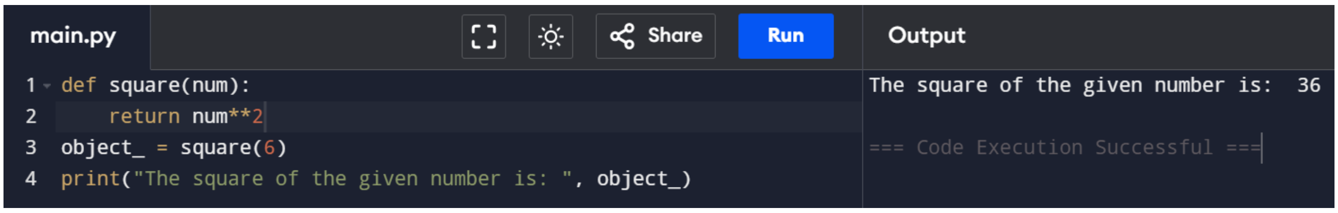
Python Project

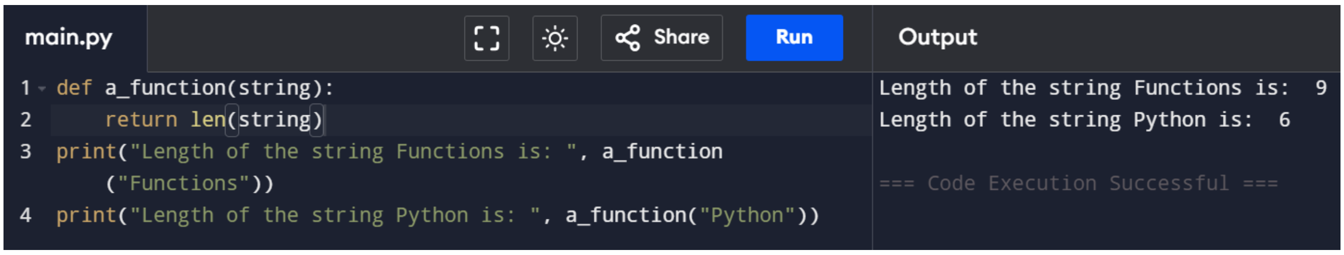
Python Code for User-Defined function



```python
def square(num):
    return num**2
object_ = square(6)
print("The square of the given number is: ", object_)
```

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

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

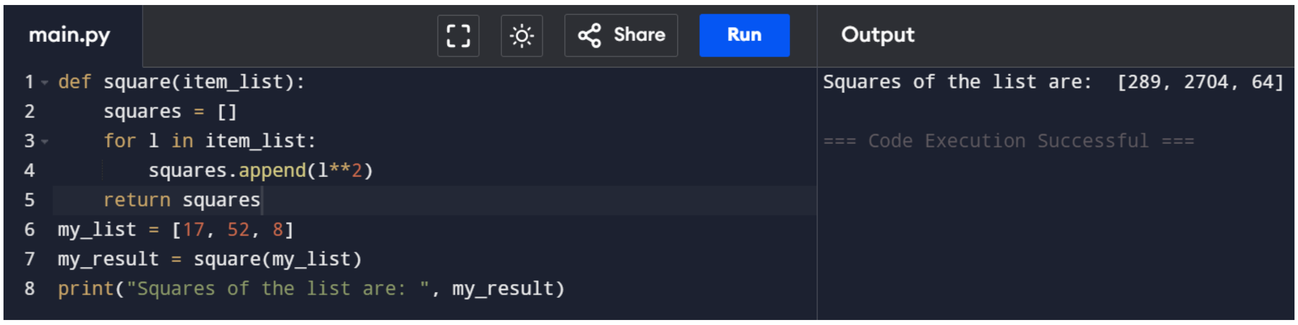
Python Code for calling a function



```python
def a_function(string):
    return len(string)
print("Length of the string Functions is: ", a_function
    ("Functions"))
print("Length of the string Python is: ", a_function("Python"))
```

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

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

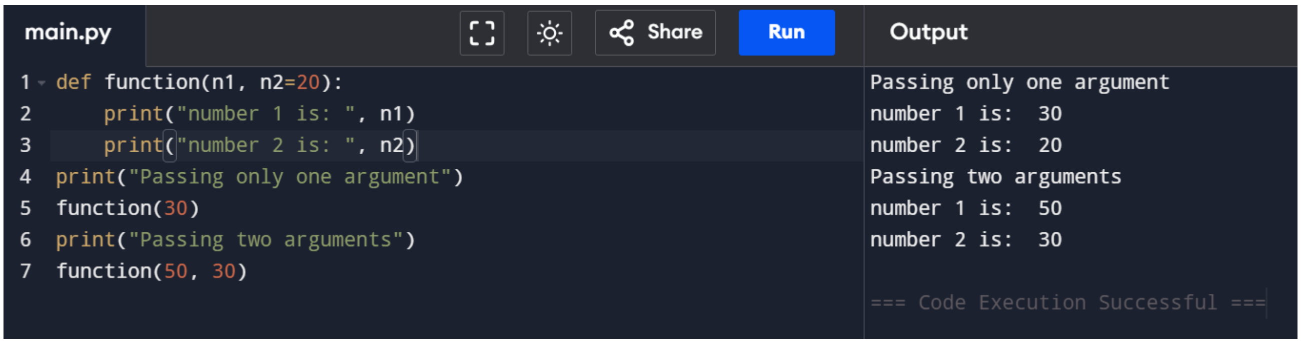
Python Code for Pass by Reference vs. Value



```python
def square(item_list):
    squares = []
    for l in item_list:
        squares.append(l**2)
    return squares
my_list = [17, 52, 8]
my_result = square(my_list)
print("Squares of the list are: ", my_result)
```

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

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

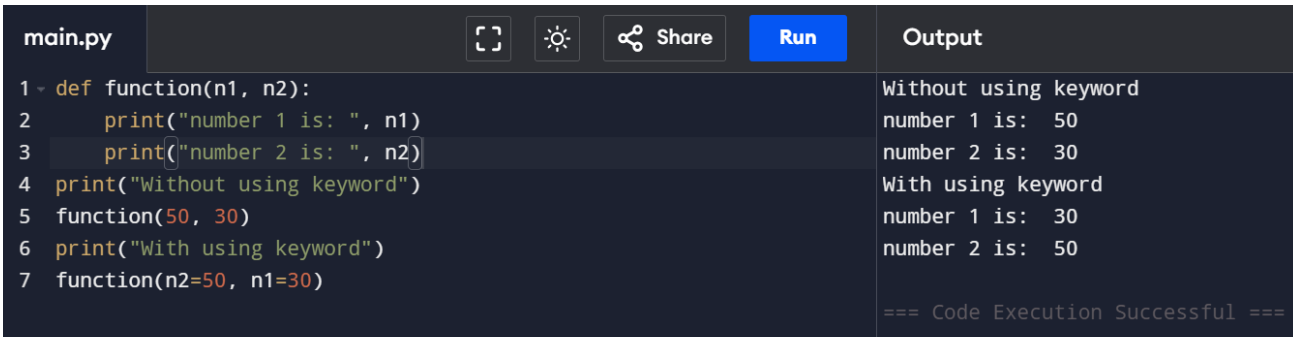
Python code to demonstrate the use of default arguments



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

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

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

Python code to demonstrate the use of keyword arguments

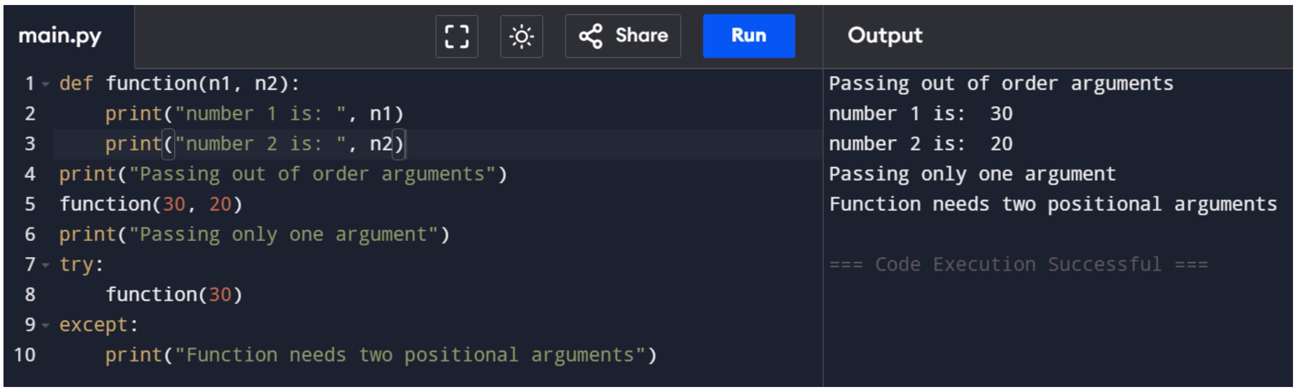


```python
def function(n1, n2):
    print("number 1 is: ", n1)
    print("number 2 is: ", n2)
print("Without using keyword")
function(50, 30)
print("With using keyword")
function(n2=50, n1=30)
```

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

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

Python Project

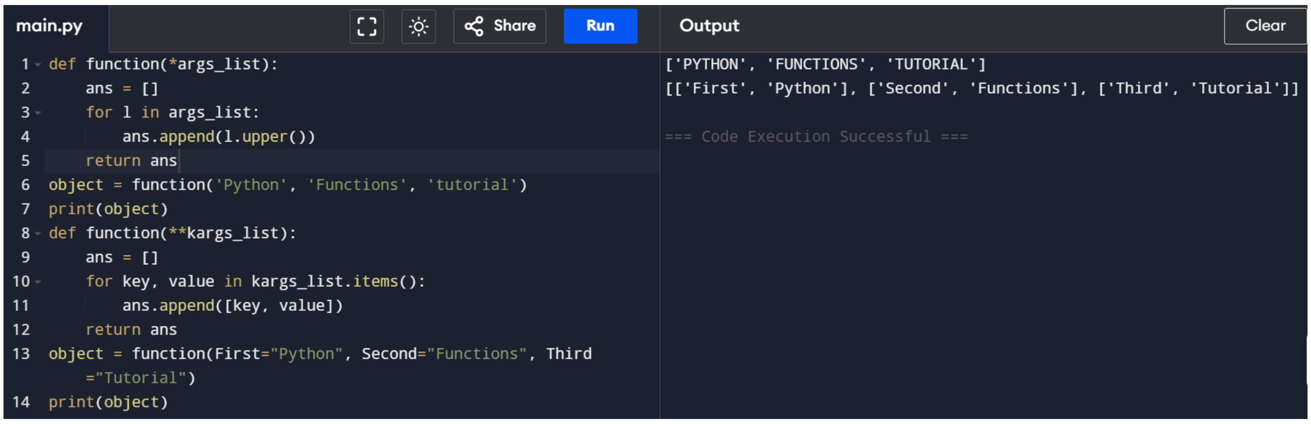
Python code to demonstrate the use of default arguments



```python
def function(n1, n2):
    print("number 1 is: ", n1)
    print("number 2 is: ", n2)
print("Passing out of order arguments")
function(30, 20)
print("Passing only one argument")
try:
    function(30)
except:
    print("Function needs two positional arguments")
```

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

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

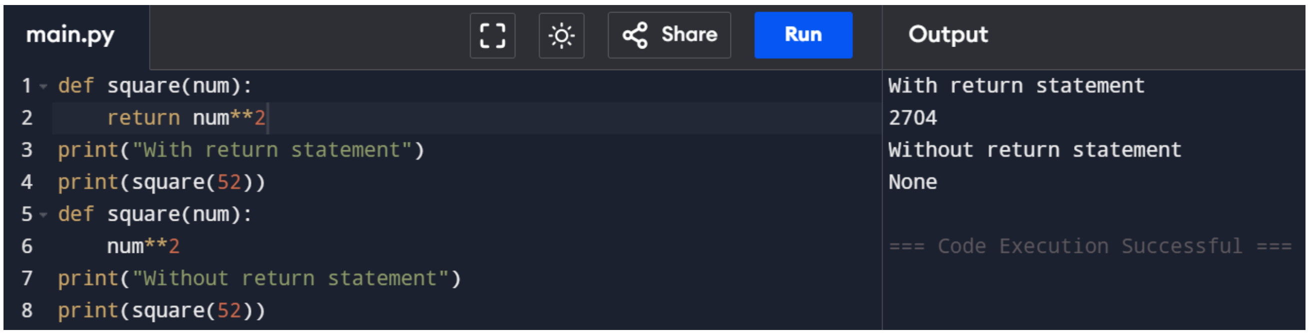
Python code to demonstrate the use of variable-length arguments



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

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

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

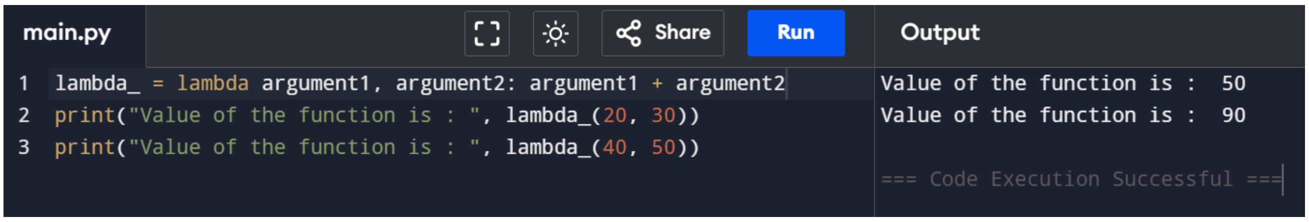
Python code to demonstrate the use of return statements



```python
def square(num):
    return num**2
print("With return statement")
print(square(52))
def square(num):
    num**2
print("Without return statement")
print(square(52))
```

```
With return statement
2704
Without return statement
None

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

Python code to demonstrate ananymous functions

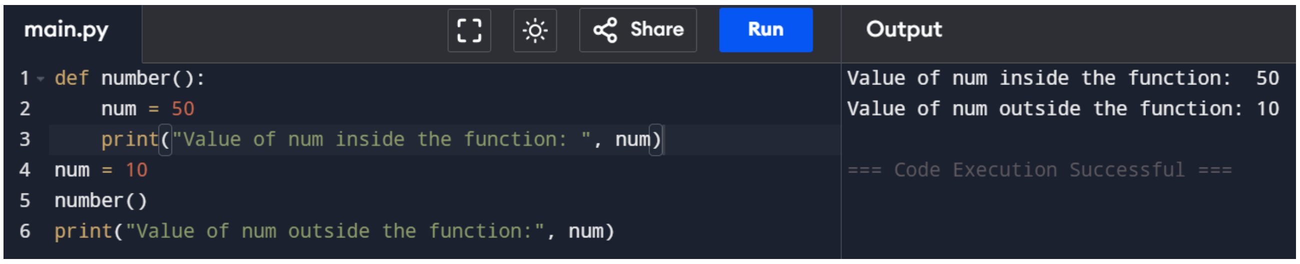


```python
lambda_ = lambda argument1, argument2: argument1 + argument2
print("Value of the function is : ", lambda_(20, 30))
print("Value of the function is : ", lambda_(40, 50))
```

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

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

Python Project

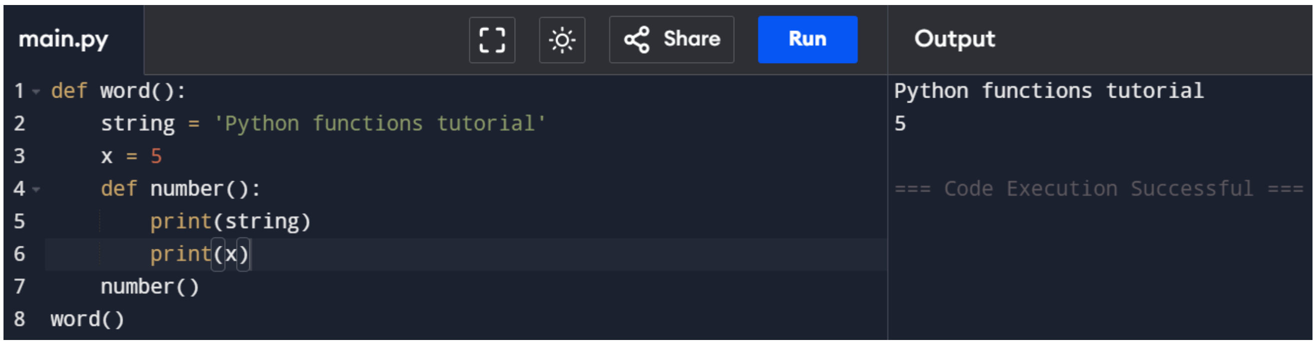
Python code to demonstrate scope and lifetime of variables



```python
def number():
    num = 50
    print("Value of num inside the function: ", num)
num = 10
number()
print("Value of num outside the function:", num)
```

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

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

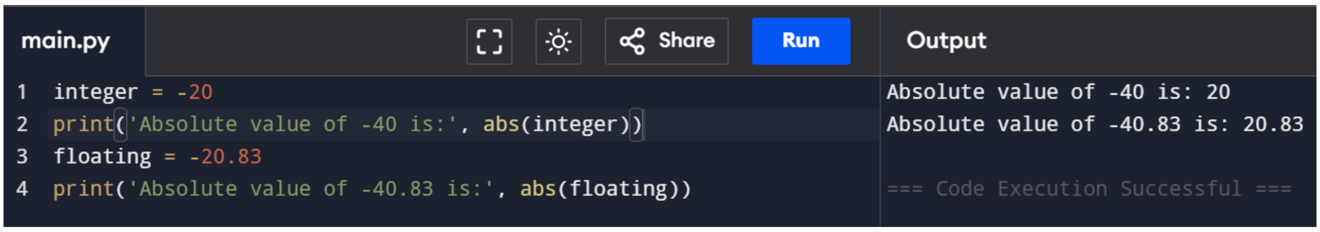
Python code to show how to access variables of a nested functions



```python
def word():
    string = 'Python functions tutorial'
    x = 5
    def number():
        print(string)
        print(x)
    number()
word()
```

```
Python functions tutorial
5

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

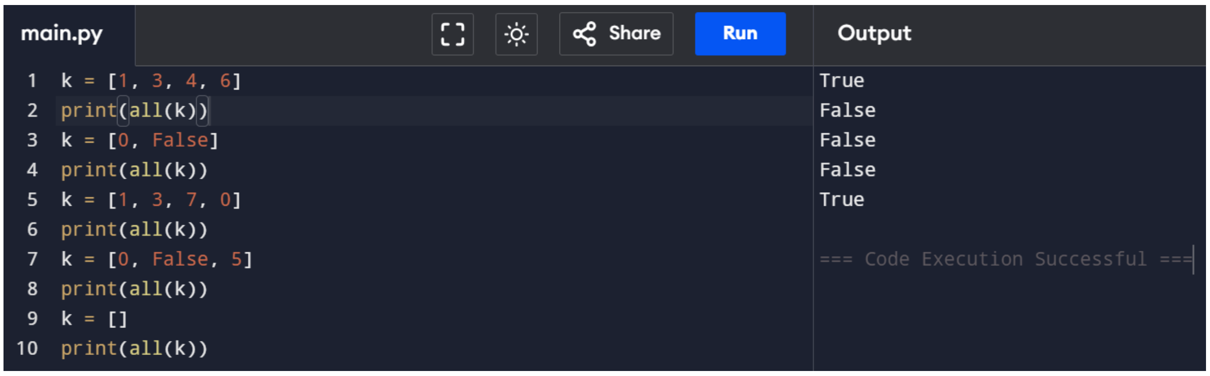
Python abs() Function



```python
integer = -20
print('Absolute value of -40 is:', abs(integer))
floating = -20.83
print('Absolute value of -40.83 is:', abs(floating))
```

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

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

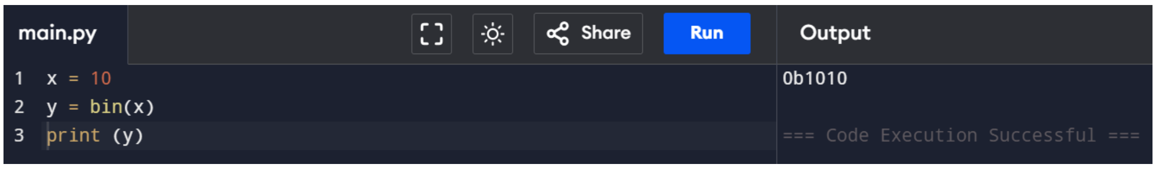
Python all() Function



```python
k = [1, 3, 4, 6]
print(all(k))
k = [0, False]
print(all(k))
k = [1, 3, 7, 0]
print(all(k))
k = [0, False, 5]
print(all(k))
k = []
print(all(k))
```

```
True
False
False
False
True

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

Python bin() Function

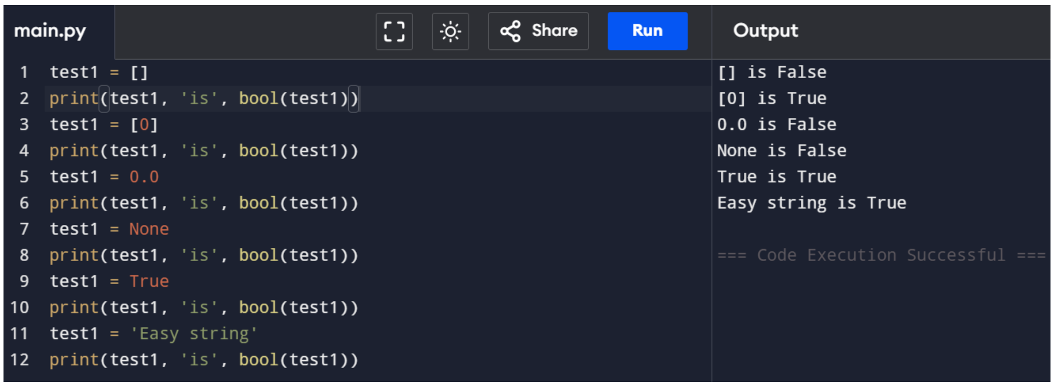


```python
x = 10
y = bin(x)
print (y)
```

```
0b1010

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

Python Project

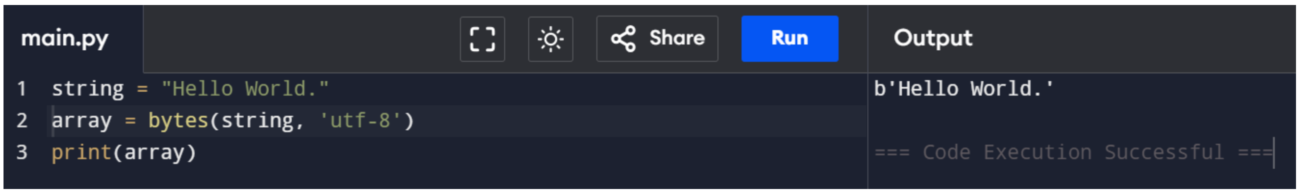
Python bool()



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

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

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

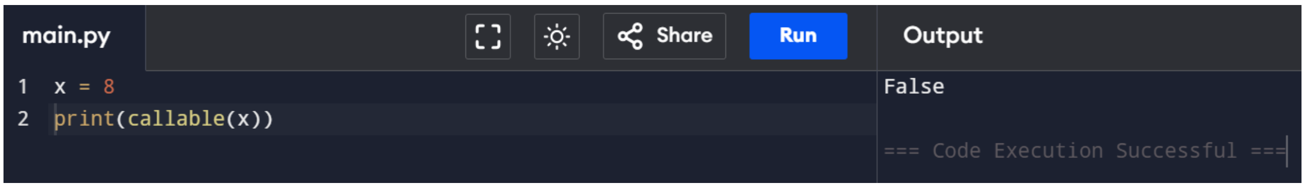
Python bytes()



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

```
b'Hello World.'

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

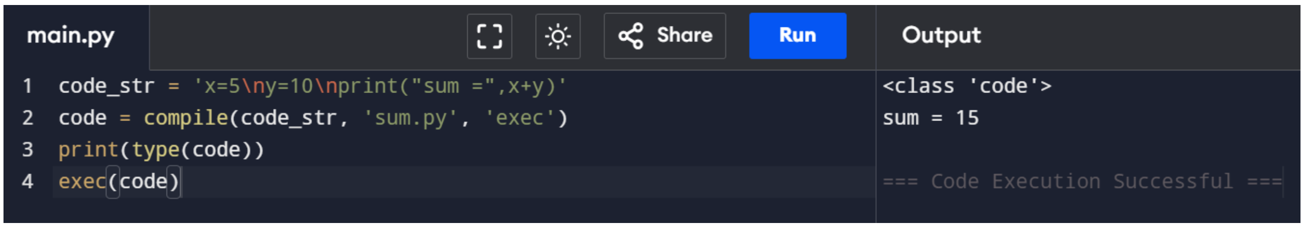
Python callable()



```python
x = 8
print(callable(x))
```

```
False

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

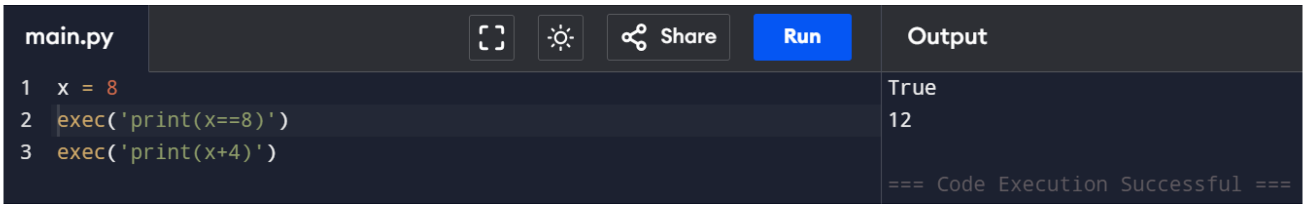
Python compile()



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

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

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

Python exec()

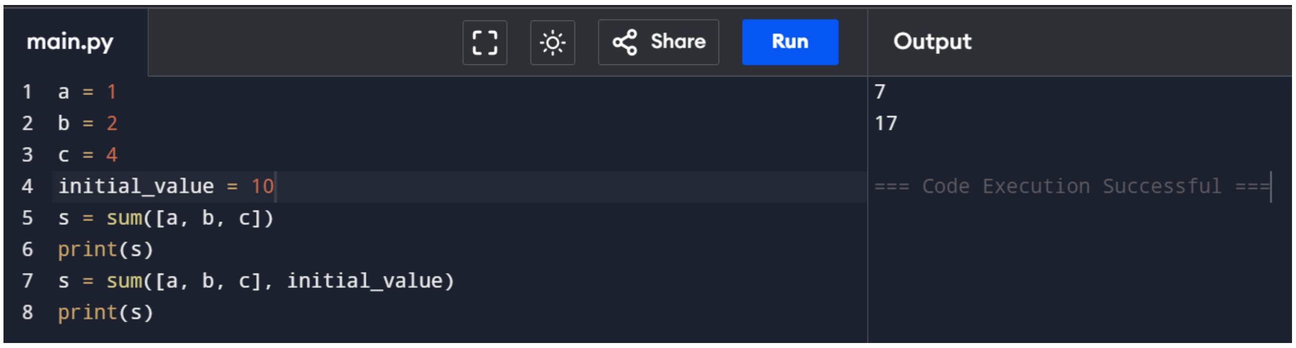


```python
x = 8
exec('print(x==8)')
exec('print(x+4)')
```

```
True
12

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

Python Project

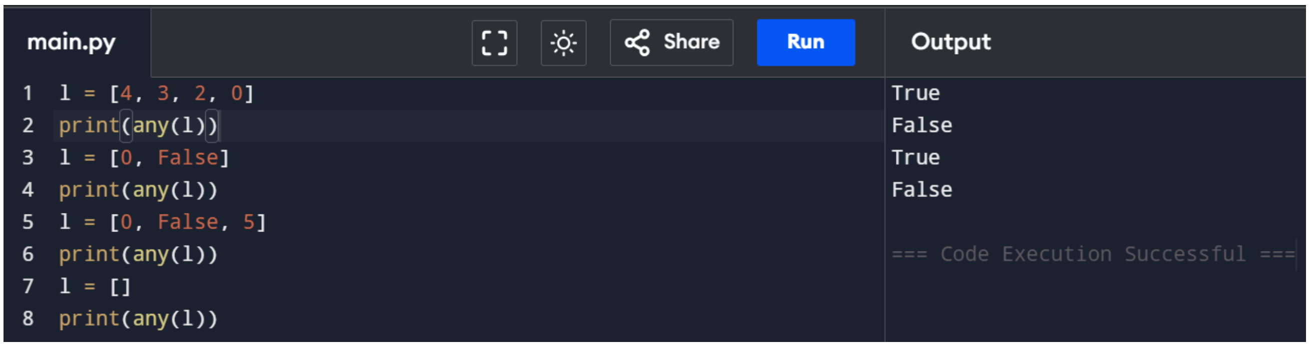
Python sum()



```python
a = 1
b = 2
c = 4
initial_value = 10
s = sum([a, b, c])
print(s)
s = sum([a, b, c], initial_value)
print(s)
```

```
7
17

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

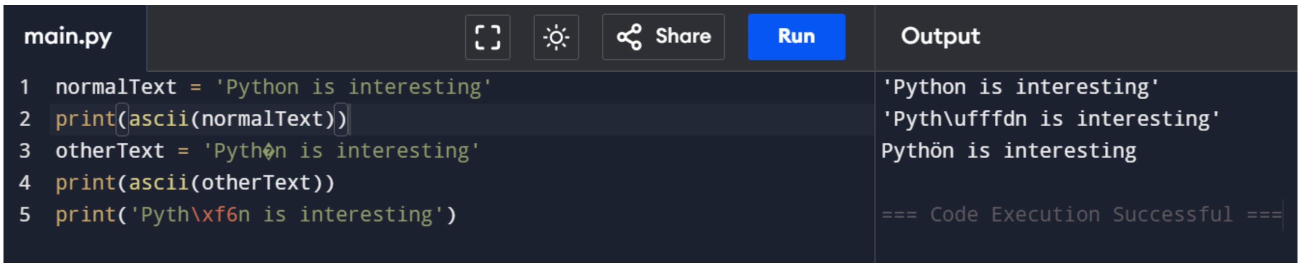
Python any() Function



```python
l = [4, 3, 2, 0]
print(any(l))
l = [0, False]
print(any(l))
l = [0, False, 5]
print(any(l))
l = []
print(any(l))
```

```
True
False
True
False

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

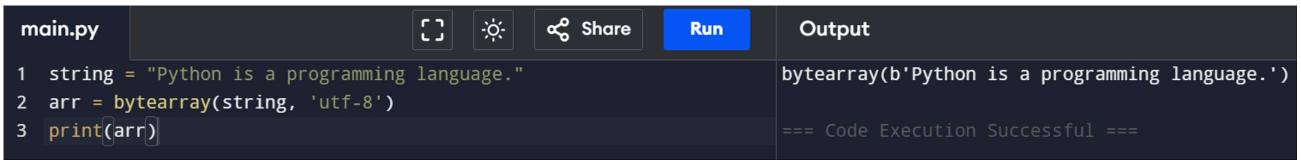
Python ascii() Function



```python
normalText = 'Python is interesting'
print(ascii(normalText))
otherText = 'Pyth�n is interesting'
print(ascii(otherText))
print('Pyth\xf6n is interesting')
```

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

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

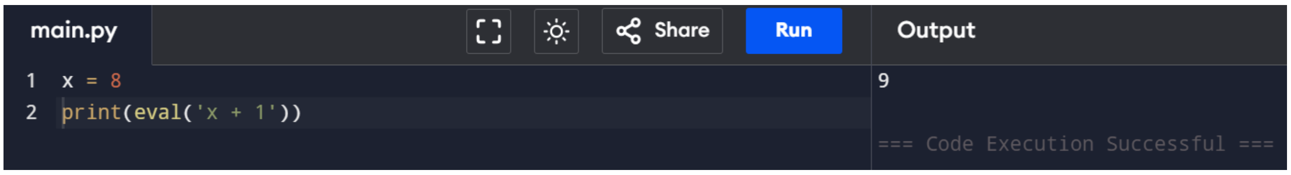
Python bytearray()



```python
string = "Python is a programming language."
arr = bytearray(string, 'utf-8')
print(arr)
```

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

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

Python eval()

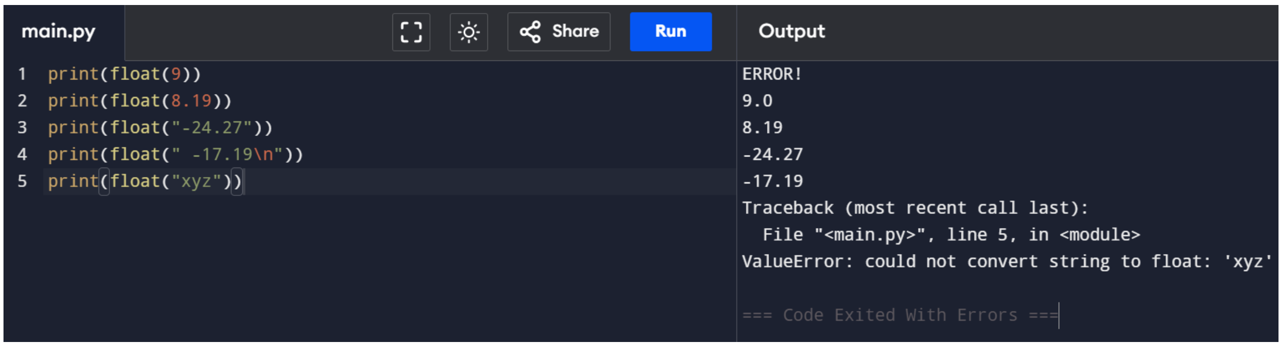


```python
x = 8
print(eval('x + 1'))
```

```
9

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

Python Project

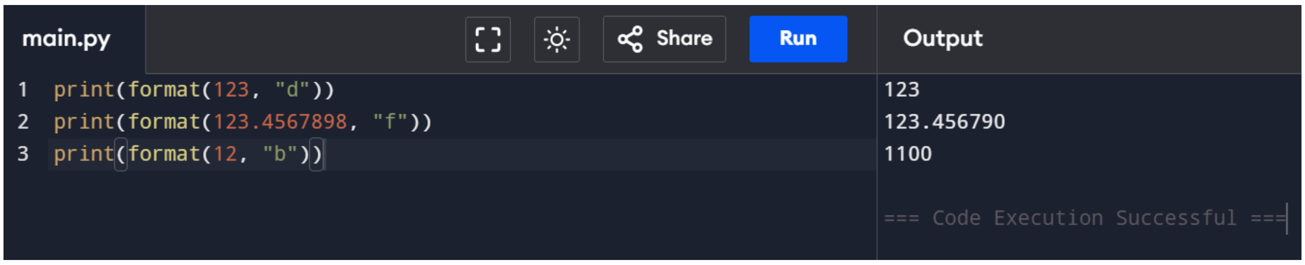
Python float()



```python
print(float(9))
print(float(8.19))
print(float("-24.27"))
print(float(" -17.19\n"))
print(float("xyz"))
```

```
ERROR!
9.0
8.19
-24.27
-17.19
Traceback (most recent call last):
  File "<main.py>", line 5, in <module>
ValueError: could not convert string to float: 'xyz'

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

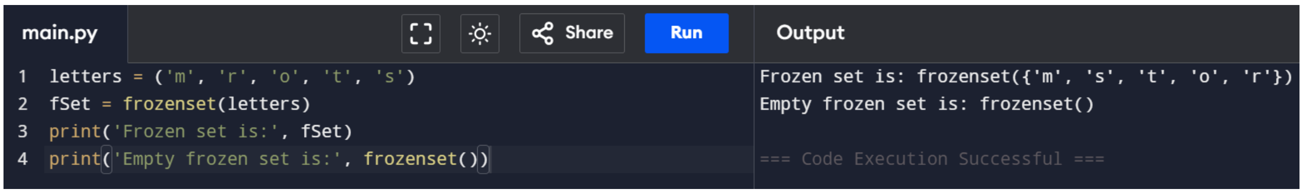
Python format() Function



```python
print(format(123, "d"))
print(format(123.4567898, "f"))
print(format(12, "b"))
```

```
123
123.456790
1100

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

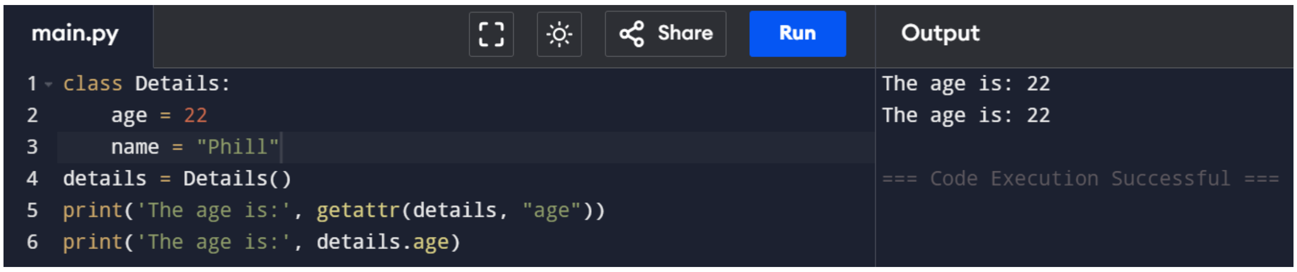
Python frozenset()



```python
letters = ('m', 'r', 'o', 't', 's')
fSet = frozenset(letters)
print('Frozen set is:', fSet)
print('Empty frozen set is:', frozenset())
```

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

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

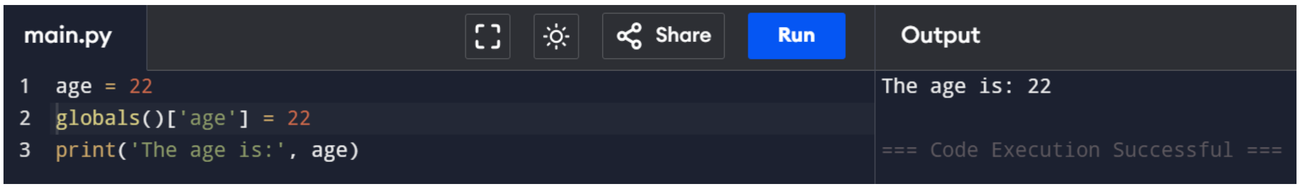
Python getattr() Function



```python
class Details:
    age = 22
    name = "Phill"
details = Details()
print('The age is:', getattr(details, "age"))
print('The age is:', details.age)
```

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

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

Python globals() Function

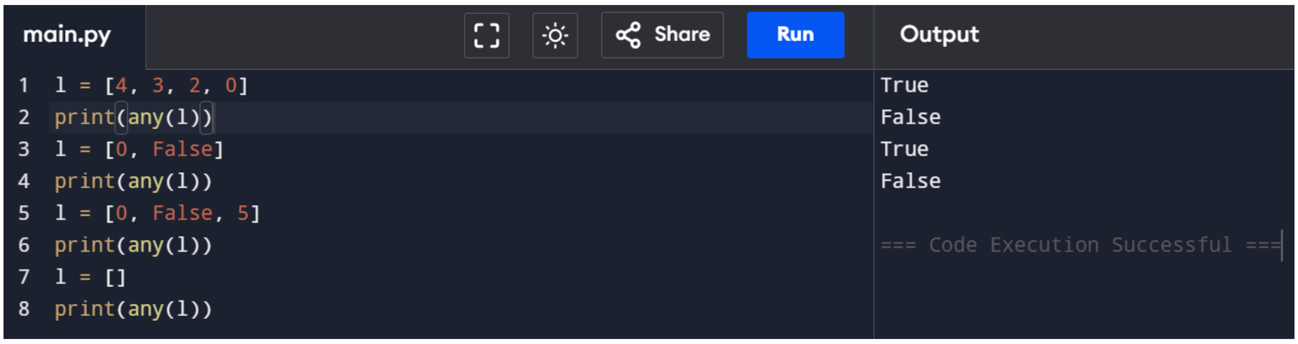


```python
age = 22
globals()['age'] = 22
print('The age is:', age)
```

```
The age is: 22

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

Python Project

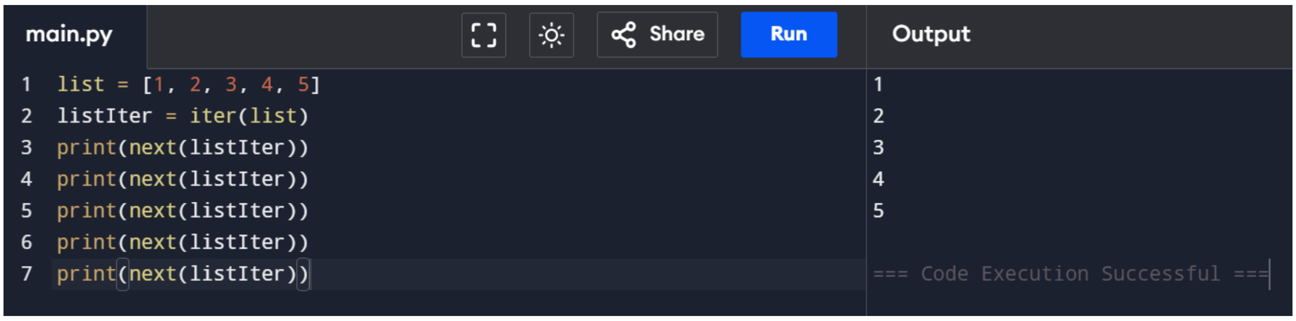
Python hasattr() Function



```python
l = [4, 3, 2, 0]
print(any(l))
l = [0, False]
print(any(l))
l = [0, False, 5]
print(any(l))
l = []
print(any(l))
```

```
True
False
True
False

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

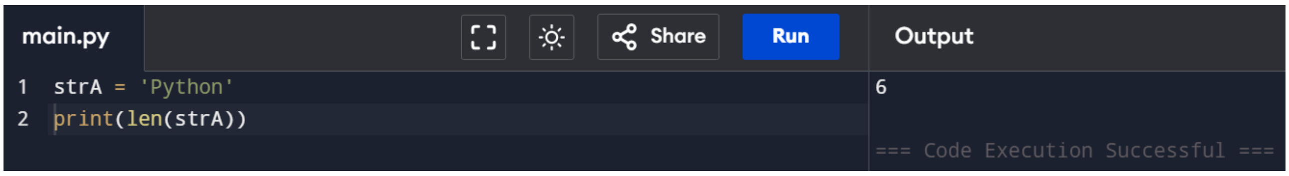
Python iter() Function



```python
list = [1, 2, 3, 4, 5]
listIter = iter(list)
print(next(listIter))
print(next(listIter))
print(next(listIter))
print(next(listIter))
print(next(listIter))
```

```
1
2
3
4
5

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

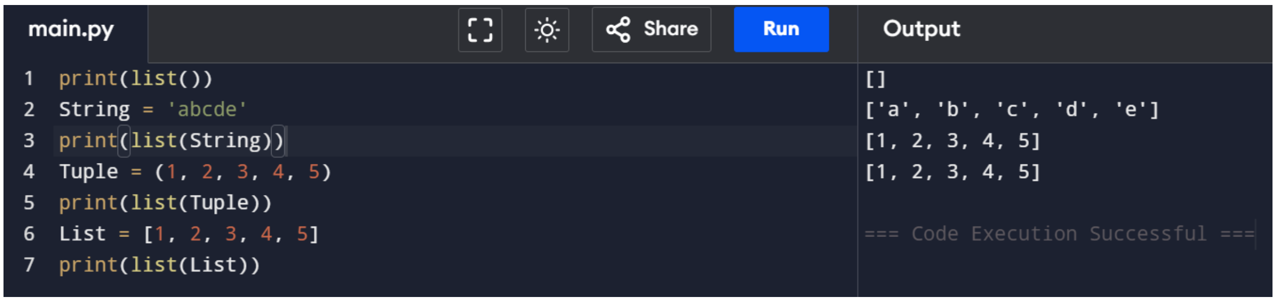
Python len() Function



```python
strA = 'Python'
print(len(strA))
```

```
6

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

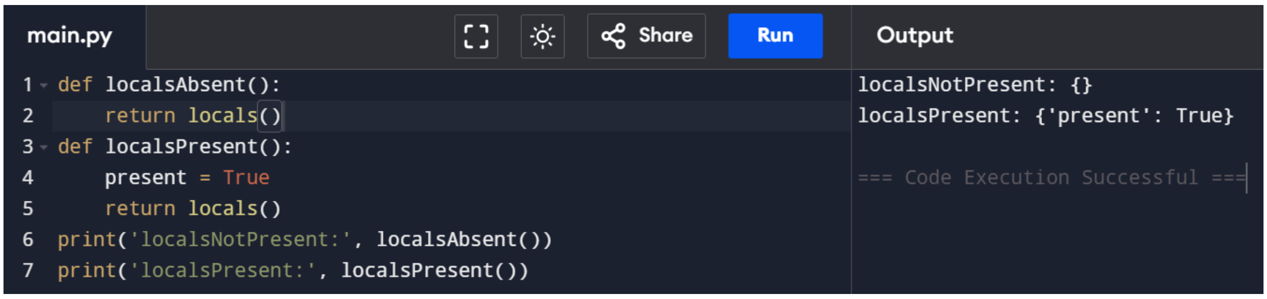
Python list()



```python
print(list())
String = 'abcde'
print(list(String))
Tuple = (1, 2, 3, 4, 5)
print(list(Tuple))
List = [1, 2, 3, 4, 5]
print(list(List))
```

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

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

Python locals() Function

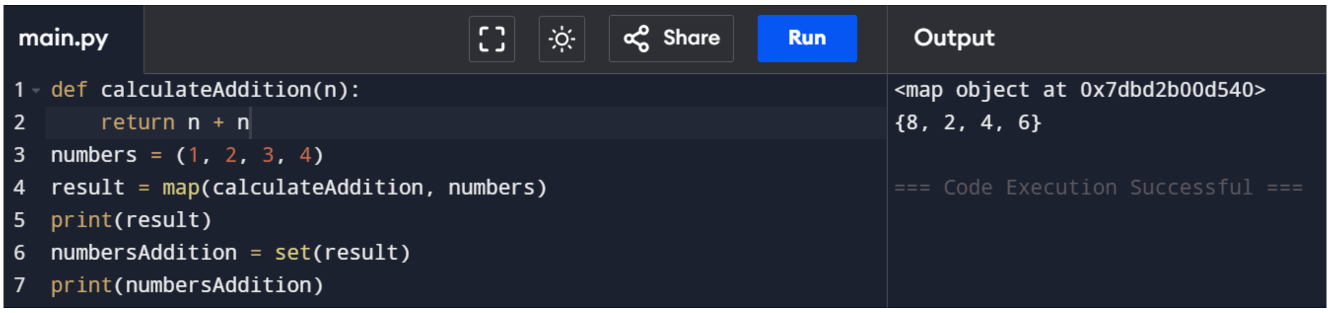


```python
def localsAbsent():
    return locals()
def localsPresent():
    present = True
    return locals()
print('localsNotPresent:', localsAbsent())
print('localsPresent:', localsPresent())
```

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

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

Python Project

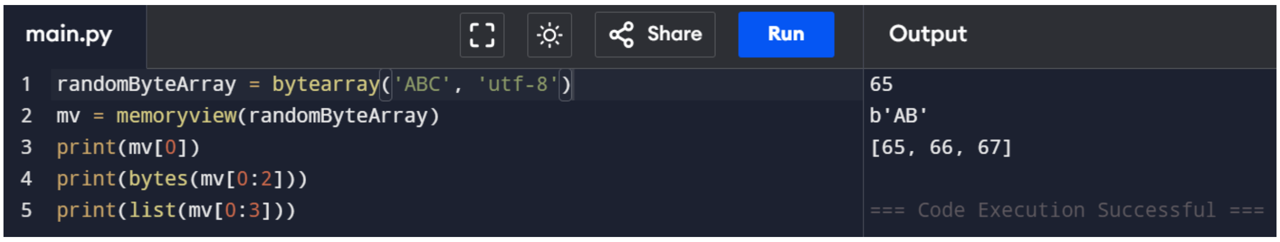
Python map() Function



```python
def calculateAddition(n):
    return n + n
numbers = (1, 2, 3, 4)
result = map(calculateAddition, numbers)
print(result)
numbersAddition = set(result)
print(numbersAddition)
```

```
<map object at 0x7dbd2b00d540>
{8, 2, 4, 6}

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

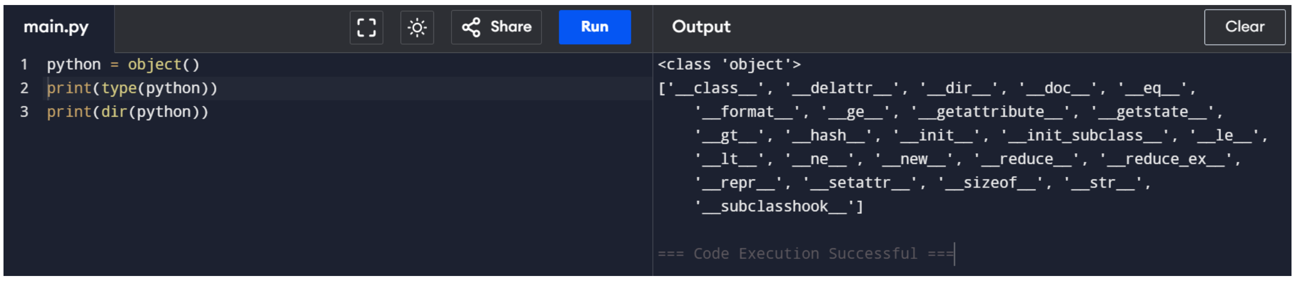
Python memoryview () Function



```python
randomByteArray = bytearray('ABC', 'utf-8')
mv = memoryview(randomByteArray)
print(mv[0])
print(bytes(mv[0:2]))
print(list(mv[0:3]))
```

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

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

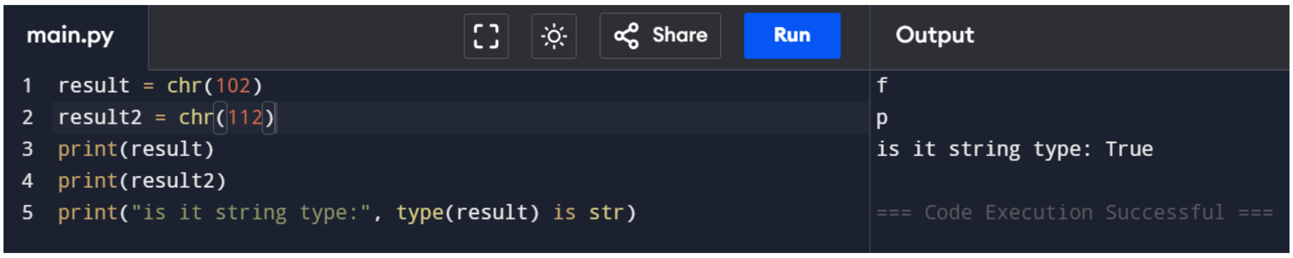
Python object()



```python
python = object()
print(type(python))
print(dir(python))
```

```
<class 'object'>
['__class__', '__delattr__', '__dir__', '__doc__', '__eq__',
    '__format__', '__ge__', '__getattribute__', '__getstate__',
    '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__',
    '__lt__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
    '__repr__', '__setattr__', '__sizeof__', '__str__',
    '__subclasshook__']

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

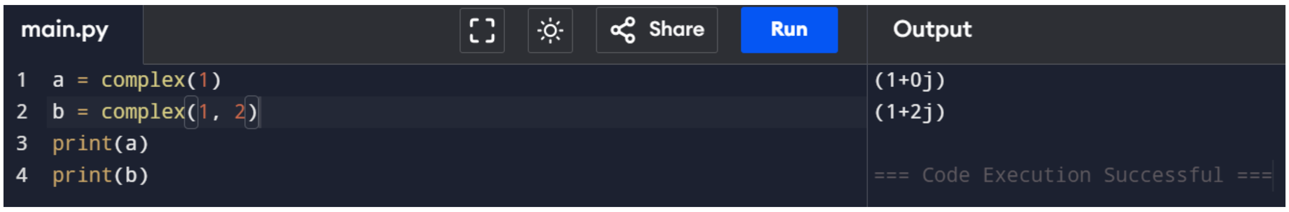
Python chr() Function



```python
result = chr(102)
result2 = chr(112)
print(result)
print(result2)
print("is it string type:", type(result) is str)
```

```
f
p
is it string type: True

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

Python complex() function

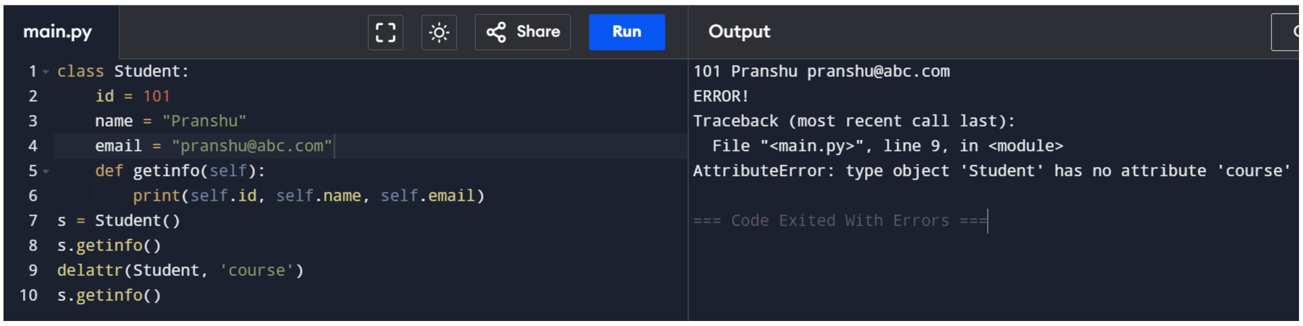


```python
a = complex(1)
b = complex(1, 2)
print(a)
print(b)
```

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

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

Python Project

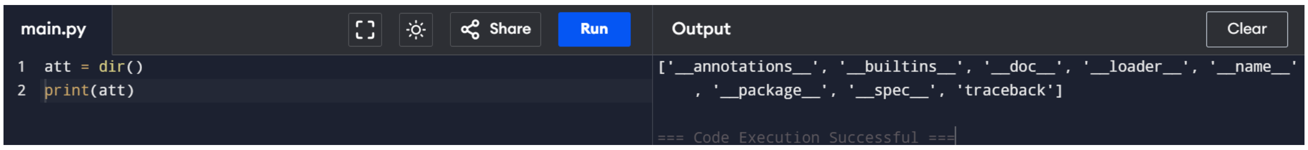
Python delattr() Function



```python
class Student:
    id = 101
    name = "Pranshu"
    email = "pranshu@abc.com"
    def getinfo(self):
        print(self.id, self.name, self.email)
s = Student()
s.getinfo()
delattr(Student, 'course')
s.getinfo()
```

```
101 Pranshu pranshu@abc.com
ERROR!
Traceback (most recent call last):
  File "<main.py>", line 9, in <module>
AttributeError: type object 'Student' has no attribute 'course'

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

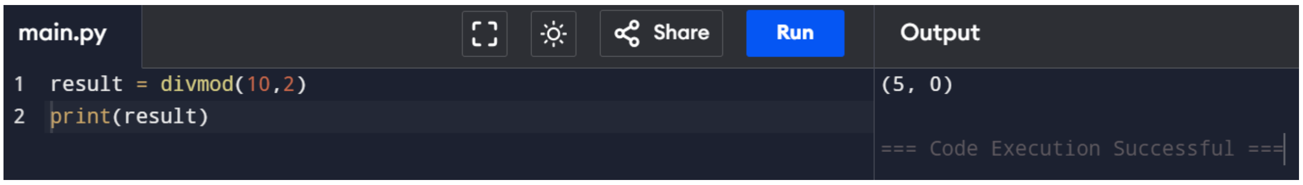
Python dir() Function



```python
att = dir()
print(att)
```

```
['__annotations__', '__builtins__', '__doc__', '__loader__', '__name__'
 , '__package__', '__spec__', 'traceback']

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

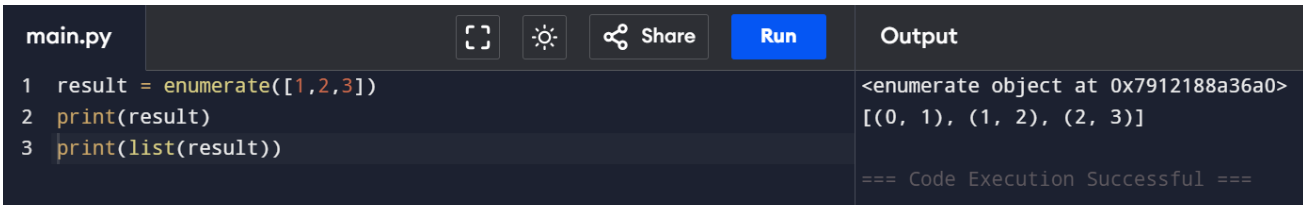
Python divmod() Function



```python
result = divmod(10,2)
print(result)
```

```
(5, 0)

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

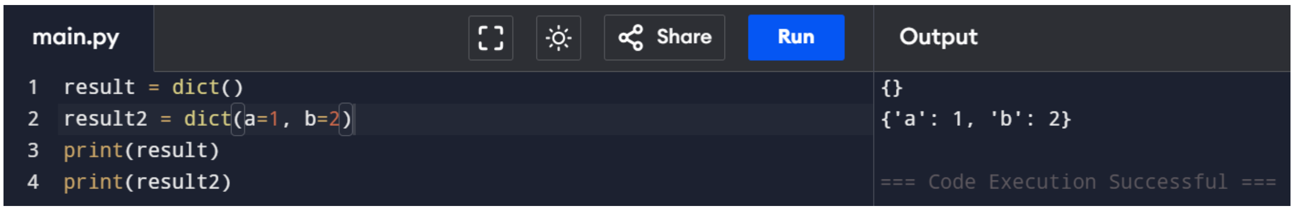
Python enumerate() Function



```python
result = enumerate([1,2,3])
print(result)
print(list(result))
```

```
<enumerate object at 0x7912188a36a0>
[(0, 1), (1, 2), (2, 3)]

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

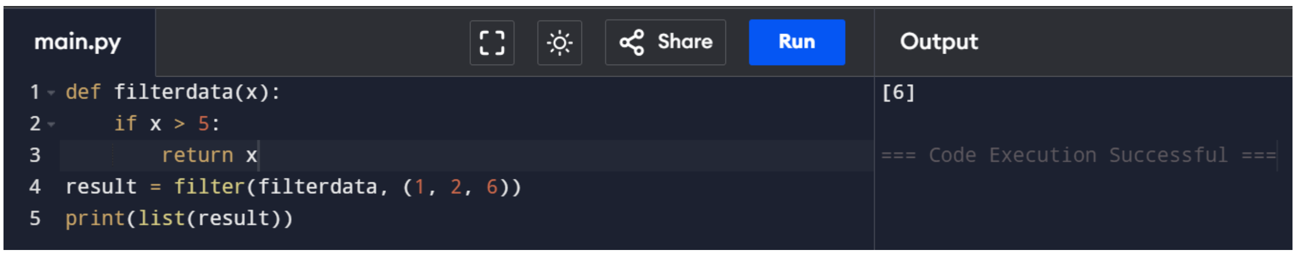
Python dict()



```python
result = dict()
result2 = dict(a=1, b=2)
print(result)
print(result2)
```

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

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

Python filter() Function

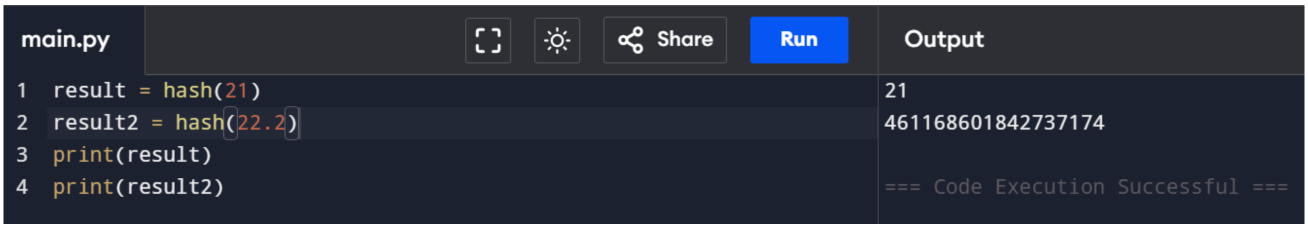


```python
def filterdata(x):
    if x > 5:
        return x
result = filter(filterdata, (1, 2, 6))
print(list(result))
```

```
[6]

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

Python Project

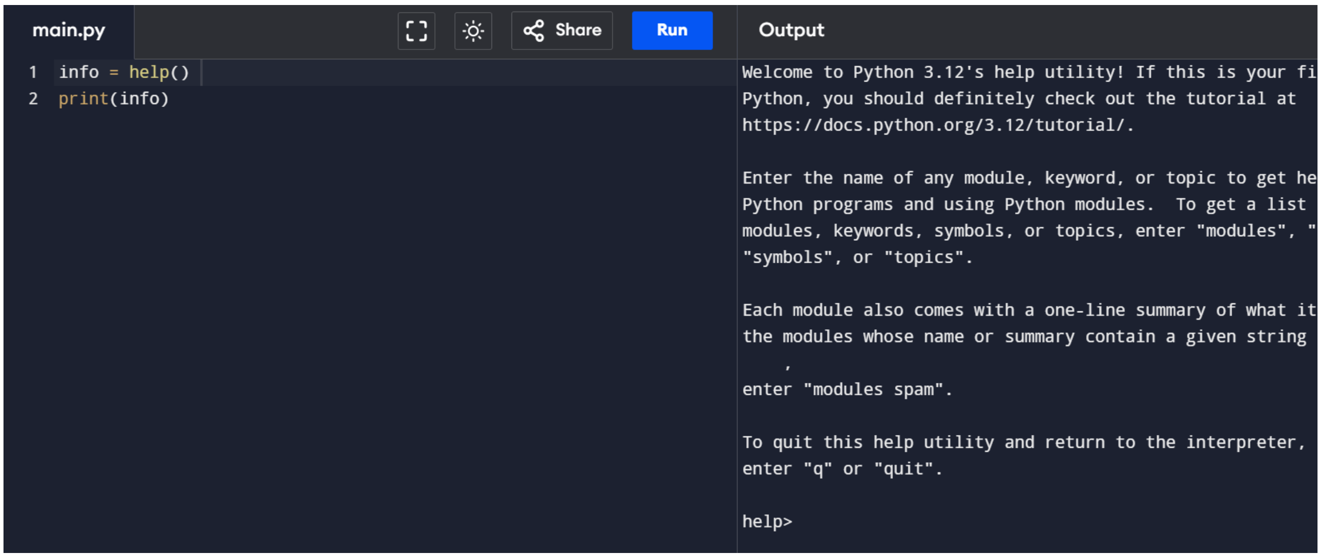
Python hash() Function



```python
result = hash(21)
result2 = hash(22.2)
print(result)
print(result2)
```

```
21
461168601842737174

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

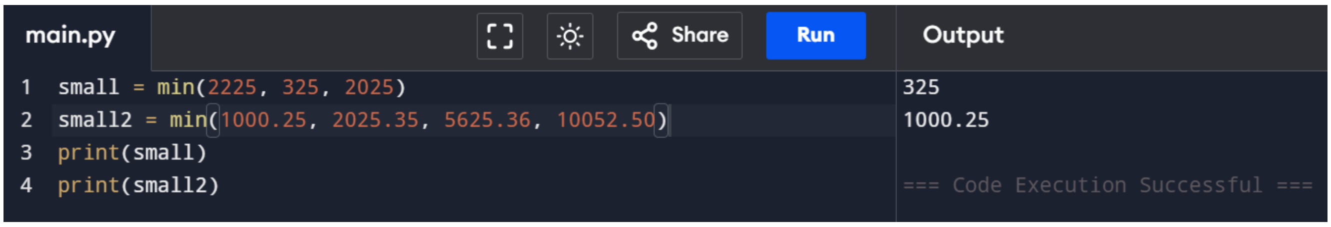
Python help() Function



```python
info = help()
print(info)
```

```
Welcome to Python 3.12's help utility! If this is your fi
Python, you should definitely check out the tutorial at
https://docs.python.org/3.12/tutorial/.

Enter the name of any module, keyword, or topic to get he
Python programs and using Python modules.  To get a list
modules, keywords, symbols, or topics, enter "modules", "
"symbols", or "topics".

Each module also comes with a one-line summary of what it
the modules whose name or summary contain a given string
    ,
enter "modules spam".

To quit this help utility and return to the interpreter,
enter "q" or "quit".

help>
```

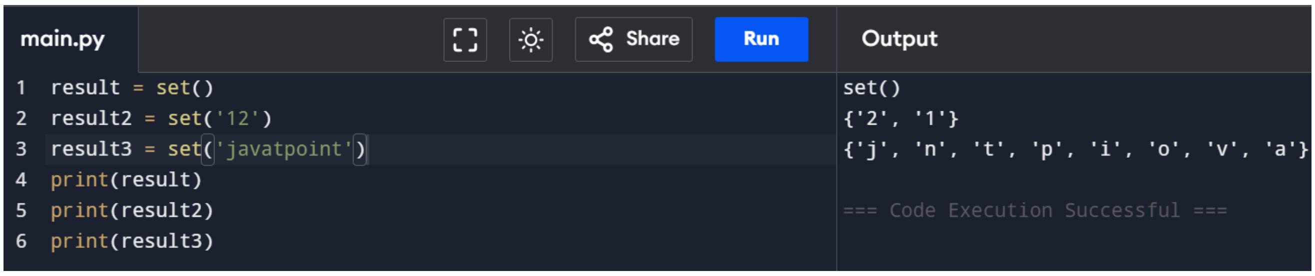
Python min() Function



```python
small = min(2225, 325, 2025)
small2 = min(1000.25, 2025.35, 5625.36, 10052.50)
print(small)
print(small2)
```

```
325
1000.25

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

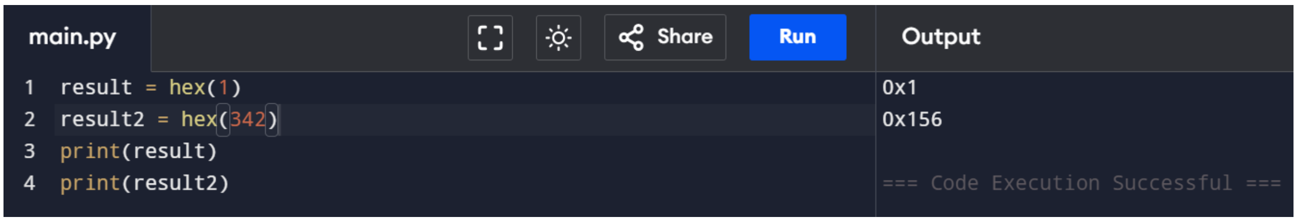
Python set() Function



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

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

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

Python hex() Function

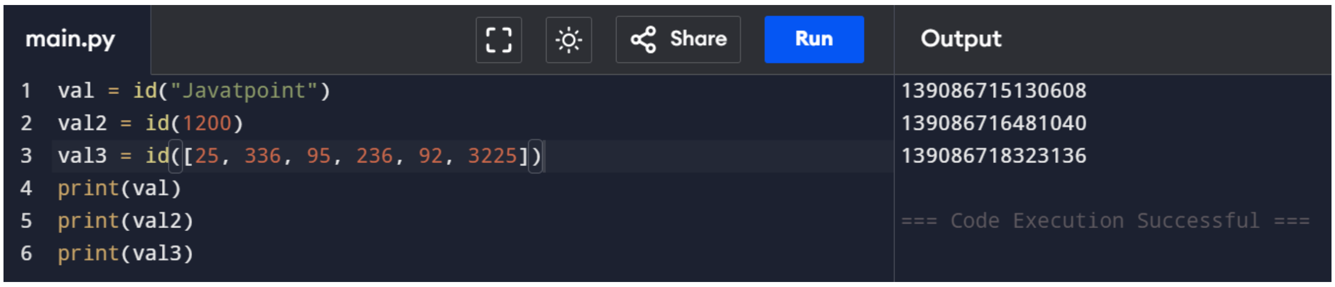


```python
result = hex(1)
result2 = hex(342)
print(result)
print(result2)
```

```
0x1
0x156

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

Python Project

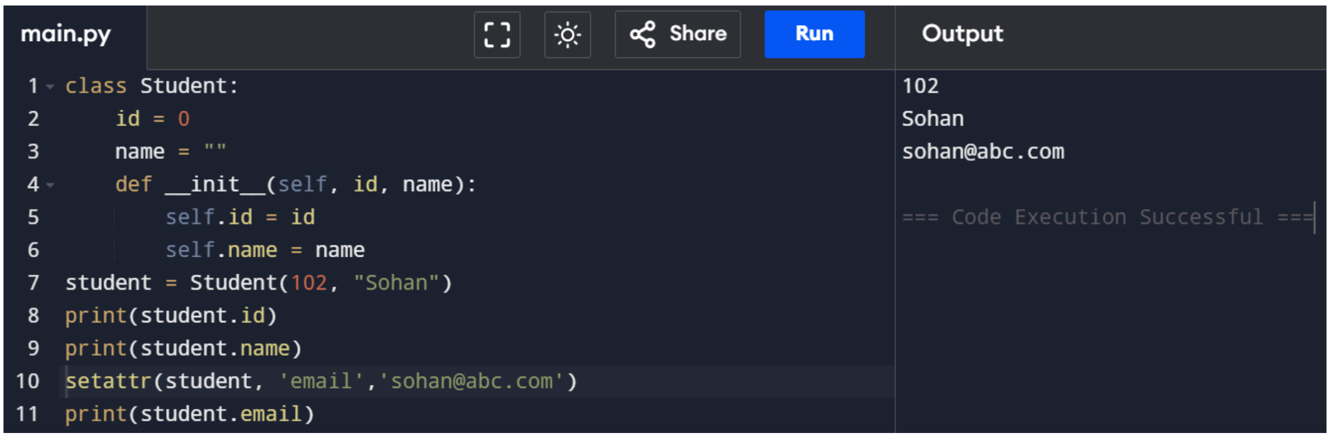
Python id() Function



```
val = id("Javatpoint")
val2 = id(1200)
val3 = id([25, 336, 95, 236, 92, 3225])
print(val)
print(val2)
print(val3)
```

```
139086715130608
139086716481040
139086718323136

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

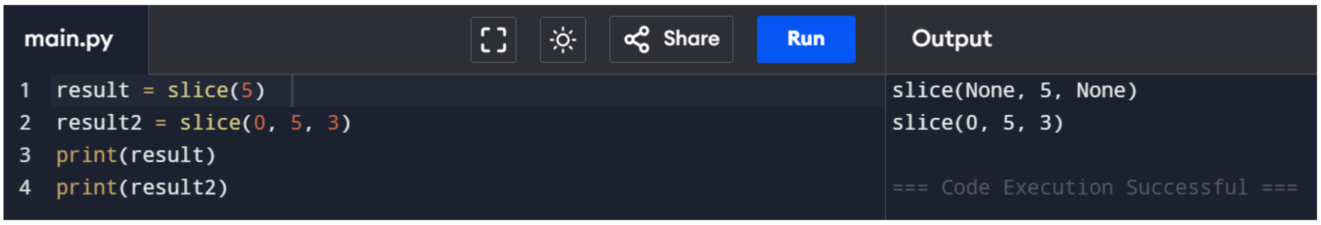
Python setattr() Function



```
class Student:
    id = 0
    name = ""
    def __init__(self, id, name):
        self.id = id
        self.name = name
student = Student(102, "Sohan")
print(student.id)
print(student.name)
setattr(student, 'email','sohan@abc.com')
print(student.email)
```

```
102
Sohan
sohan@abc.com

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

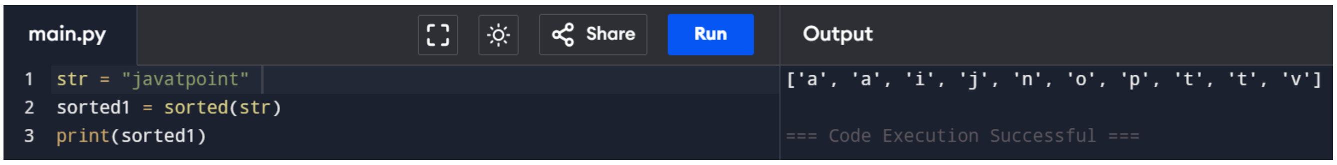
Python slice() Function



```
result = slice(5)
result2 = slice(0, 5, 3)
print(result)
print(result2)
```

```
slice(None, 5, None)
slice(0, 5, 3)

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

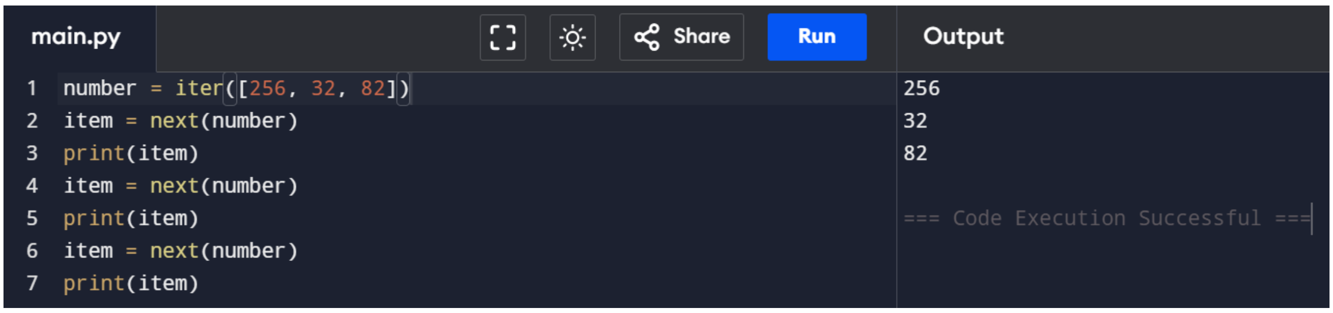
Python sorted() Function



```
str = "javatpoint"
sorted1 = sorted(str)
print(sorted1)
```

```
['a', 'a', 'i', 'j', 'n', 'o', 'p', 't', 't', 'v']

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

Python next() Function

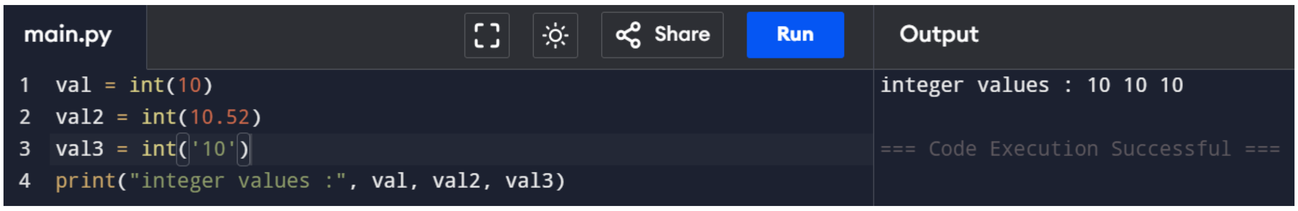


```
number = iter([256, 32, 82])
item = next(number)
print(item)
item = next(number)
print(item)
item = next(number)
print(item)
```

```
256
32
82

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

Python Project

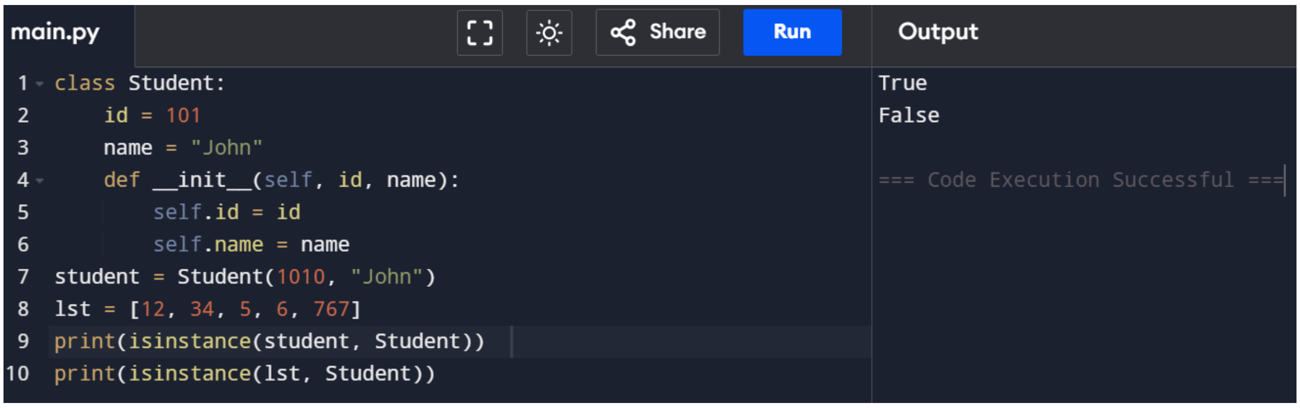
Python int() Function



```python
val = int(10)
val2 = int(10.52)
val3 = int('10')
print("integer values :", val, val2, val3)
```

```
integer values : 10 10 10

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

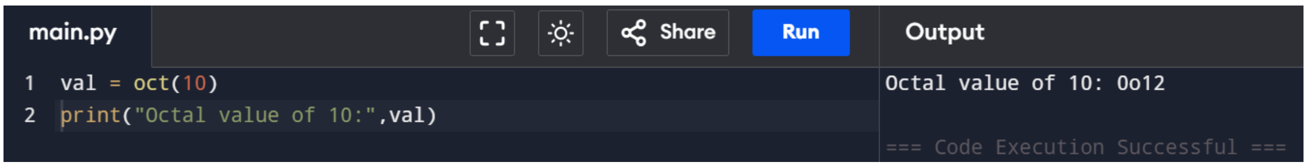
Python isinstance() function



```python
class Student:
    id = 101
    name = "John"
    def __init__(self, id, name):
        self.id = id
        self.name = name
student = Student(1010, "John")
lst = [12, 34, 5, 6, 767]
print(isinstance(student, Student))
print(isinstance(lst, Student))
```

```
True
False

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

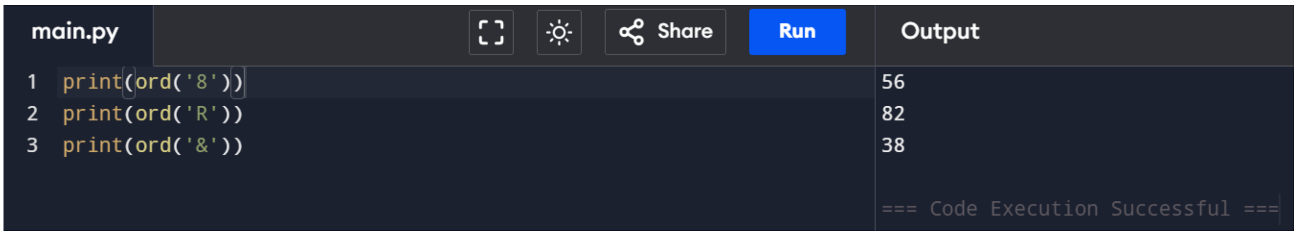
Python oct() function



```python
val = oct(10)
print("Octal value of 10:",val)
```

```
Octal value of 10: 0o12

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

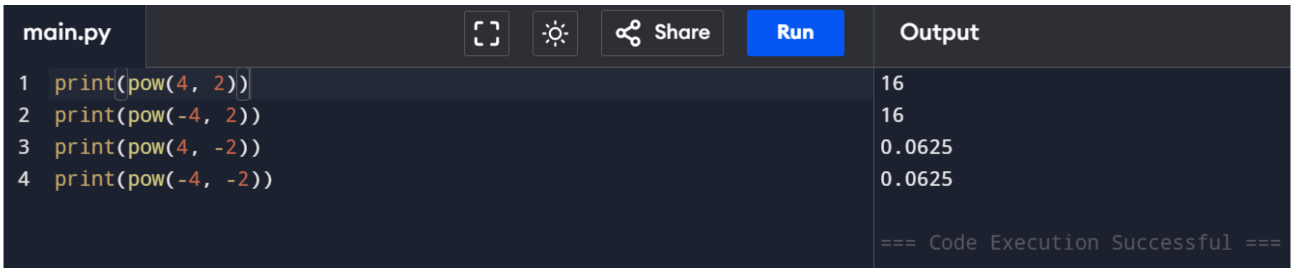
Python ord() function



```python
print(ord('8'))
print(ord('R'))
print(ord('&'))
```

```
56
82
38

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

Python pow() function

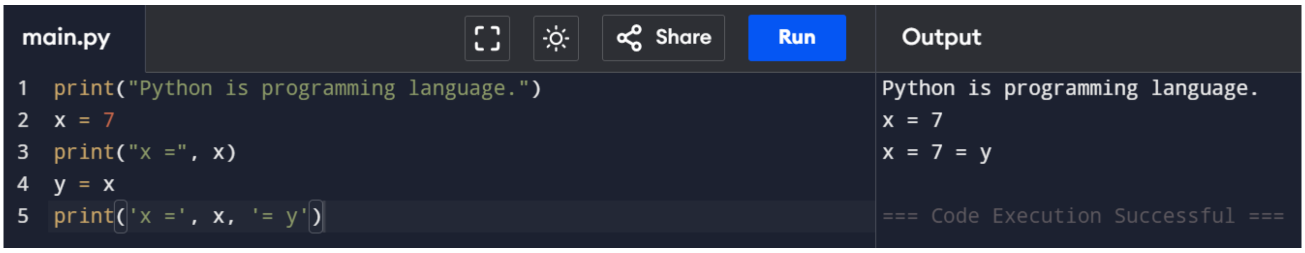


```python
print(pow(4, 2))
print(pow(-4, 2))
print(pow(4, -2))
print(pow(-4, -2))
```

```
16
16
0.0625
0.0625

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

Python Project

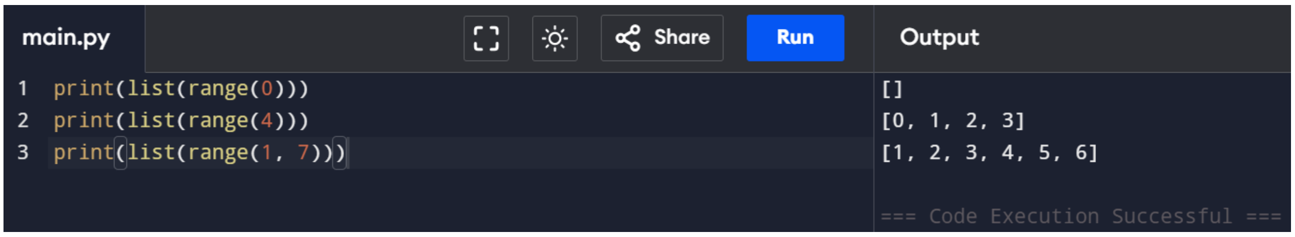
Python print() function



```python
print("Python is programming language.")
x = 7
print("x =", x)
y = x
print('x =', x, '= y')
```

```
Python is programming language.
x = 7
x = 7 = y

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

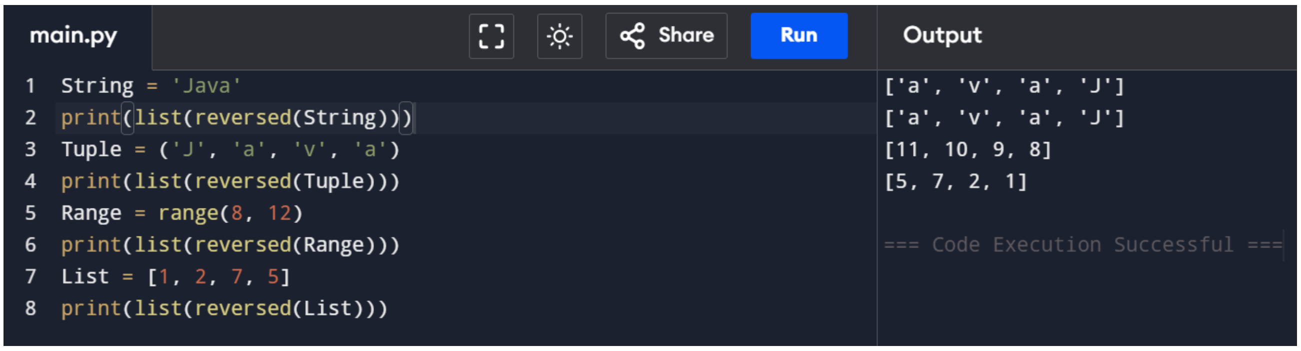
Python range() function



```python
print(list(range(0)))
print(list(range(4)))
print(list(range(1, 7)))
```

```
[]
[0, 1, 2, 3]
[1, 2, 3, 4, 5, 6]

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

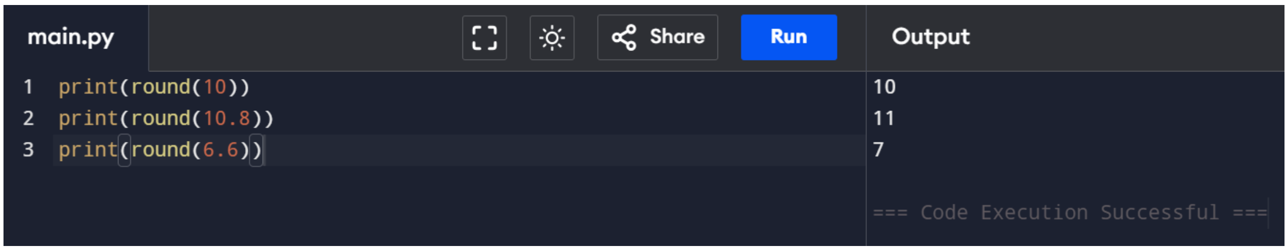
Python reversed() function



```python
String = 'Java'
print(list(reversed(String)))
Tuple = ('J', 'a', 'v', 'a')
print(list(reversed(Tuple)))
Range = range(8, 12)
print(list(reversed(Range)))
List = [1, 2, 7, 5]
print(list(reversed(List)))
```

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

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

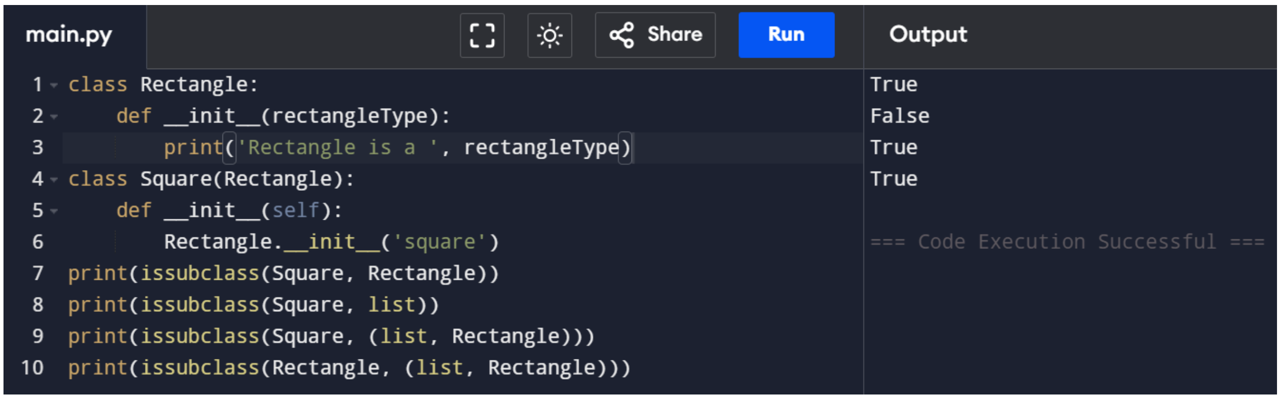
Python round() Function



```python
print(round(10))
print(round(10.8))
print(round(6.6))
```

```
10
11
7

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

Python issubclass() Function

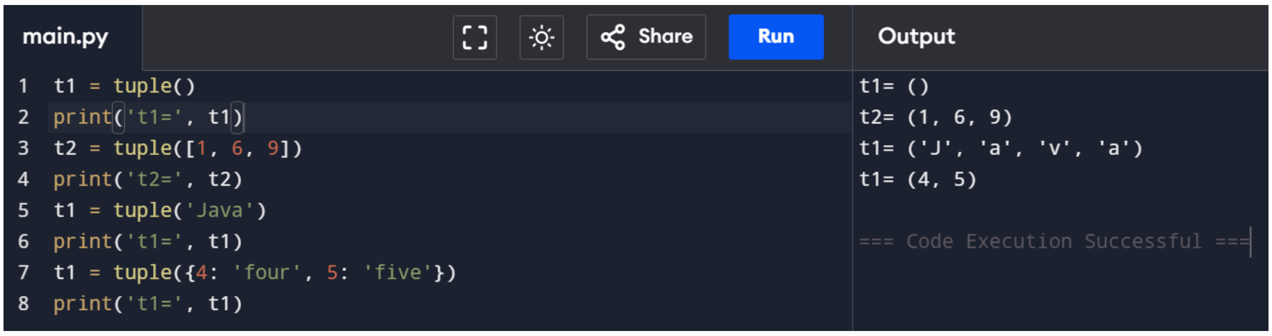


```python
class Rectangle:
    def __init__(rectangleType):
        print('Rectangle is a ', rectangleType)
class Square(Rectangle):
    def __init__(self):
        Rectangle.__init__('square')
print(issubclass(Square, Rectangle))
print(issubclass(Square, list))
print(issubclass(Square, (list, Rectangle)))
print(issubclass(Rectangle, (list, Rectangle)))
```

```
True
False
True
True

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

Python Project

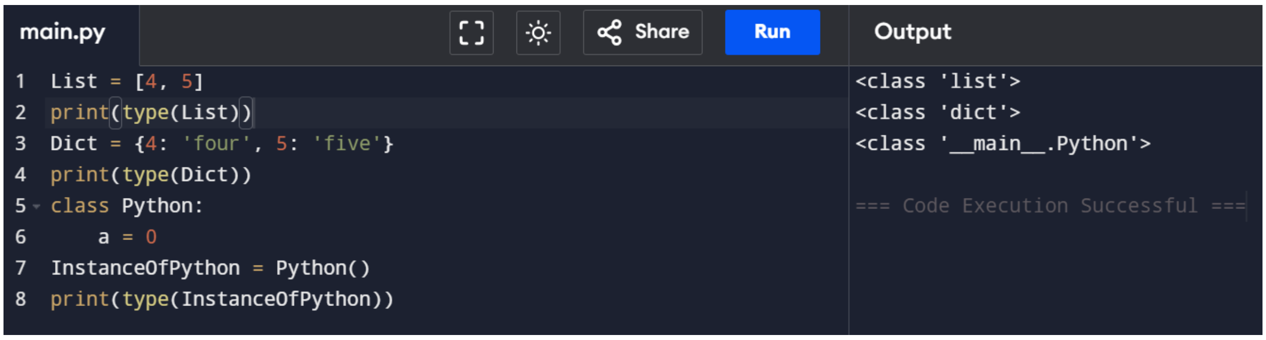
Python tuple() Function



```python
t1 = tuple()
print('t1=', t1)
t2 = tuple([1, 6, 9])
print('t2=', t2)
t1 = tuple('Java')
print('t1=', t1)
t1 = tuple({4: 'four', 5: 'five'})
print('t1=', t1)
```

```
t1= ()
t2= (1, 6, 9)
t1= ('J', 'a', 'v', 'a')
t1= (4, 5)

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

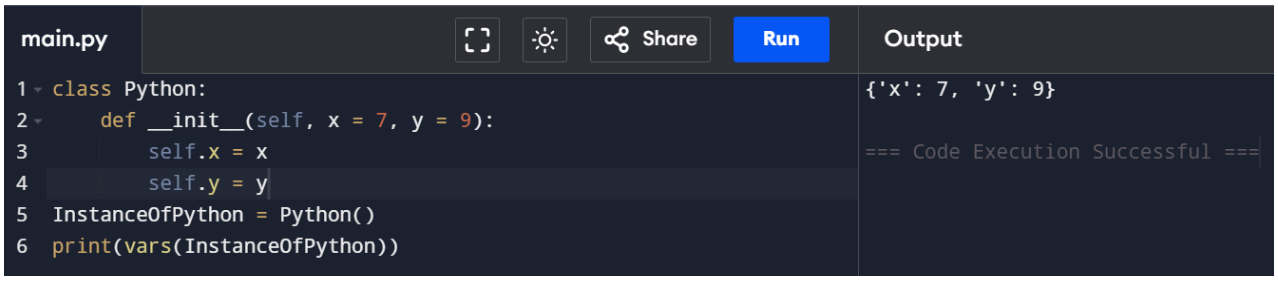
Python type() Function



```python
List = [4, 5]
print(type(List))
Dict = {4: 'four', 5: 'five'}
print(type(Dict))
class Python:
    a = 0
InstanceOfPython = Python()
print(type(InstanceOfPython))
```

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

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

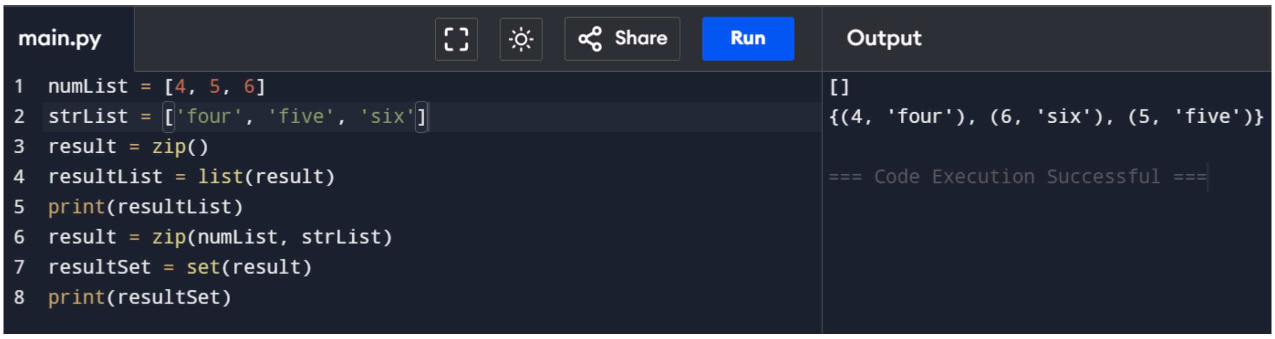
Python vars() Function



```python
class Python:
    def __init__(self, x = 7, y = 9):
        self.x = x
        self.y = y
InstanceOfPython = Python()
print(vars(InstanceOfPython))
```

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

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

Python zip() Function

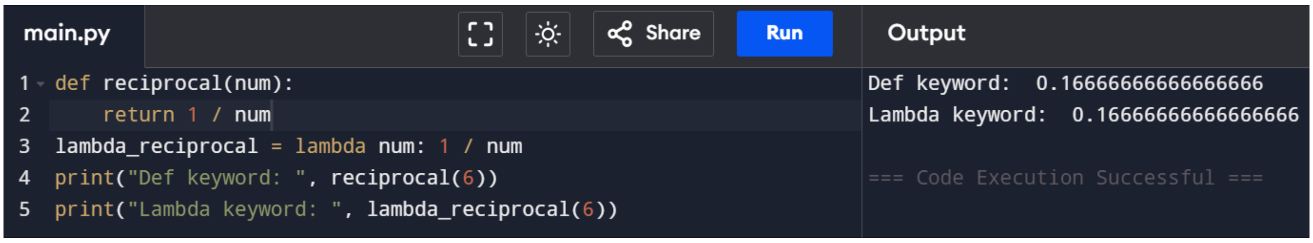


```python
numList = [4, 5, 6]
strList = ['four', 'five', 'six']
result = zip()
resultList = list(result)
print(resultList)
result = zip(numList, strList)
resultSet = set(result)
print(resultSet)
```

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

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

Python Project

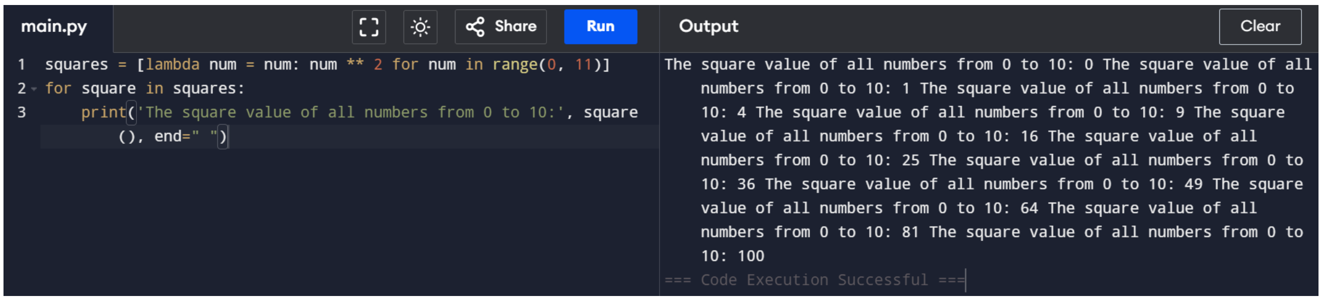
Python code to show the reciprocal of the given number to highlight the difference between def() and lambda()



```python
def reciprocal(num):
    return 1 / num
lambda_reciprocal = lambda num: 1 / num
print("Def keyword: ", reciprocal(6))
print("Lambda keyword: ", lambda_reciprocal(6))
```

```
Def keyword:  0.16666666666666666
Lambda keyword:  0.16666666666666666

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

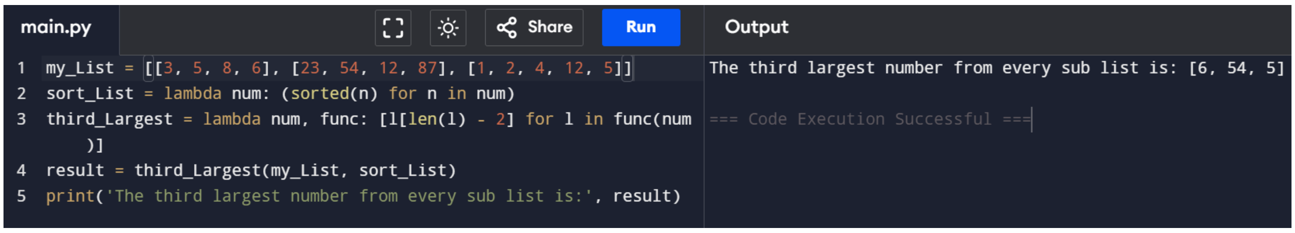
Code to calculate square of each number of lists using list comprehension



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

```
The square value of all numbers from 0 to 10: 0 The square value of all
    numbers from 0 to 10: 1 The square value of all numbers from 0 to
    10: 4 The square value of all numbers from 0 to 10: 9 The square
    value of all numbers from 0 to 10: 16 The square value of all
    numbers from 0 to 10: 25 The square value of all numbers from 0 to
    10: 36 The square value of all numbers from 0 to 10: 49 The square
    value of all numbers from 0 to 10: 64 The square value of all
    numbers from 0 to 10: 81 The square value of all numbers from 0 to
    10: 100
=== Code Execution Successful ===
```

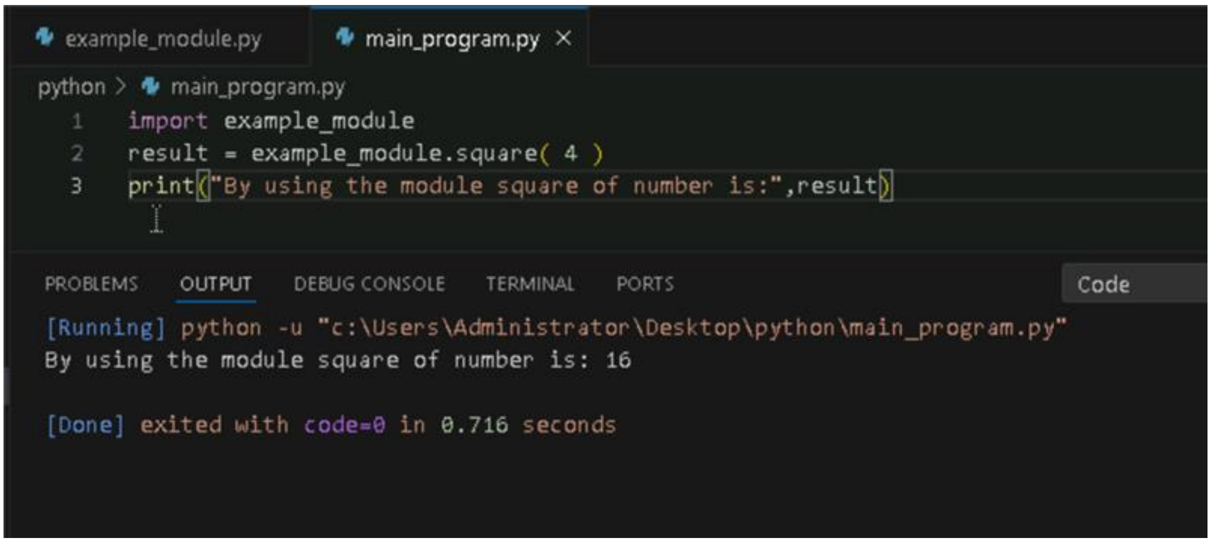
Code to print the third largest number of the given list using the lambda function



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

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

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

Python Modules

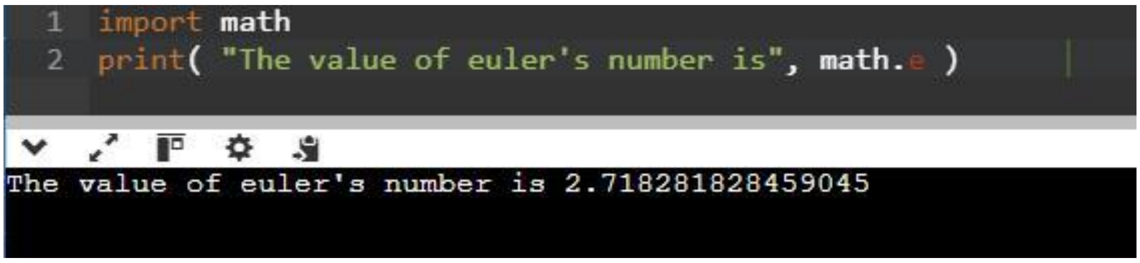


```python
import example_module
result = example_module.square( 4 )
print("By using the module square of number is:",result)
```

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

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

Python Project

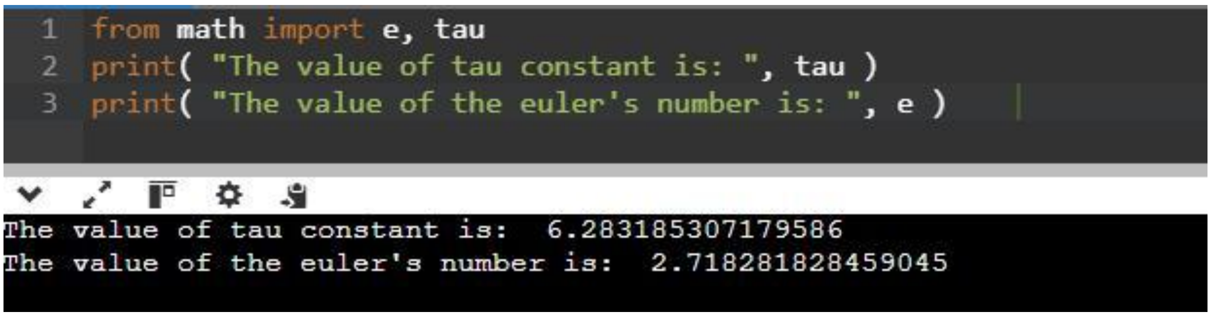
Importing and also Renaming



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

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

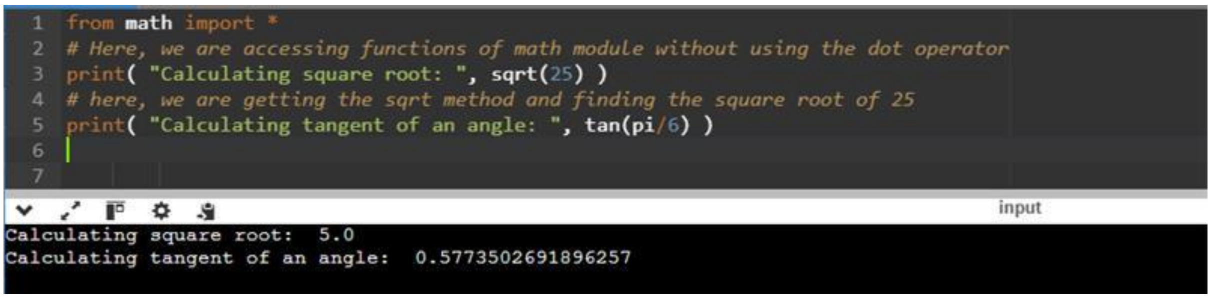
Python from...import Statement



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

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

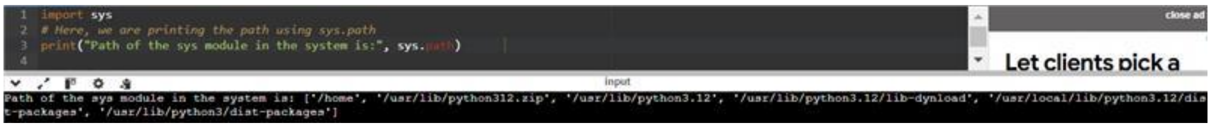
Import all Names - From import * Statement



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

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

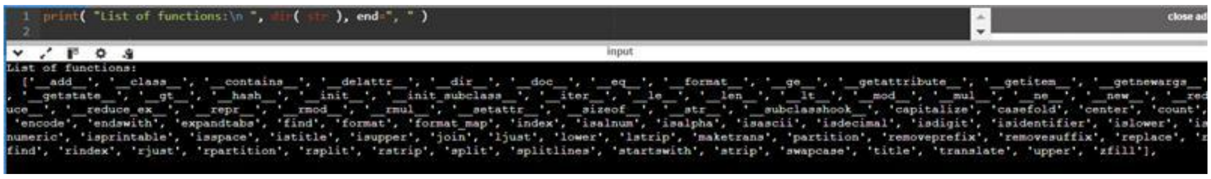
Locating Path of Modules



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

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

The dir() Built-in Function

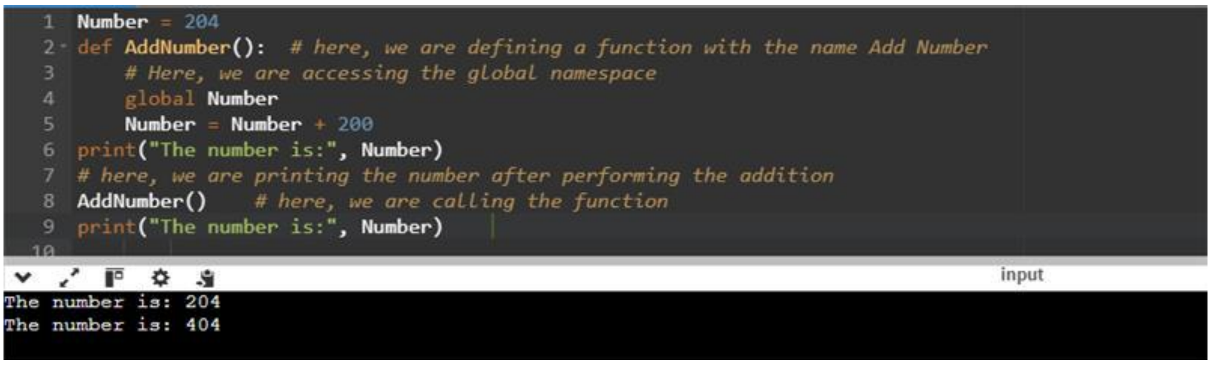


```python
print( "List of functions:\n ", dir( str ), end=", " )
```

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

Python Project

Namespaces and Scoping

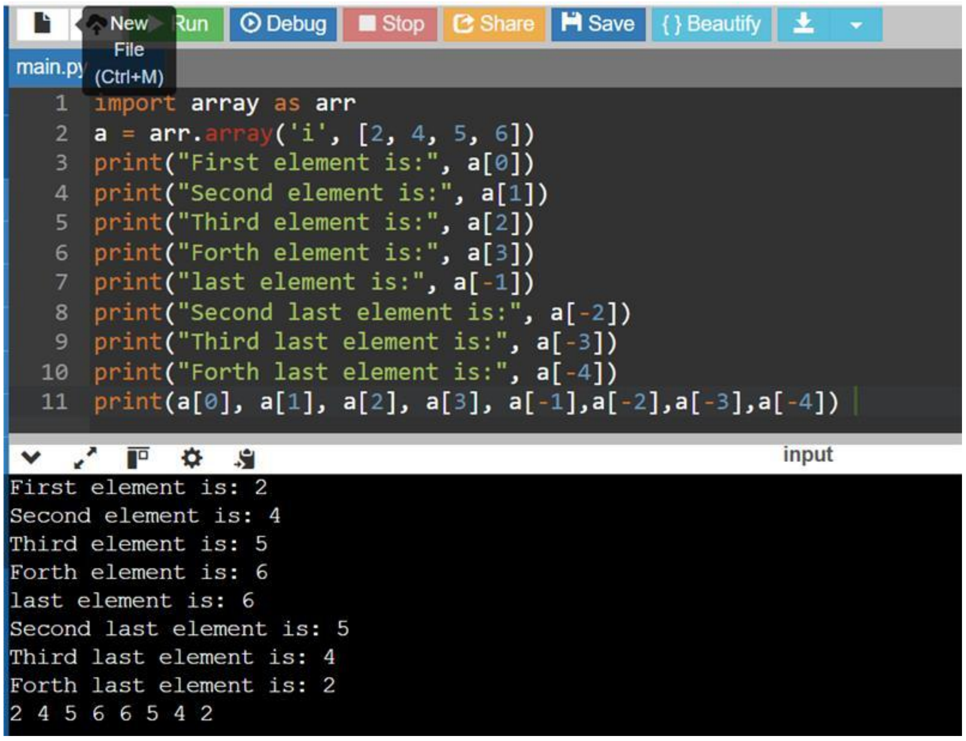


```python
Number = 204
def AddNumber():  # here, we are defining a function with the name Add Number
    # Here, we are accessing the global namespace
    global Number
    Number = Number + 200
print("The number is:", Number)
# here, we are printing the number after performing the addition
AddNumber()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

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

PYTHON ARRAYS

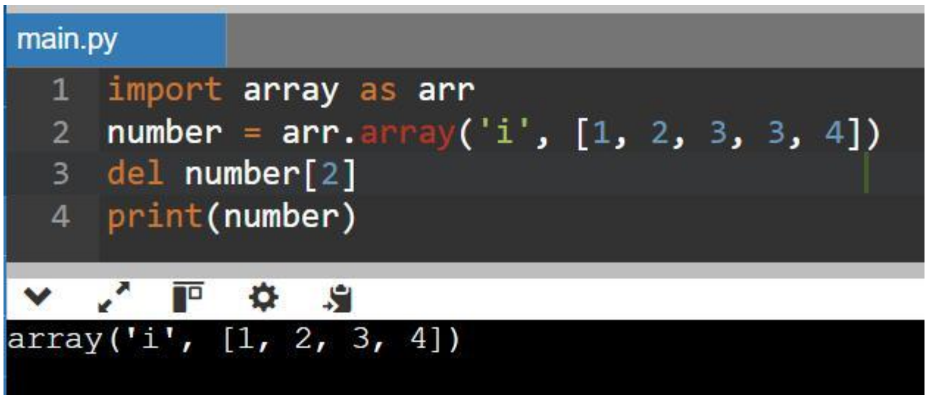
1. **Accessing array elements**



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

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

Deleting the elements from Array

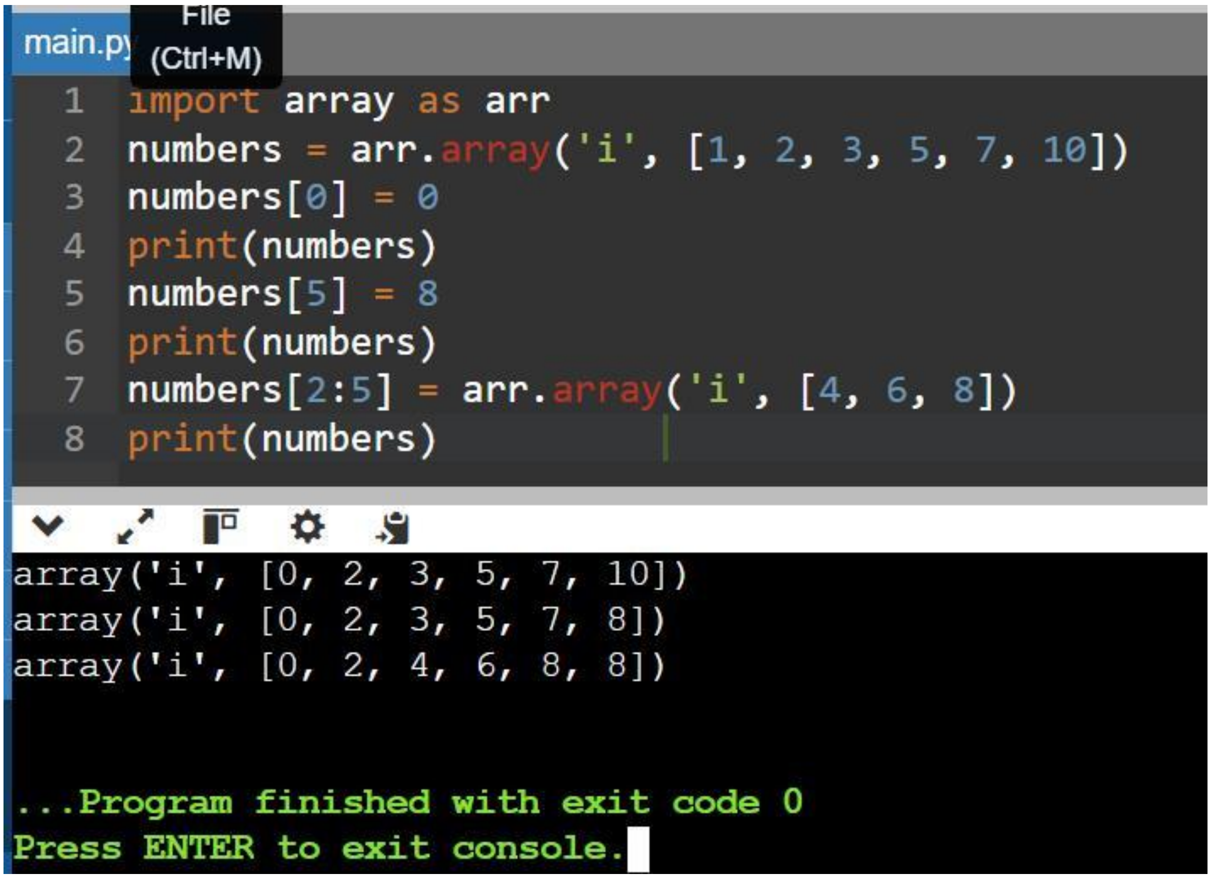


```python
import array as arr
number = arr.array('i', [1, 2, 3, 3, 4])
del number[2]
print(number)
```

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

Python Project

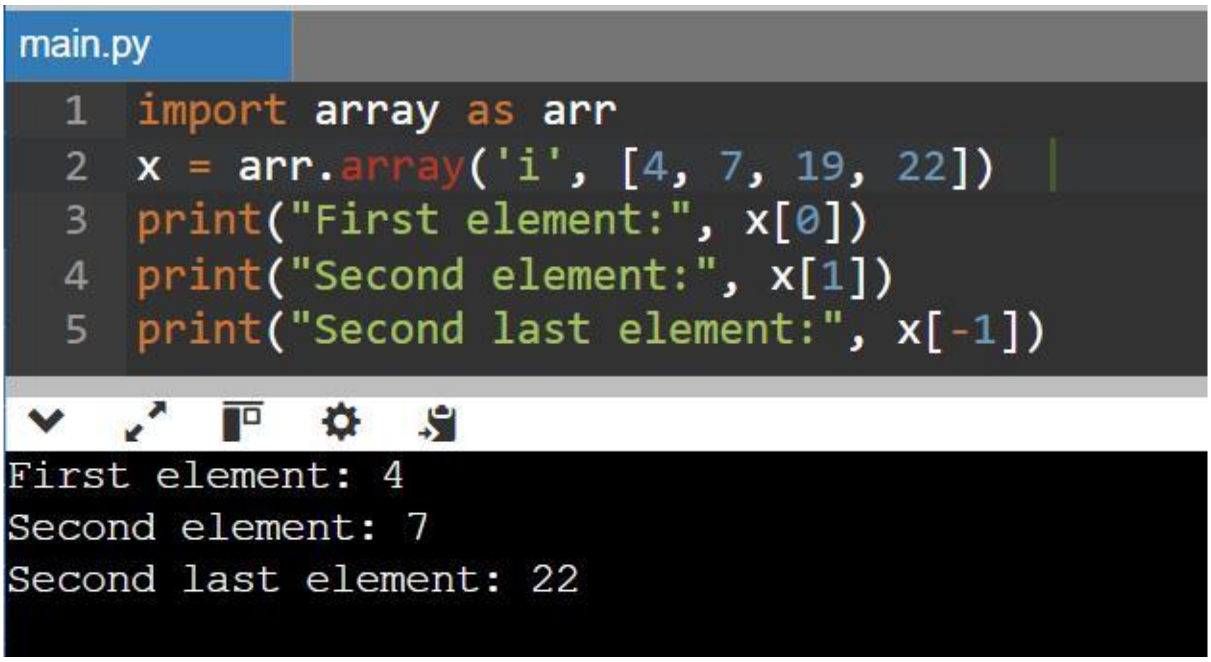
1. **Adding or changing the elements in Array**



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

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


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

To find the length of array

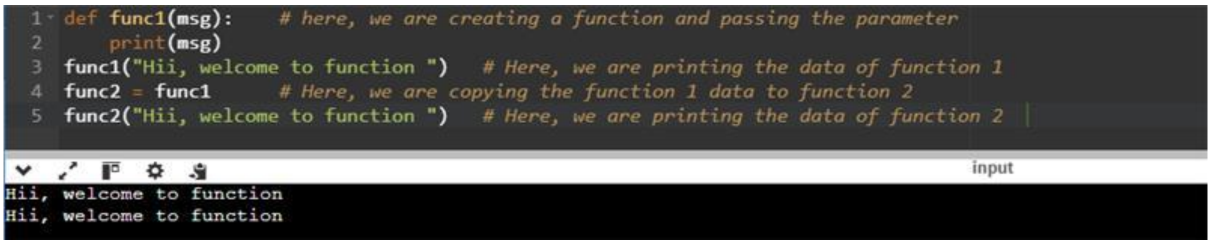


```python
import array as arr
x = arr.array('i', [4, 7, 19, 22])
print("First element:", x[0])
print("Second element:", x[1])
print("Second last element:", x[-1])
```

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

Python Project

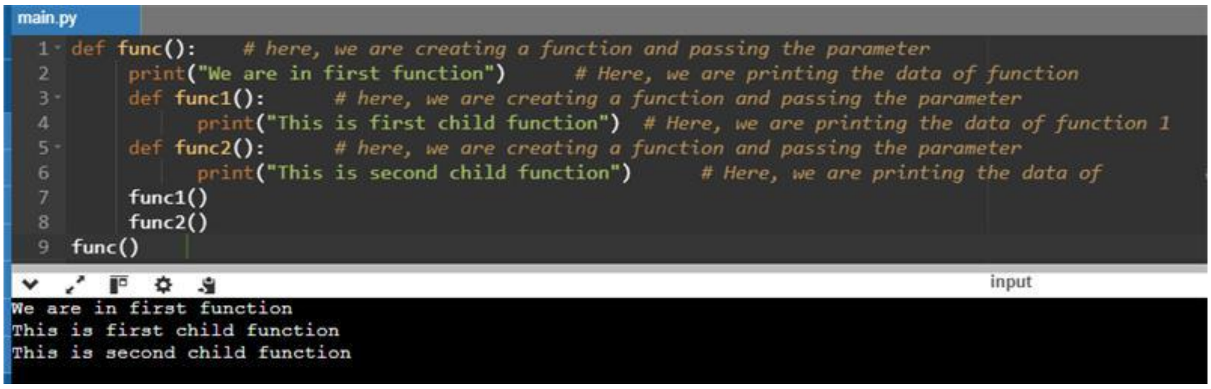
Python Decorator



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

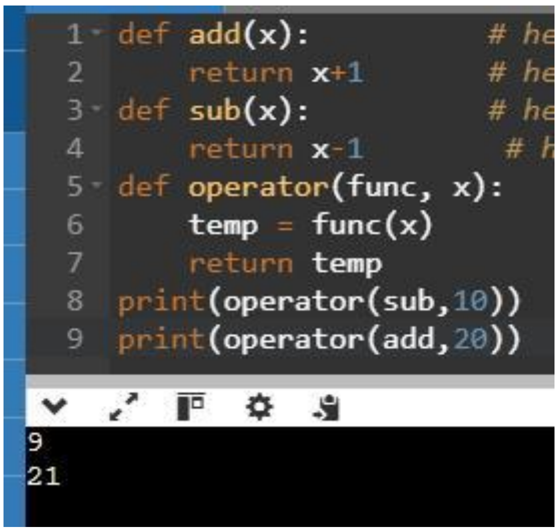
```
Hii, welcome to function
Hii, welcome to function
```

Inner Function



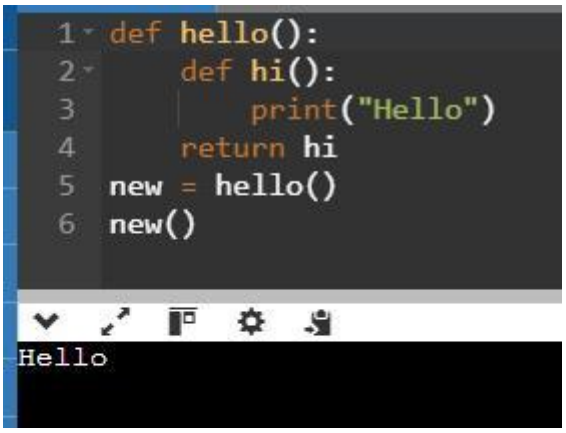
```python
def func():     # here, we are creating a function and passing the parameter
    print("We are in first function")      # Here, we are printing the data of function
    def func1():      # here, we are creating a function and passing the parameter
        print("This is first child function")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

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



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

```
9
21
```

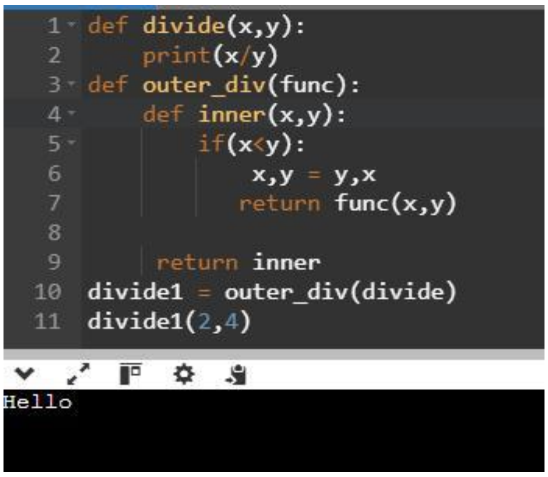


```python
def hello():
    def hi():
        print("Hello")
    return hi
new = hello()
new()
```

```
Hello
```

Python Project

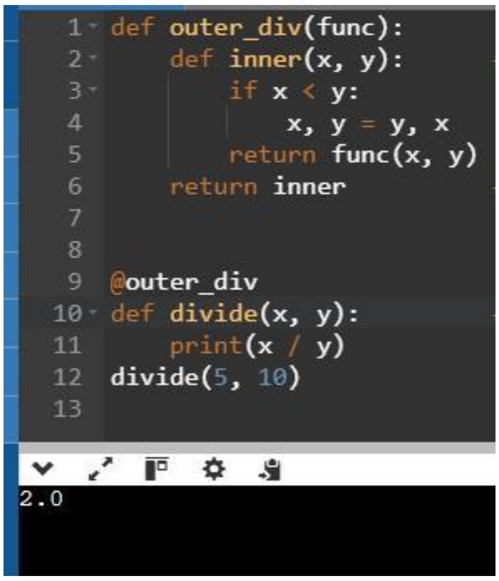
Decorating functions with parameters



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

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

```
Hello
```

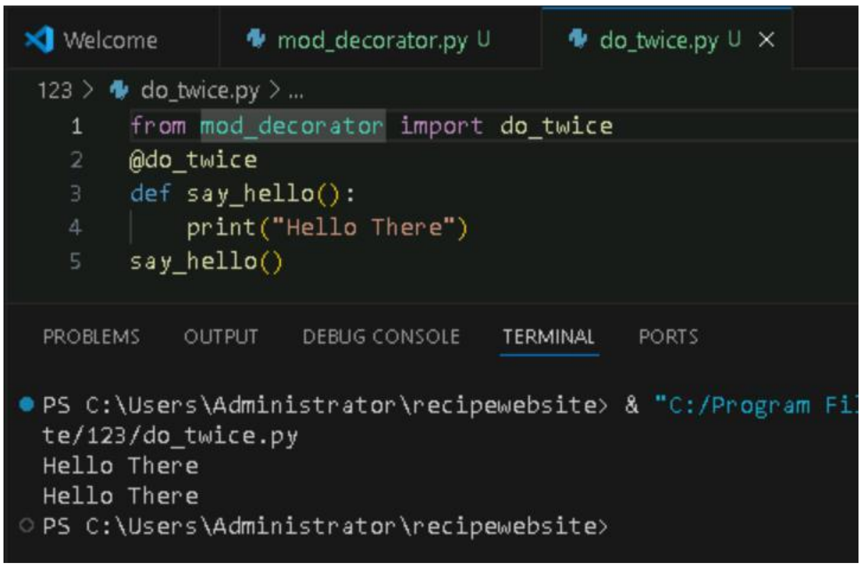
Syntactic Decorator



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


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

```
2.0
```

Reusing Decorator

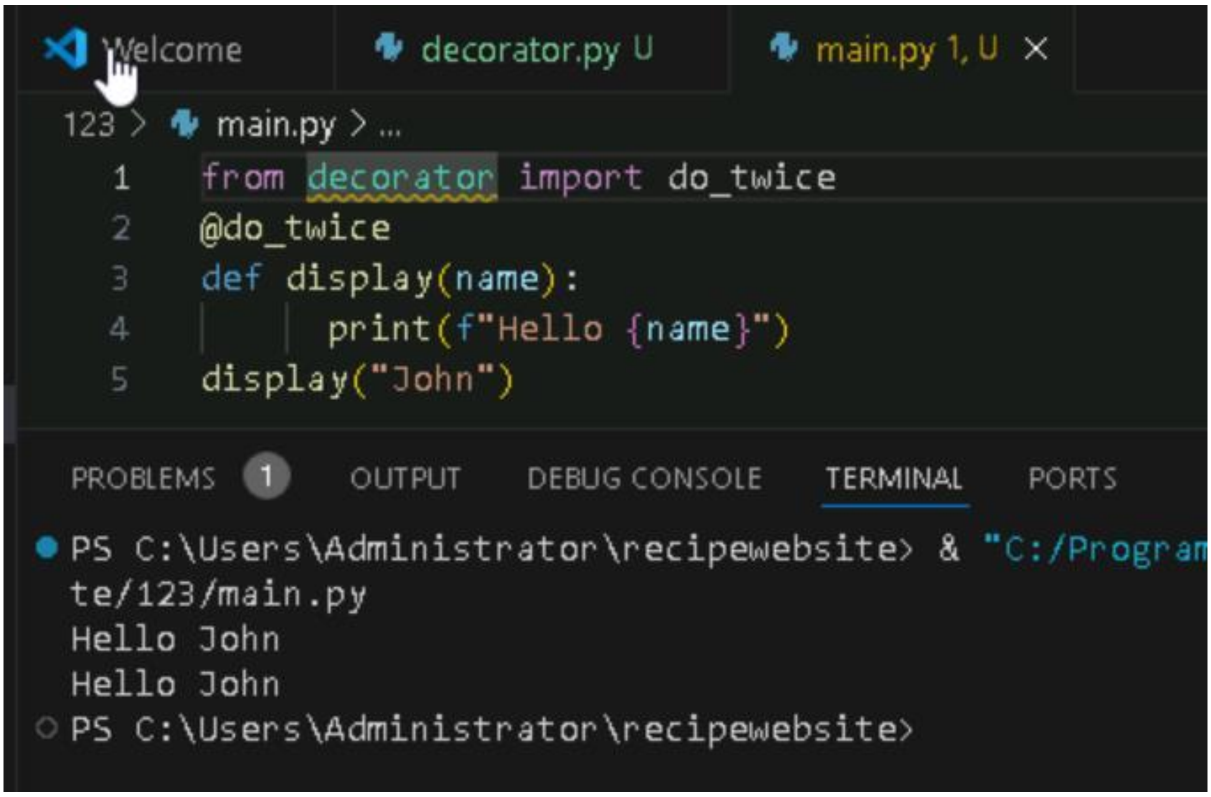


```python
from mod_decorator import do_twice
@do_twice
def say_hello():
    print("Hello There")
say_hello()
```

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

Python Project

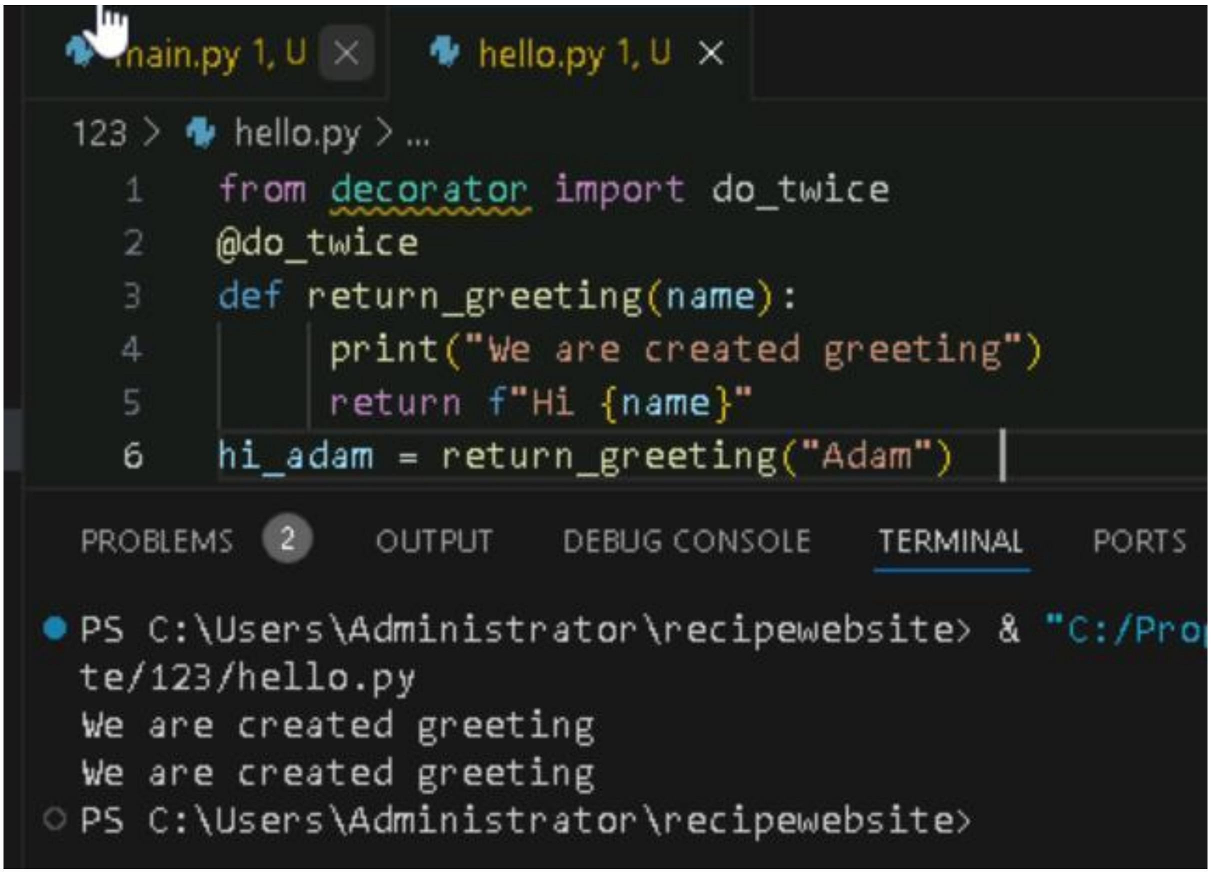
Python Decorator with Argument



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

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

Returning Values from Decorated Functions

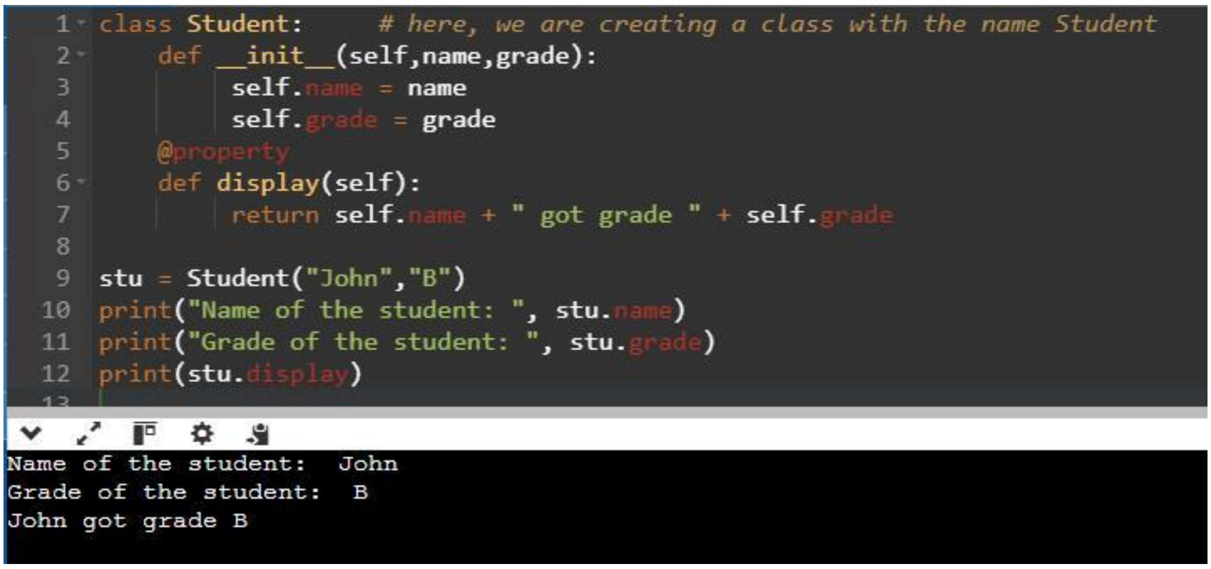


```python
from decorator import do_twice
@do_twice
def return_greeting(name):
        print("We are created greeting")
        return f"Hi {name}"
hi_adam = return_greeting("Adam")
```

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

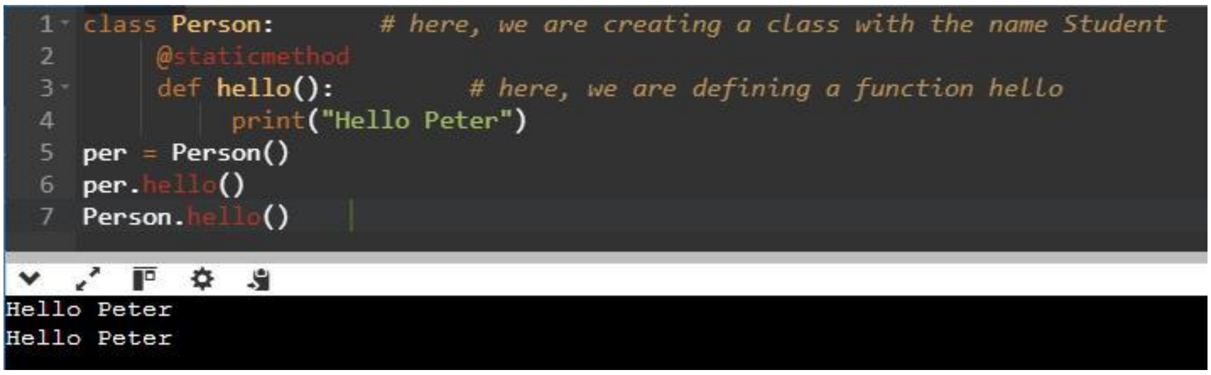
Python Project

Fancy Decorators



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

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

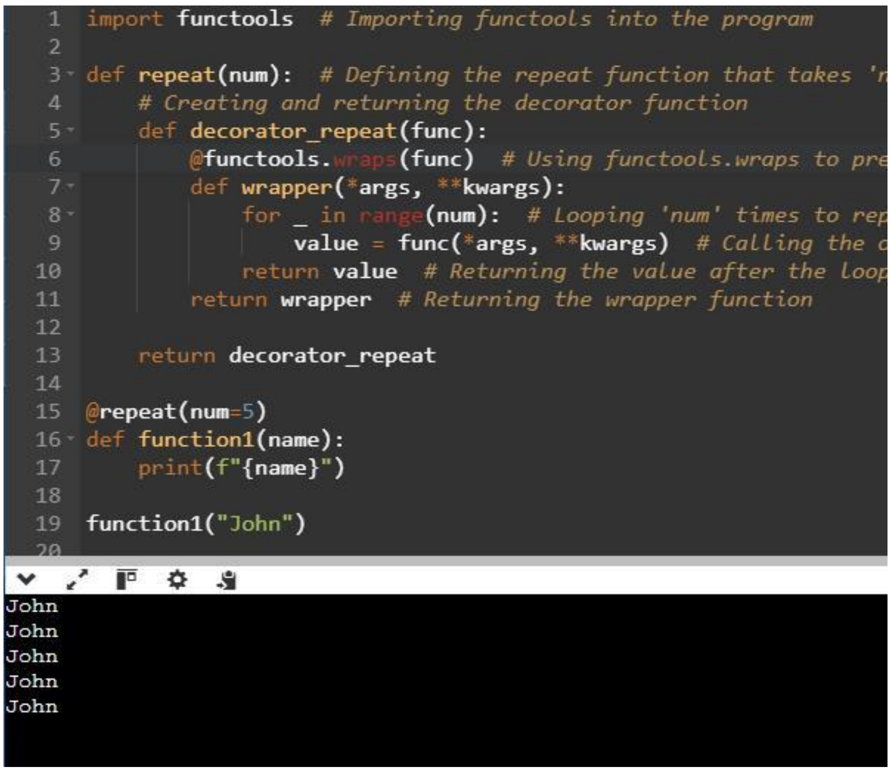
```
Name of the student:  John
Grade of the student:  B
John got grade B
```



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

```
Hello Peter
Hello Peter
```

Decorator with Arguments

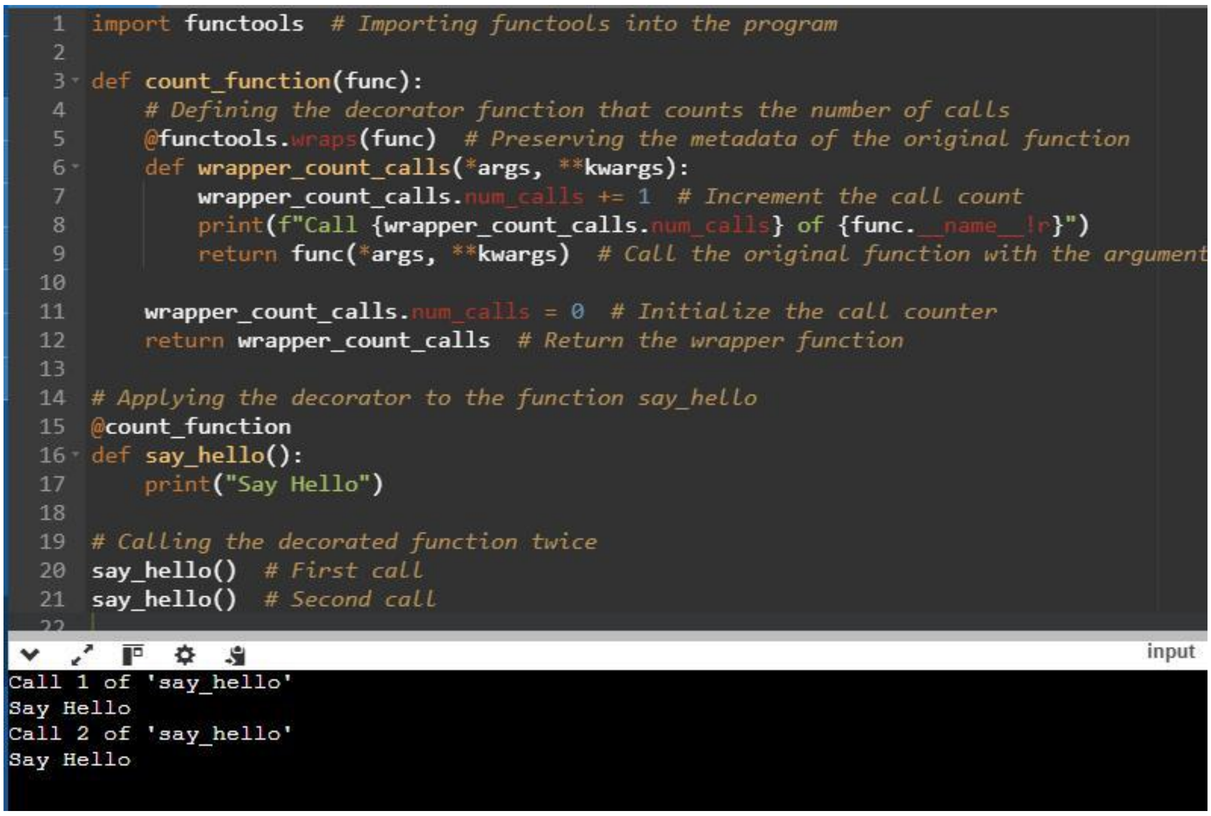


```python
import functools  # Importing functools into the program

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

    return decorator_repeat

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

function1("John")
```

```
John
John
John
John
John
```

Python Project

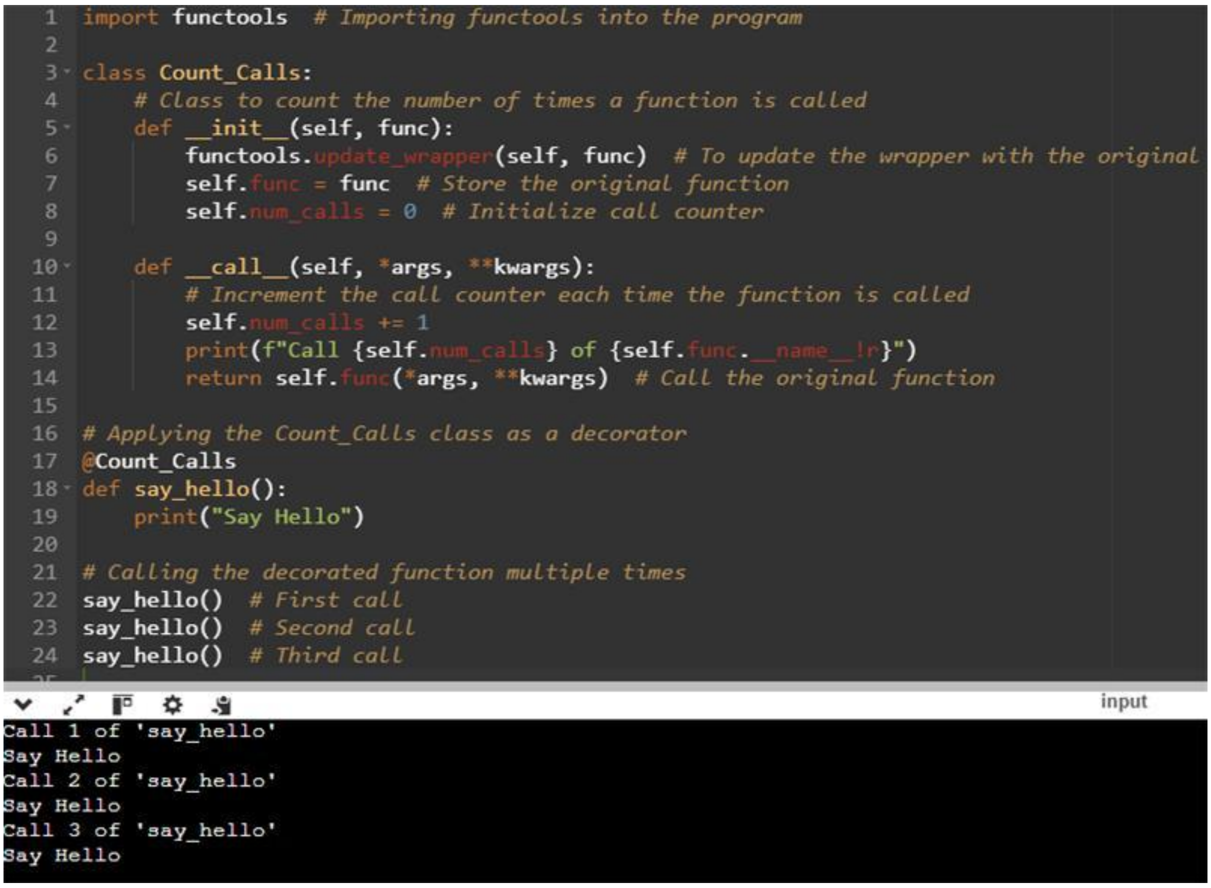
Stateful Decorators



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

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

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

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

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

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

Classes as Decorators



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

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

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

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

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

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