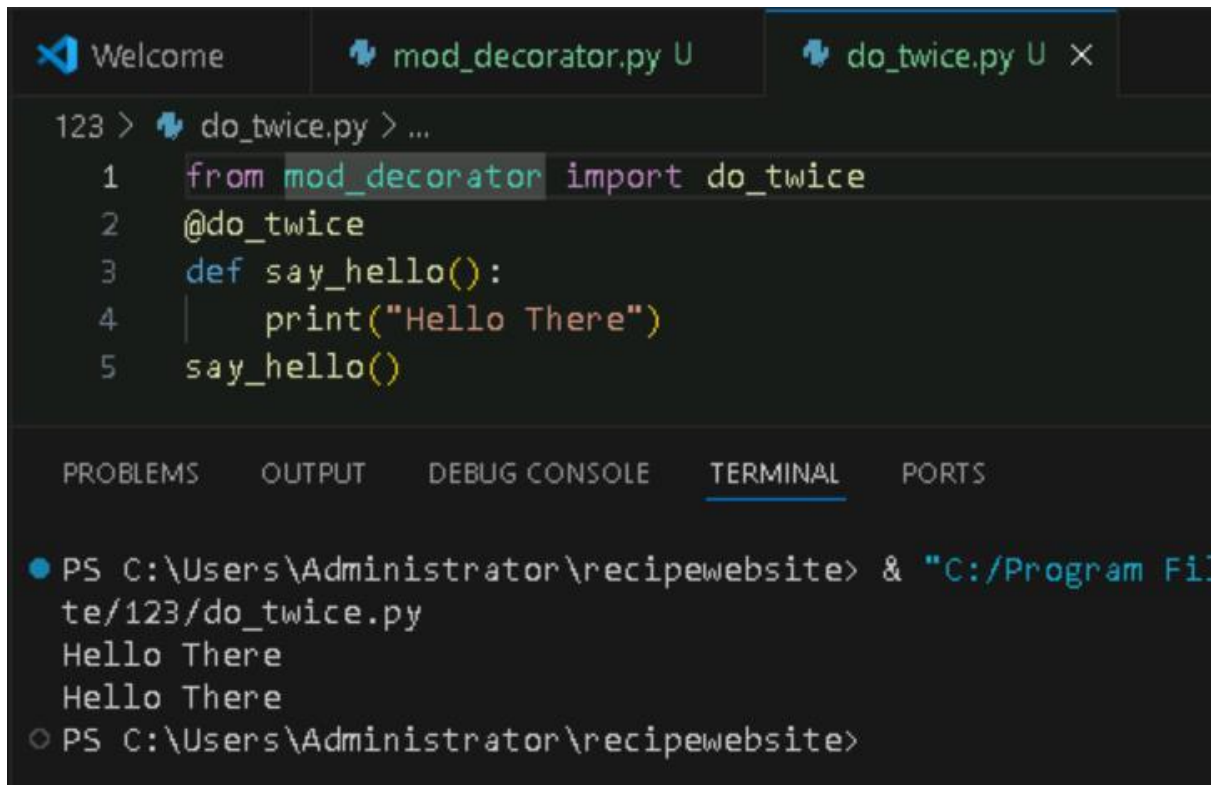
Hello

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



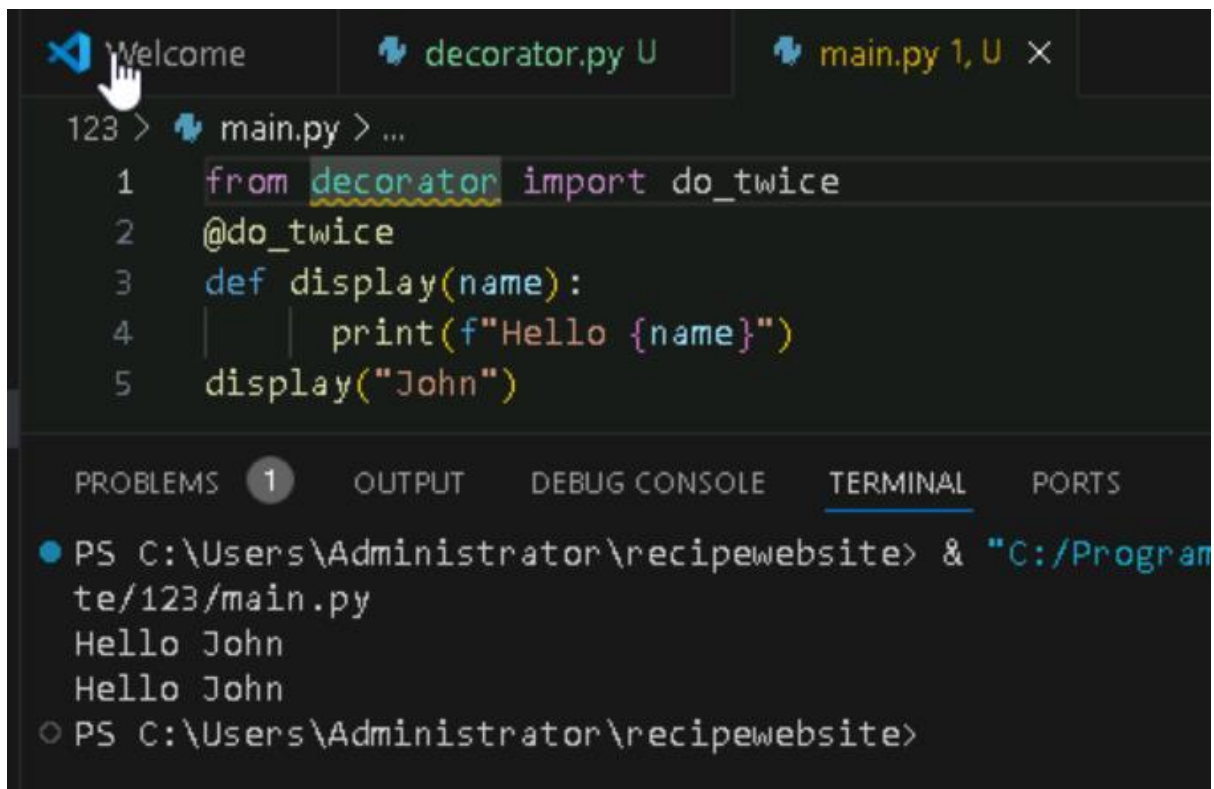
The screenshot shows the Visual Studio Code interface with two tabs: 'mod_decorator.py U' and 'do_twice.py U'. The 'do_twice.py' file is open, showing a Python script that imports the 'do_twice' decorator from 'mod_decorator' and applies it to a 'say_hello' function. The terminal output shows the function being called twice, resulting in 'Hello There' being printed twice.

```
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/Scripts/python.exe C:/Users/Administrator/recipewebsite/123/do_twice.py
Hello There
Hello There
○ PS C:\Users\Administrator\recipewebsite>
```

5. Python Decorator with Argument



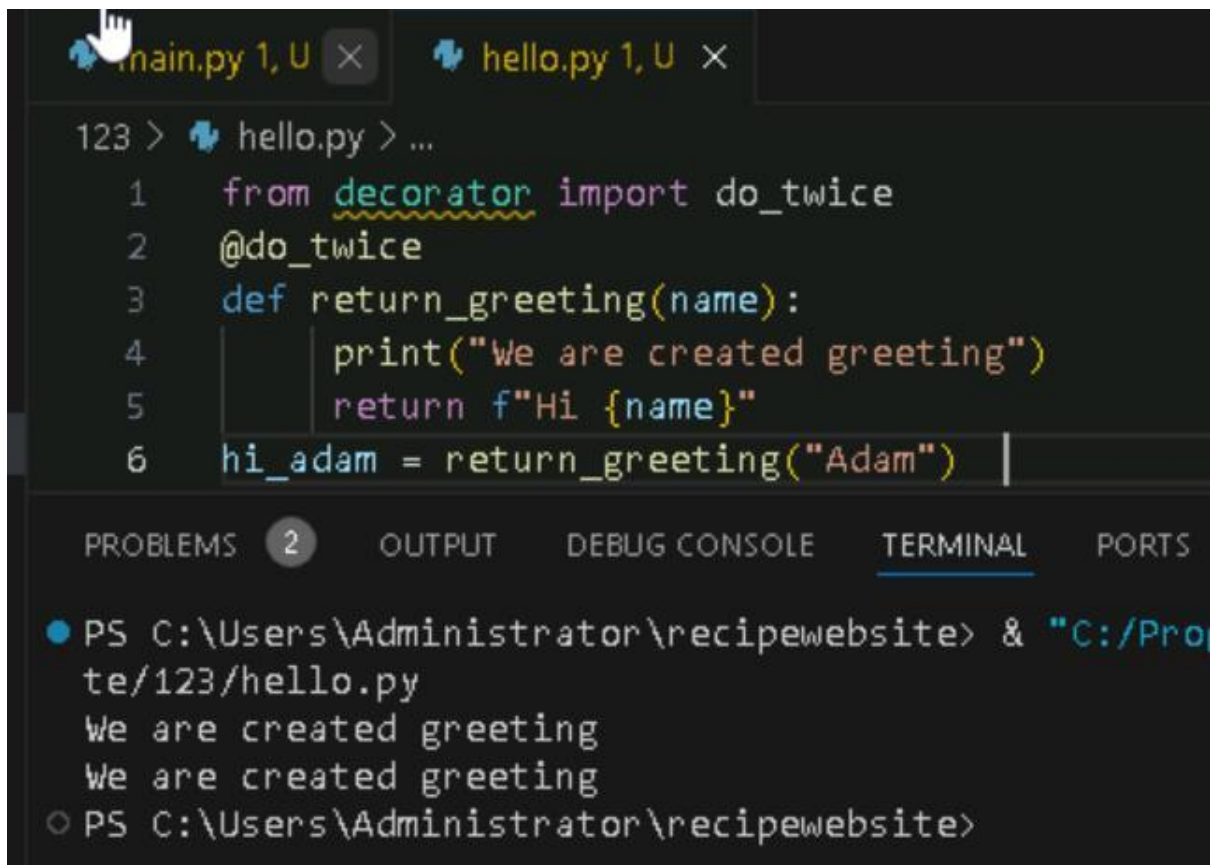
The screenshot shows the Visual Studio Code interface with two tabs: 'decorator.py U' and 'main.py 1, U'. The 'main.py' file is open, showing a Python script that imports the 'do_twice' decorator from 'decorator' and applies it to a 'display' function. The terminal output shows the function being called twice, resulting in 'Hello John' being printed twice.

```
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/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



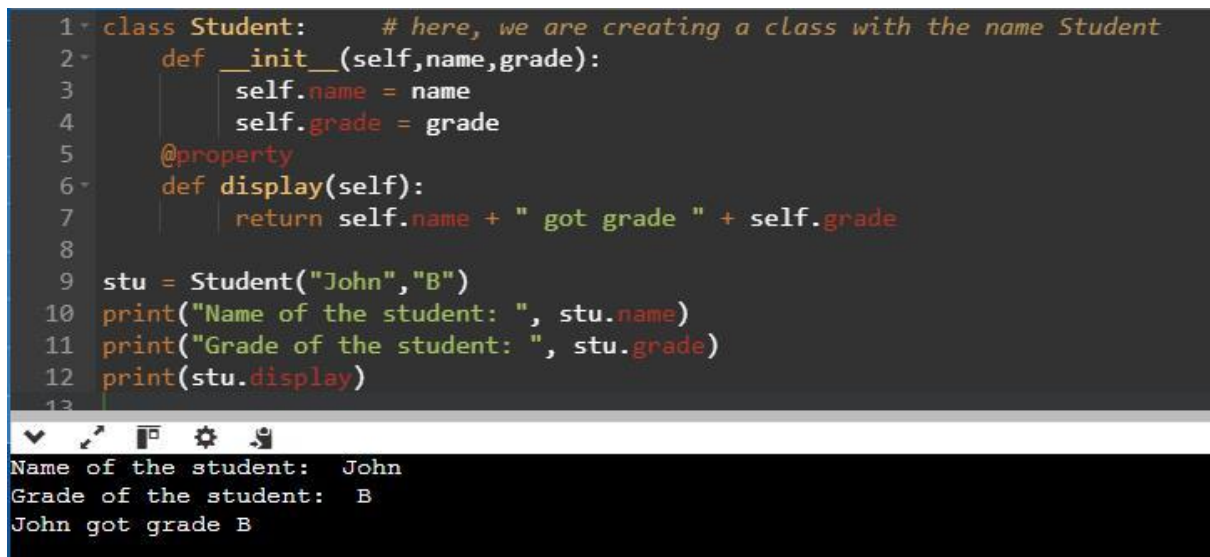
The screenshot shows a code editor with two tabs: `main.py 1, U` and `hello.py 1, U`. The `hello.py` tab is active, displaying the following Python code:

```
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>
```

7. Fancy Decorators



The screenshot shows a code editor with a single tab containing the following Python code:

```
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 output is shown:

```
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
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

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 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

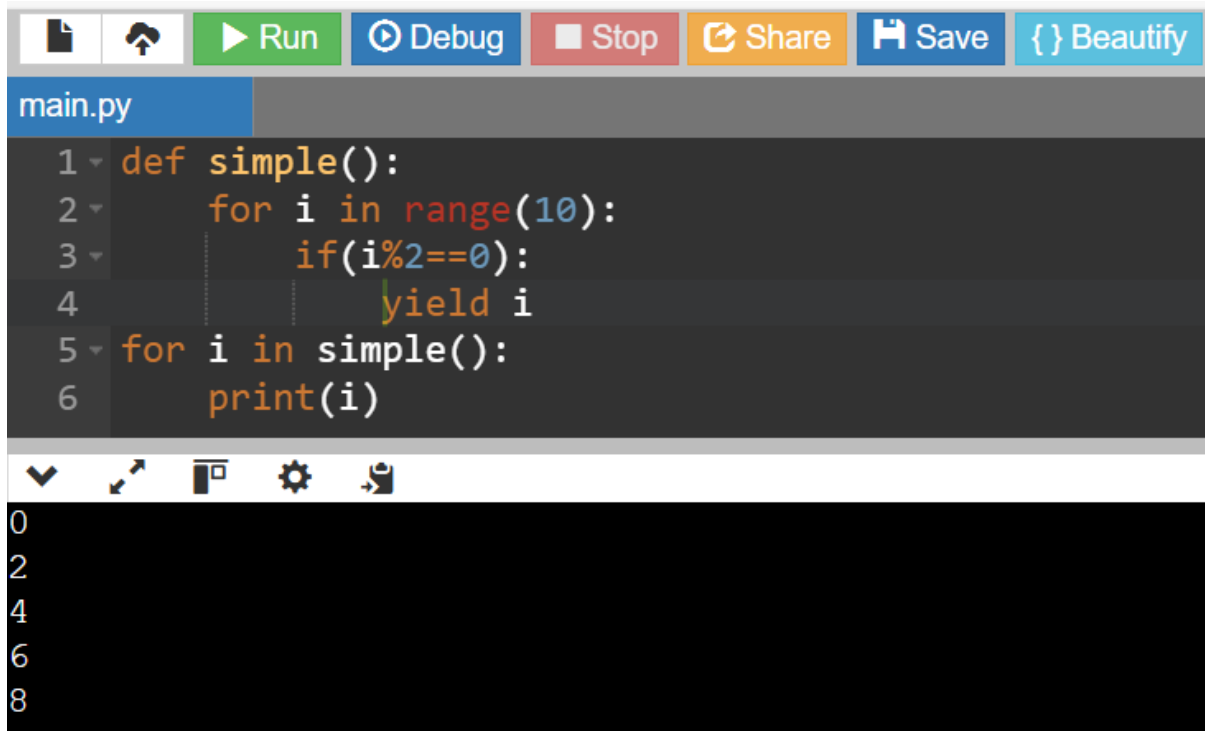
10. Classes as Decorators

```
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
```

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

PYTHON GENERATORS

1. To create Generator function in python

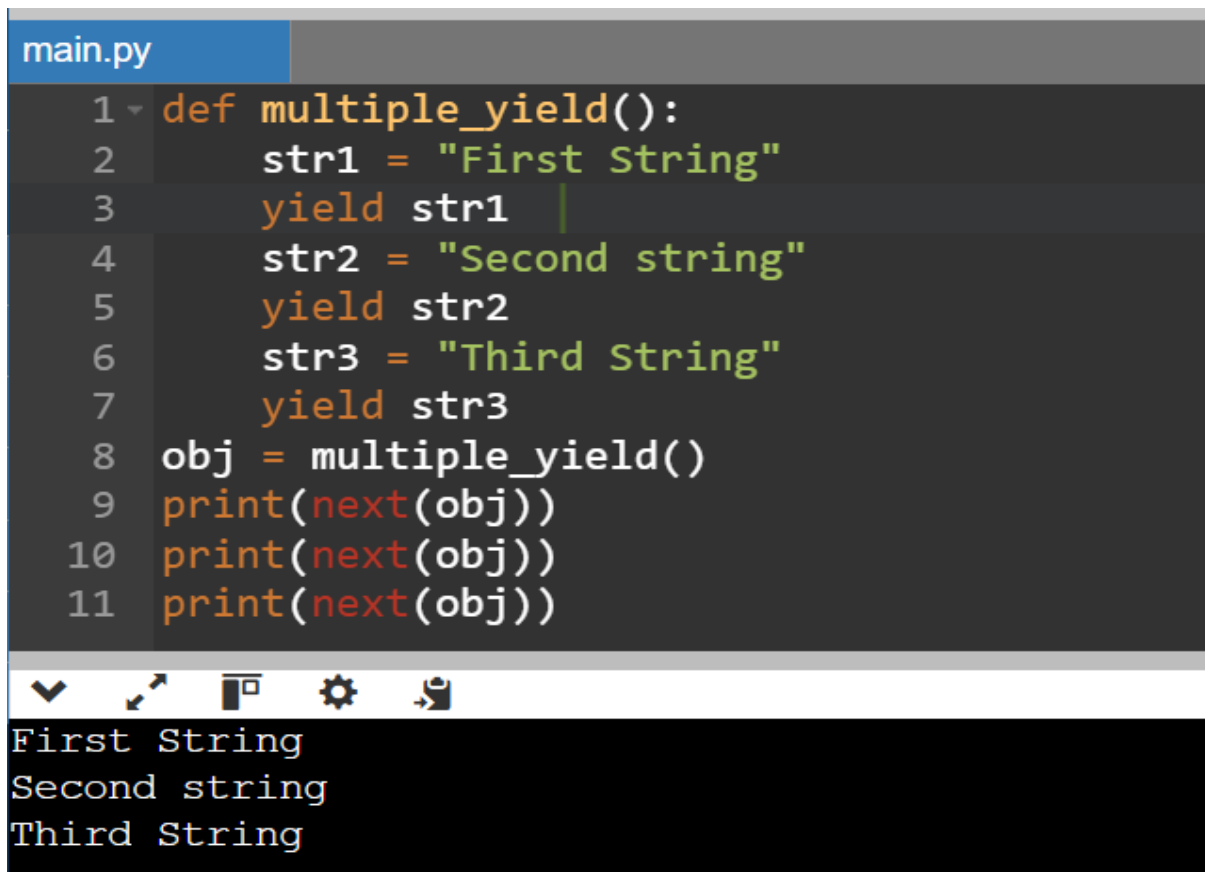


The screenshot shows a Python IDE with a toolbar at the top containing icons for file operations, a 'Run' button, 'Debug', 'Stop', 'Share', 'Save', and 'Beautify'. The editor window, titled 'main.py', contains the following code:

```
1 def simple():
2     for i in range(10):
3         if(i%2==0):
4             yield i
5 for i in simple():
6     print(i)
```

Below the editor is a toolbar with icons for a dropdown menu, zoom in, zoom out, settings, and a cursor icon. The output console at the bottom displays the numbers 0, 2, 4, 6, and 8, which are the even numbers yielded by the 'simple' generator function.

2. Using multiple Yield Statement



The screenshot shows a Python IDE with the same toolbar as the first image. The editor window, titled 'main.py', contains the following code:

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

Below the editor is a toolbar with icons for a dropdown menu, zoom in, zoom out, settings, and a cursor icon. The output console at the bottom displays the strings 'First String', 'Second string', and 'Third String', which are the values yielded by the 'multiple_yield' generator function.

3. Generator Expression

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

<generator object <genexpr> at 0x772aeb7bb9f0>
[1, 8, 27, 64, 125, 216, 343]

4. Multiplication table using Generators

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

15
30
45
60
75
90
105
120
135
150

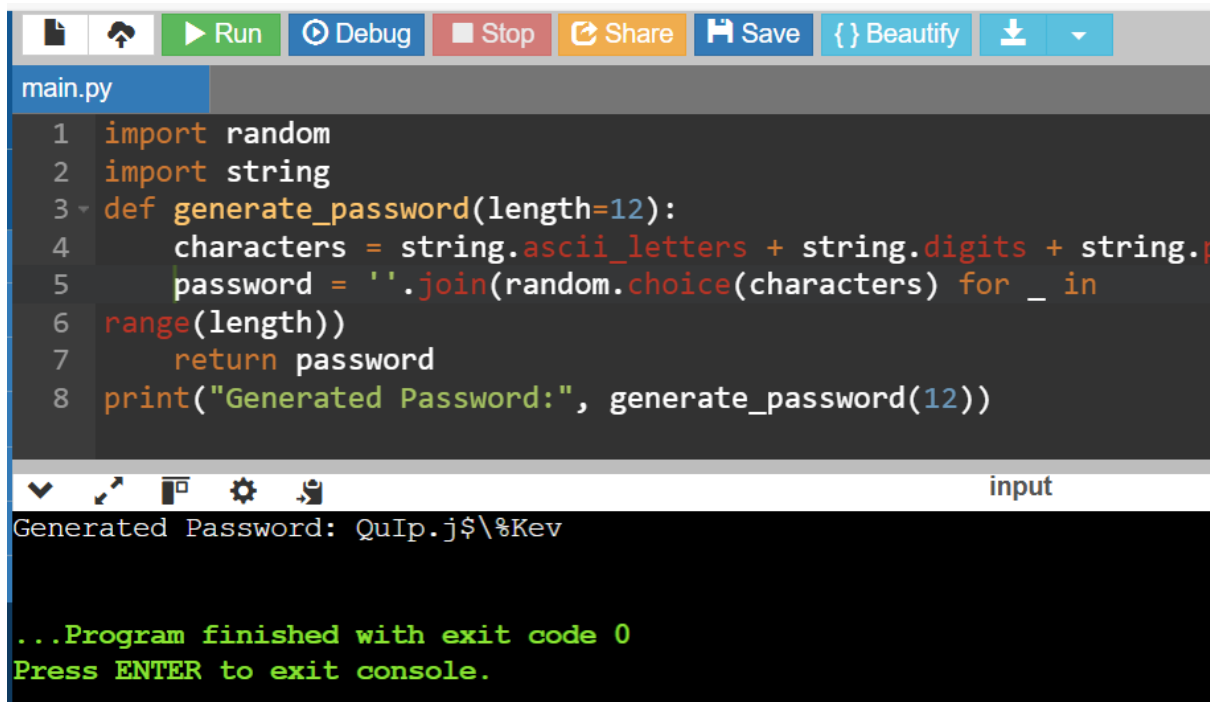
5. Using next () on Generator Object

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

1
8
27
64

PYTHON BASIC PROJECT

1. PASSWORD GENERATOR



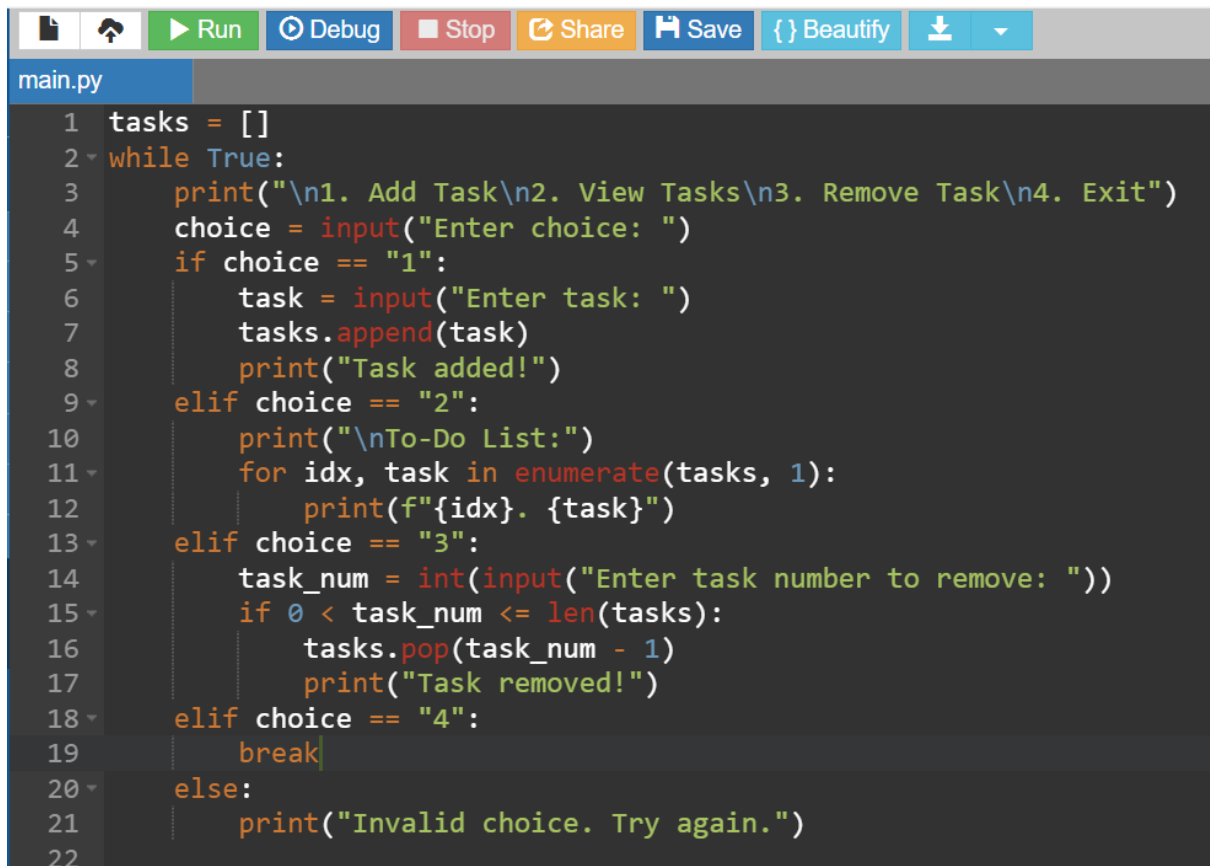
The screenshot shows a Python IDE with a toolbar at the top containing icons for file operations, a 'Run' button, 'Debug', 'Stop', 'Share', 'Save', 'Beautify', and a download icon. The file 'main.py' is open, displaying the following code:

```
1 import random
2 import string
3 def generate_password(length=12):
4     characters = string.ascii_letters + string.digits + string.punctuation
5     password = ''.join(random.choice(characters) for _ in
6 range(length))
7     return password
8 print("Generated Password:", generate_password(12))
```

Below the code editor, the console output is visible, showing the generated password and the program's completion message:

```
Generated Password: QuIp.j$%\%Kev
...Program finished with exit code 0
Press ENTER to exit console.
```

2. TO-DO LIST



The screenshot shows a Python IDE with the same toolbar as the first image. The file 'main.py' is open, displaying the following code for a to-do list application:

```
1 tasks = []
2 while True:
3     print("\n1. Add Task\n2. View Tasks\n3. Remove Task\n4. Exit")
4     choice = input("Enter choice: ")
5     if choice == "1":
6         task = input("Enter task: ")
7         tasks.append(task)
8         print("Task added!")
9     elif choice == "2":
10        print("\nTo-Do List:")
11        for idx, task in enumerate(tasks, 1):
12            print(f"{idx}. {task}")
13    elif choice == "3":
14        task_num = int(input("Enter task number to remove: "))
15        if 0 < task_num <= len(tasks):
16            tasks.pop(task_num - 1)
17            print("Task removed!")
18    elif choice == "4":
19        break
20    else:
21        print("Invalid choice. Try again.")
22
```

OUTPUT

```
input
1. Add Task
2. View Tasks
3. Remove Task
4. Exit
Enter choice: 1
Enter task: work
Task added!

1. Add Task
2. View Tasks
3. Remove Task
4. Exit
Enter choice: 1
Enter task: read
Task added!

1. Add Task
2. View Tasks
3. Remove Task
4. Exit
Enter choice: 1
Enter task: sleep
Task added!

1. Add Task
2. View Tasks
3. Remove Task
4. Exit
Enter choice: 2

To-Do List:
1. work
2. read
3. sleep

1. Add Task
2. View Tasks
3. Remove Task
4. Exit
```

3. WEATHER APP (API Based)

```
main.py
1 import requests
2 API_KEY = "8f2d6822fb2e4524adf20f8132e6f463"
3 city = input("Enter city name: ")
4 url = f"http://api.openweathermap.org/data/2.5/weather?q={city}&appid={API_KEY}&units=metric"
5 response = requests.get(url).json()
6 if response["cod"] == 200:
7     print(f"\nCity: {response['name']}")
8     print(f"Temperature: {response['main']['temp']}°C")
9     print(f"Weather: {response['weather'][0]['description']}")
10 else:
11     print("\nCity not found!")

input
Enter city name: hyderabad

City: Hyderabad
Temperature: 28.92°C
Weather: few clouds
```

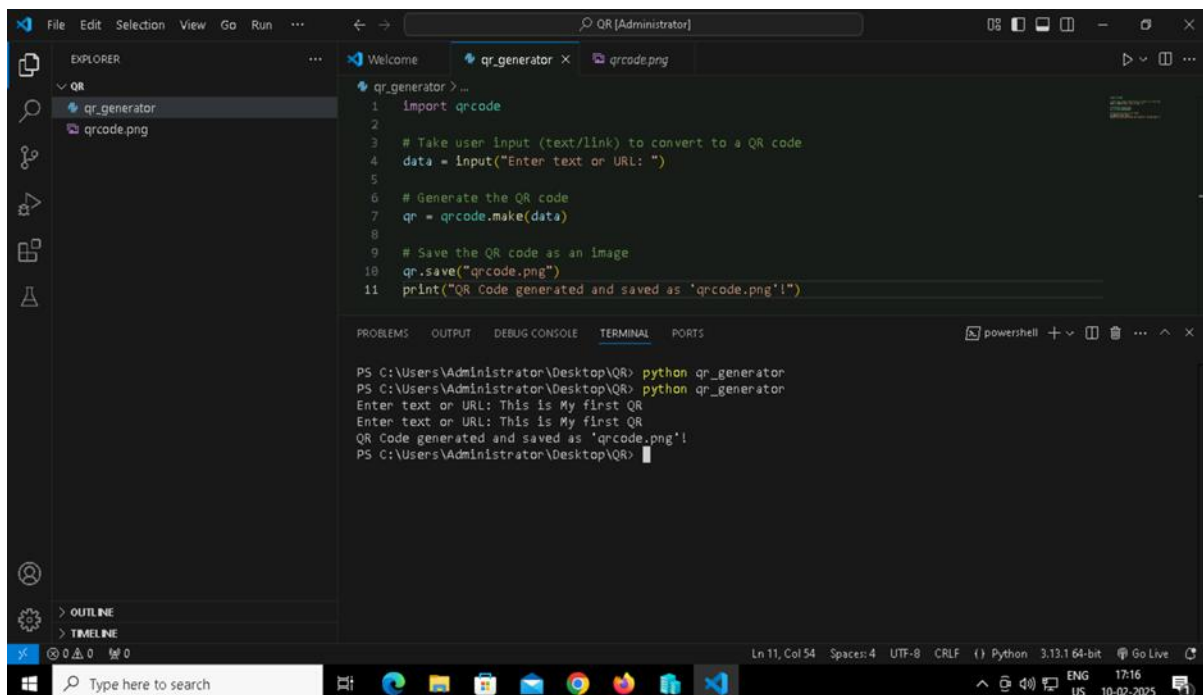
4. NUMBER GUESSING GAME

```
main.py
1 import random
2 number = random.randint(1, 100)
3 while True:
4     guess = int(input("Guess the number (1-100): "))
5     if guess < number:
6         print("Too low! Try again.")
7     elif guess > number:
8         print("Too high! Try again.")
9     else:
10        print("Congratulations! You guessed it right.")
11        break
```

input

Guess the number (1-100): 22
Too low! Try again.
Guess the number (1-100): 6
Too low! Try again.
Guess the number (1-100): 15
Too low! Try again.
Guess the number (1-100): 25
Too low! Try again.
Guess the number (1-100): 35
Congratulations! You guessed it right.

5. QR CODE GENERATOR



```
File Edit Selection View Go Run ... QR [Administrator]
EXPLORER
  QR
    qr_generator
    qrcode.png
  qr_generator > ...
1 import qrcode
2
3 # Take user input (text/link) to convert to a QR code
4 data = input("Enter text or URL: ")
5
6 # Generate the QR code
7 qr = qrcode.make(data)
8
9 # Save the QR code as an image
10 qr.save("qrcode.png")
11 print("QR Code generated and saved as 'qrcode.png'!")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\Administrator\Desktop\QR> python qr_generator
PS C:\Users\Administrator\Desktop\QR> python qr_generator
Enter text or URL: This is My first QR
Enter text or URL: This is My first QR
QR Code generated and saved as 'qrcode.png'!
PS C:\Users\Administrator\Desktop\QR>

Ln 11, Col 54 Spaces: 4 UTF-8 CRLF Python 3.13.1 64-bit Go Live

OUTPUT

