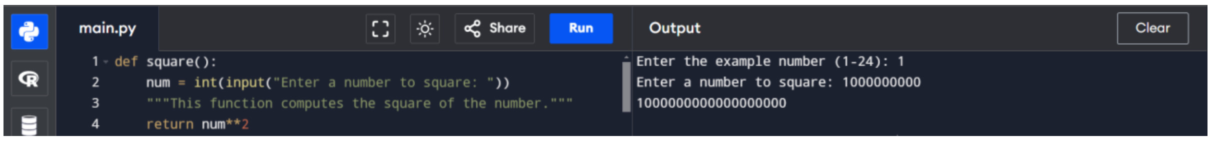
PYTHON FUNCTIONS

Example:

Python Functions:

Examples :

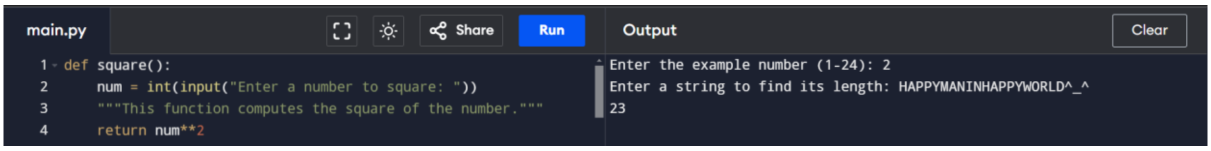
1.



```python
def square():
    num = int(input("Enter a number to square: "))
    """This function computes the square of the number."""
    return num**2
```

```
Enter the example number (1-24): 1
Enter a number to square: 1000000000
1000000000000000000
```

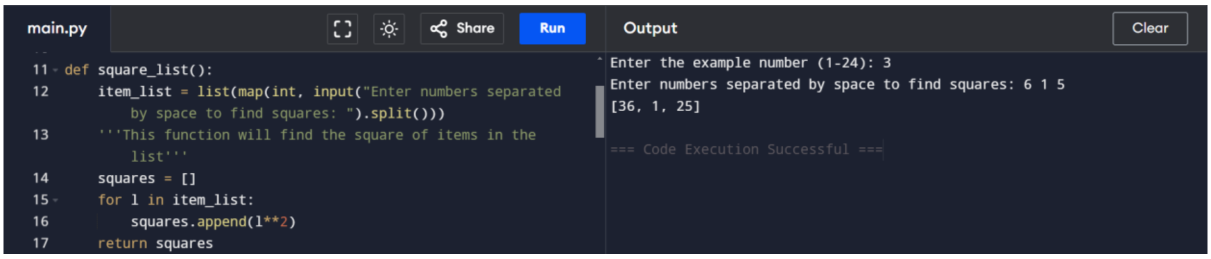
2.



```python
def square():
    num = int(input("Enter a number to square: "))
    """This function computes the square of the number."""
    return num**2
```

```
Enter the example number (1-24): 2
Enter a string to find its length: HAPPYMANINHAPPYWORLD^_^
23
```

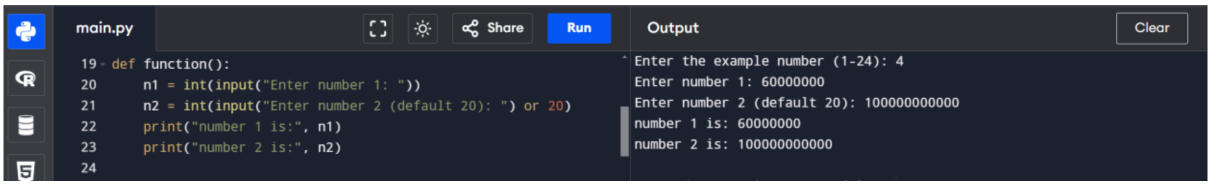
3.



```python
def square_list():
    item_list = list(map(int, input("Enter numbers separated
        by space to find squares: ").split()))
    '''This function will find the square of items in the
        list'''
    squares = []
    for l in item_list:
        squares.append(l**2)
    return squares
```

```
Enter the example number (1-24): 3
Enter numbers separated by space to find squares: 6 1 5
[36, 1, 25]

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

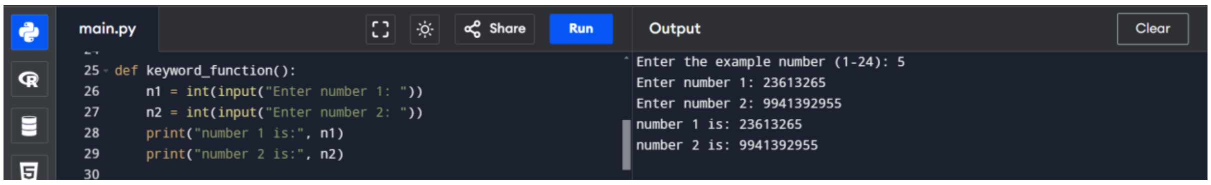
4.



```python
def function():
    n1 = int(input("Enter number 1: "))
    n2 = int(input("Enter number 2 (default 20): ") or 20)
    print("number 1 is:", n1)
    print("number 2 is:", n2)
```

```
Enter the example number (1-24): 4
Enter number 1: 60000000
Enter number 2 (default 20): 100000000000
number 1 is: 60000000
number 2 is: 100000000000
```

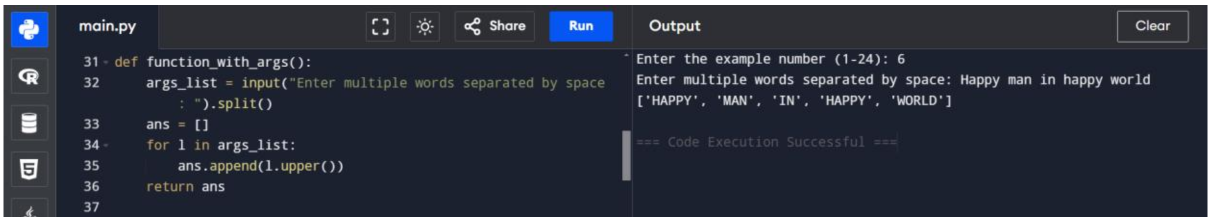
5.



```python
def keyword_function():
    n1 = int(input("Enter number 1: "))
    n2 = int(input("Enter number 2: "))
    print("number 1 is:", n1)
    print("number 2 is:", n2)
```

```
Enter the example number (1-24): 5
Enter number 1: 23613265
Enter number 2: 9941392955
number 1 is: 23613265
number 2 is: 9941392955
```

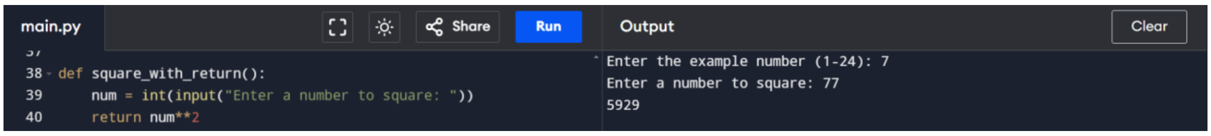
6.



```python
def function_with_args():
    args_list = input("Enter multiple words separated by space: ").split()
    ans = []
    for l in args_list:
        ans.append(l.upper())
    return ans
```

```
Enter the example number (1-24): 6
Enter multiple words separated by space: Happy man in happy world
['HAPPY', 'MAN', 'IN', 'HAPPY', 'WORLD']

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

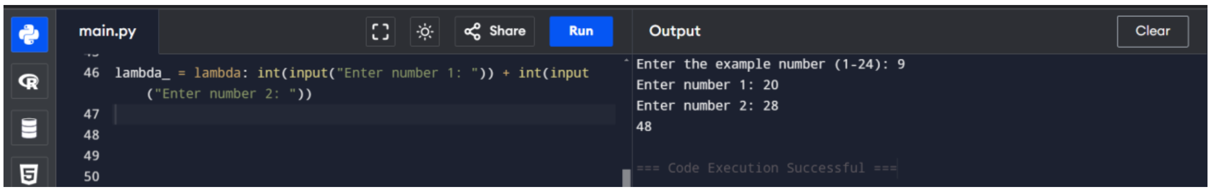
7.



```python
def square_with_return():
    num = int(input("Enter a number to square: "))
    return num**2
```

```
Enter the example number (1-24): 7
Enter a number to square: 77
5929
```

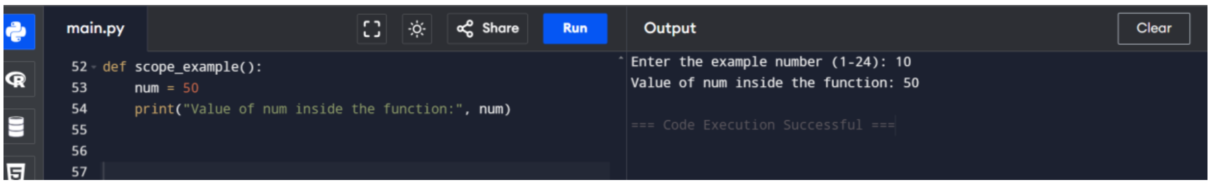
8.



```python
lambda_ = lambda: int(input("Enter number 1: ")) + int(input("Enter number 2: "))
```

```
Enter the example number (1-24): 9
Enter number 1: 20
Enter number 2: 28
48

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

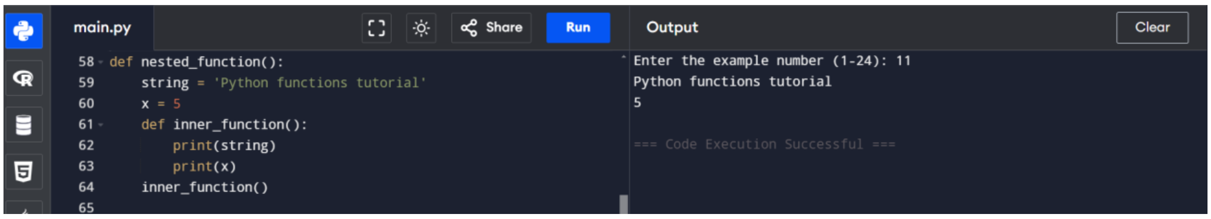
9.



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

```
Enter the example number (1-24): 10
Value of num inside the function: 50

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

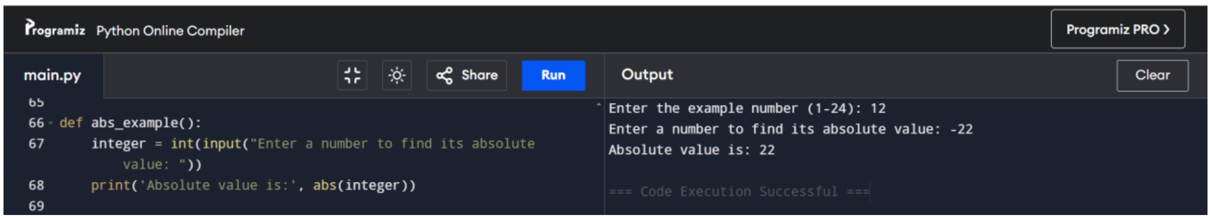
10.



```python
def nested_function():
    string = 'Python functions tutorial'
    x = 5
    def inner_function():
        print(string)
        print(x)
    inner_function()
```

```
Enter the example number (1-24): 11
Python functions tutorial
5

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

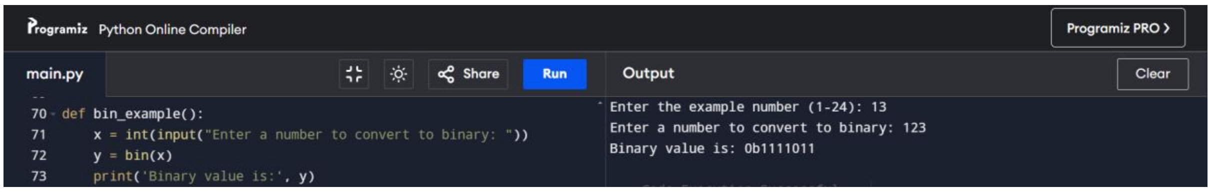
11.



```python
def abs_example():
    integer = int(input("Enter a number to find its absolute value: "))
    print('Absolute value is:', abs(integer))
```

```
Enter the example number (1-24): 12
Enter a number to find its absolute value: -22
Absolute value is: 22

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

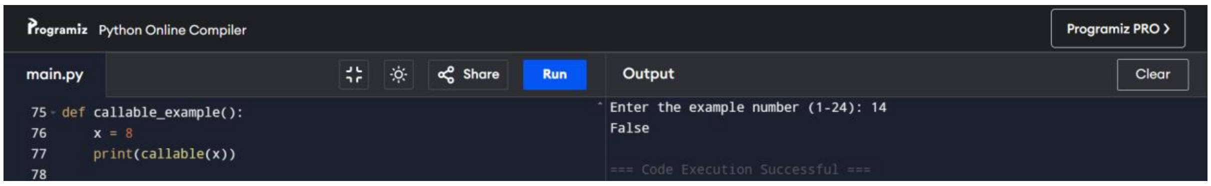
12.



```python
def bin_example():
    x = int(input("Enter a number to convert to binary: "))
    y = bin(x)
    print('Binary value is:', y)
```

```
Enter the example number (1-24): 13
Enter a number to convert to binary: 123
Binary value is: 0b1111011
```

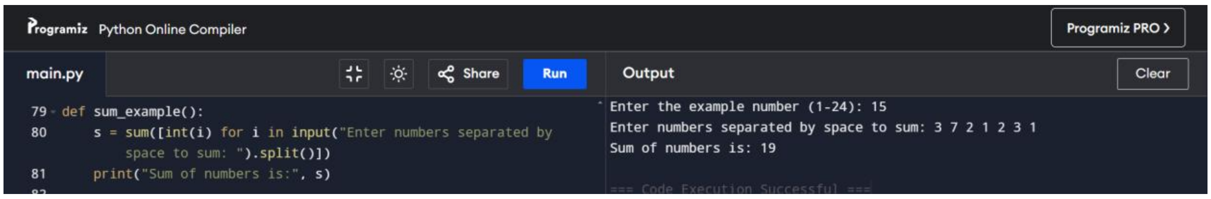
13.



```python
def callable_example():
    x = 8
    print(callable(x))
```

```
Enter the example number (1-24): 14
False

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

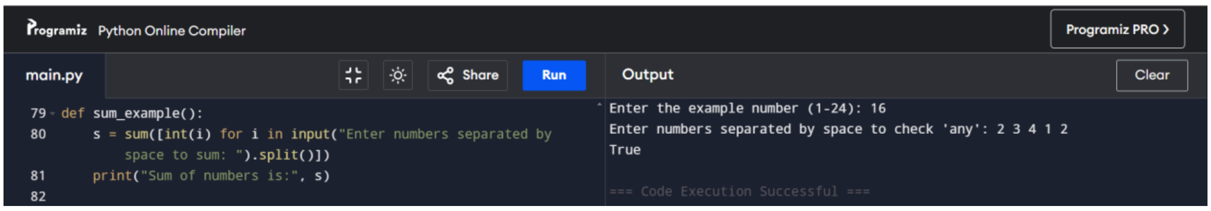
14.



```python
def sum_example():
    s = sum([int(i) for i in input("Enter numbers separated by
        space to sum: ").split()])
    print("Sum of numbers is:", s)
```

```
Enter the example number (1-24): 15
Enter numbers separated by space to sum: 3 7 2 1 2 3 1
Sum of numbers is: 19

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

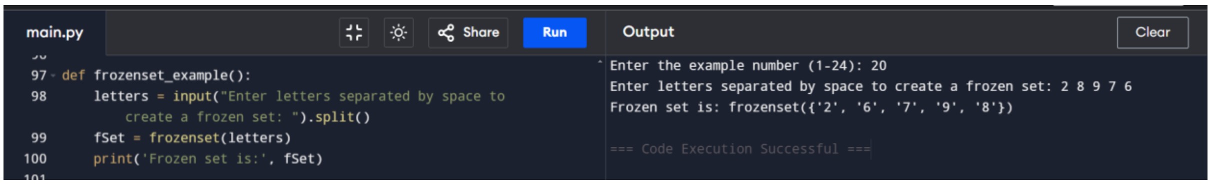
15.



```python
def sum_example():
    s = sum([int(i) for i in input("Enter numbers separated by
        space to sum: ").split()])
    print("Sum of numbers is:", s)
```

```
Enter the example number (1-24): 16
Enter numbers separated by space to check 'any': 2 3 4 1 2
True

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

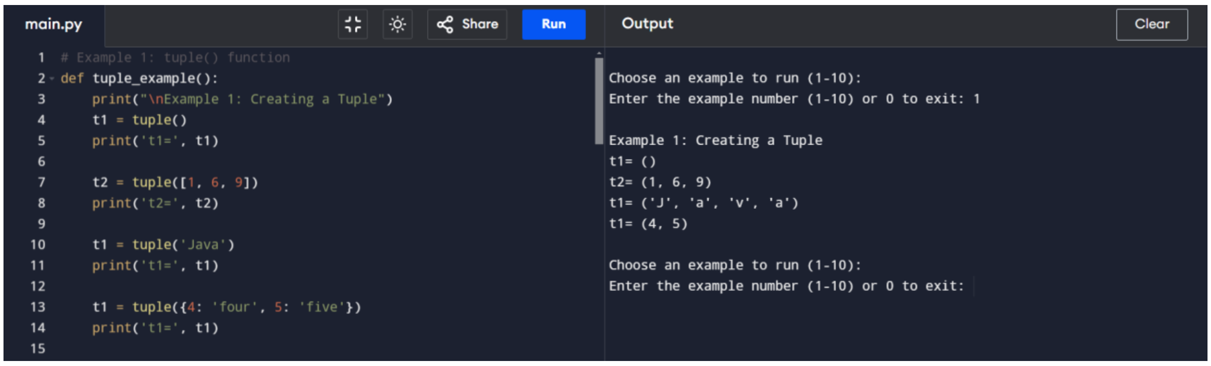
16.



```python
def frozenset_example():
    letters = input("Enter letters separated by space to
        create a frozen set: ").split()
    fSet = frozenset(letters)
    print('Frozen set is:', fSet)
```

```
Enter the example number (1-24): 20
Enter letters separated by space to create a frozen set: 2 8 9 7 6
Frozen set is: frozenset({'2', '6', '7', '9', '8'})

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

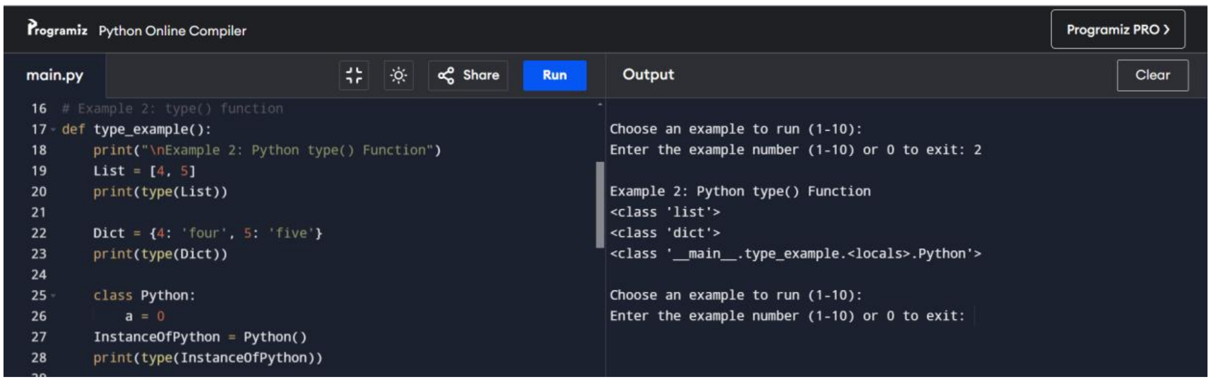
17.



```python
# Example 1: tuple() function
def tuple_example():
    print("\nExample 1: Creating a Tuple")
    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)

```

```
Choose an example to run (1-10):
Enter the example number (1-10) or 0 to exit: 1

Example 1: Creating a Tuple
t1= ()
t2= (1, 6, 9)
t1= ('J', 'a', 'v', 'a')
t1= (4, 5)

Choose an example to run (1-10):
Enter the example number (1-10) or 0 to exit:
```

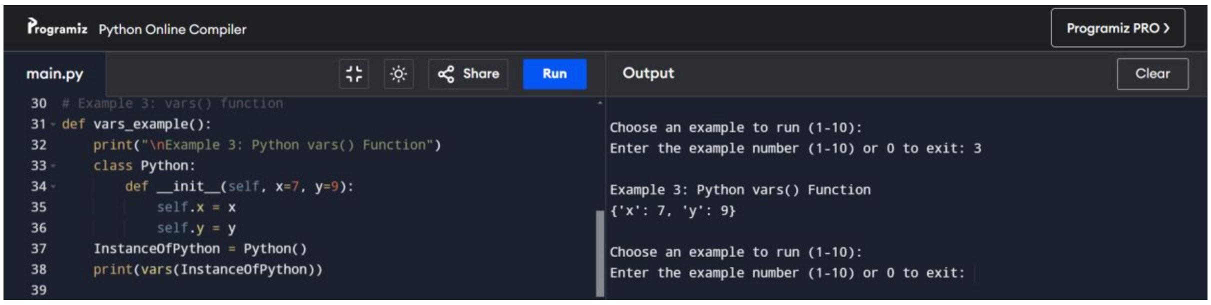
18.



```python
# Example 2: type() function
def type_example():
    print("\nExample 2: Python type() Function")
    List = [4, 5]
    print(type(List))

    Dict = {4: 'four', 5: 'five'}
    print(type(Dict))

    class Python:
        a = 0
    InstanceOfPython = Python()
    print(type(InstanceOfPython))
```

```
Choose an example to run (1-10):
Enter the example number (1-10) or 0 to exit: 2

Example 2: Python type() Function
<class 'list'>
<class 'dict'>
<class '__main__.type_example.<locals>.Python'>

Choose an example to run (1-10):
Enter the example number (1-10) or 0 to exit:
```

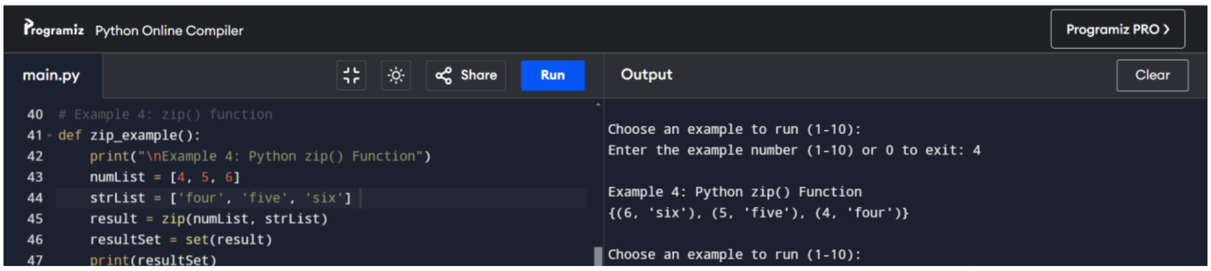
19.



```python
# Example 3: vars() function
def vars_example():
    print("\nExample 3: Python vars() Function")
    class Python:
        def __init__(self, x=7, y=9):
            self.x = x
            self.y = y
    InstanceOfPython = Python()
    print(vars(InstanceOfPython))

```

```
Choose an example to run (1-10):
Enter the example number (1-10) or 0 to exit: 3

Example 3: Python vars() Function
{'x': 7, 'y': 9}

Choose an example to run (1-10):
Enter the example number (1-10) or 0 to exit:
```

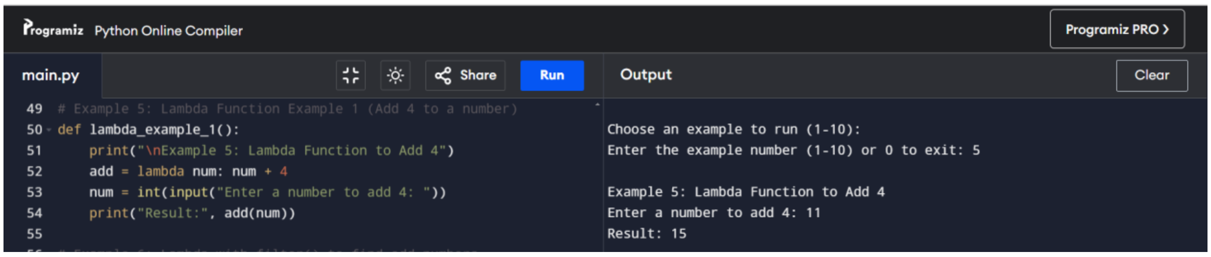
20.



```python
# Example 4: zip() function
def zip_example():
    print("\nExample 4: Python zip() Function")
    numList = [4, 5, 6]
    strList = ['four', 'five', 'six']
    result = zip(numList, strList)
    resultSet = set(result)
    print(resultSet)
```

```
Choose an example to run (1-10):
Enter the example number (1-10) or 0 to exit: 4

Example 4: Python zip() Function
{(6, 'six'), (5, 'five'), (4, 'four')}

Choose an example to run (1-10):
```

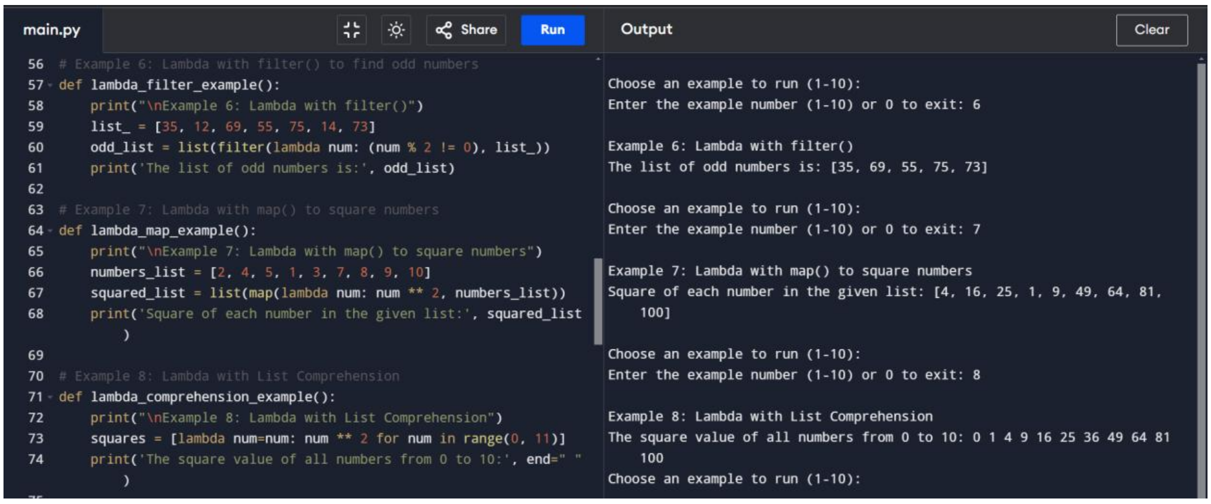
21.



```python
# Example 5: Lambda Function Example 1 (Add 4 to a number)
def lambda_example_1():
    print("\nExample 5: Lambda Function to Add 4")
    add = lambda num: num + 4
    num = int(input("Enter a number to add 4: "))
    print("Result:", add(num))

```

```
Choose an example to run (1-10):
Enter the example number (1-10) or 0 to exit: 5

Example 5: Lambda Function to Add 4
Enter a number to add 4: 11
Result: 15
```

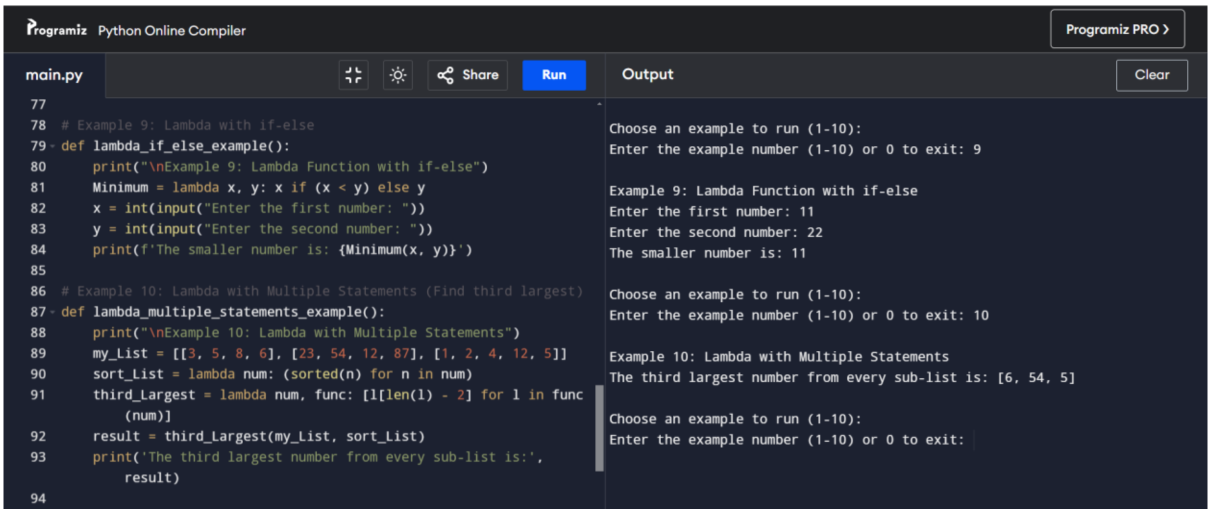
22,23,24:



```python
# Example 6: Lambda with filter() to find odd numbers
def lambda_filter_example():
    print("\nExample 6: Lambda with filter()")
    list_ = [35, 12, 69, 55, 75, 14, 73]
    odd_list = list(filter(lambda num: (num % 2 != 0), list_))
    print('The list of odd numbers is:', odd_list)

# Example 7: Lambda with map() to square numbers
def lambda_map_example():
    print("\nExample 7: Lambda with map() to square numbers")
    numbers_list = [2, 4, 5, 1, 3, 7, 8, 9, 10]
    squared_list = list(map(lambda num: num ** 2, numbers_list))
    print('Square of each number in the given list:', squared_list
        )

# Example 8: Lambda with List Comprehension
def lambda_comprehension_example():
    print("\nExample 8: Lambda with List Comprehension")
    squares = [lambda num=num: num ** 2 for num in range(0, 11)]
    print('The square value of all numbers from 0 to 10:', end=" "
        )
```

```
Choose an example to run (1-10):
Enter the example number (1-10) or 0 to exit: 6

Example 6: Lambda with filter()
The list of odd numbers is: [35, 69, 55, 75, 73]

Choose an example to run (1-10):
Enter the example number (1-10) or 0 to exit: 7

Example 7: Lambda with map() to square numbers
Square of each number in the given list: [4, 16, 25, 1, 9, 49, 64, 81,
    100]

Choose an example to run (1-10):
Enter the example number (1-10) or 0 to exit: 8

Example 8: Lambda with List Comprehension
The square value of all numbers from 0 to 10: 0 1 4 9 16 25 36 49 64 81
    100
Choose an example to run (1-10):
```

25,26:

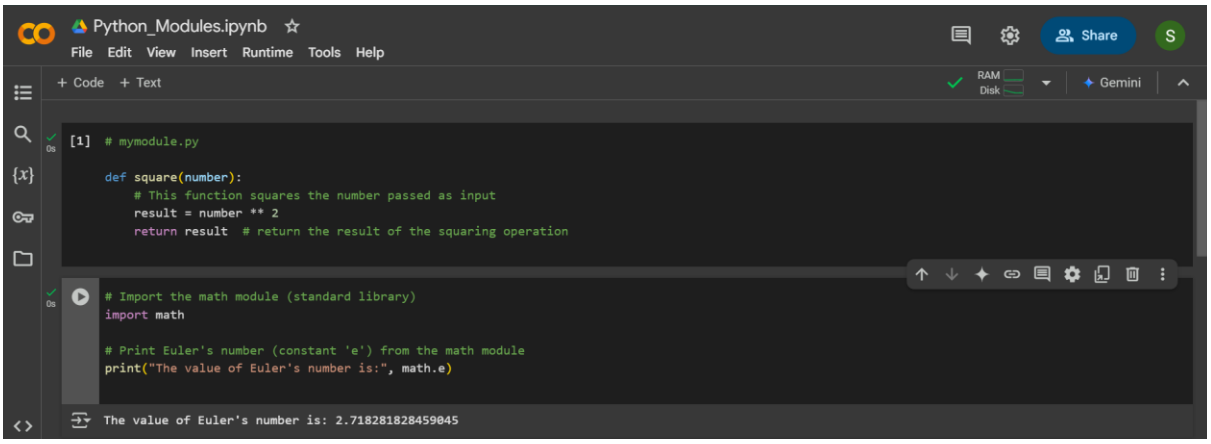


```python

# Example 9: Lambda with if-else
def lambda_if_else_example():
    print("\nExample 9: Lambda Function with if-else")
    Minimum = lambda x, y: x if (x < y) else y
    x = int(input("Enter the first number: "))
    y = int(input("Enter the second number: "))
    print(f'The smaller number is: {Minimum(x, y)}')

# Example 10: Lambda with Multiple Statements (Find third largest)
def lambda_multiple_statements_example():
    print("\nExample 10: Lambda with Multiple Statements")
    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)

```

```
Choose an example to run (1-10):
Enter the example number (1-10) or 0 to exit: 9

Example 9: Lambda Function with if-else
Enter the first number: 11
Enter the second number: 22
The smaller number is: 11

Choose an example to run (1-10):
Enter the example number (1-10) or 0 to exit: 10

Example 10: Lambda with Multiple Statements
The third largest number from every sub-list is: [6, 54, 5]

Choose an example to run (1-10):
Enter the example number (1-10) or 0 to exit:
```

Python Modules

Python import Statement

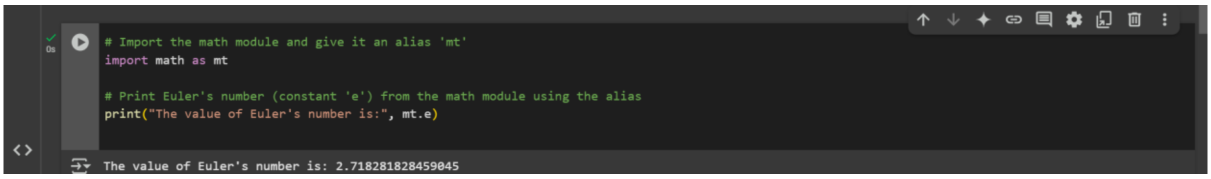


```
[1] # mymodule.py

    def square(number):
        # This function squares the number passed as input
        result = number ** 2
        return result  # return the result of the squaring operation
```

```
# Import the math module (standard library)
import math

# Print Euler's number (constant 'e') from the math module
print("The value of Euler's number is:", math.e)
```

```
The value of Euler's number is: 2.718281828459045
```

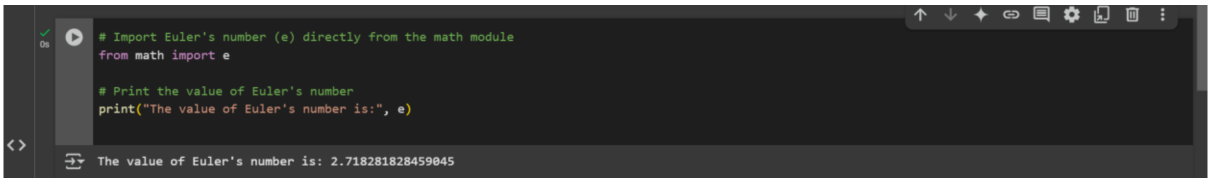
Importing and also Renaming



```
# Import the math module and give it an alias 'mt'
import math as mt

# Print Euler's number (constant 'e') from the math module using the alias
print("The value of Euler's number is:", mt.e)
```

```
The value of Euler's number is: 2.718281828459045
```

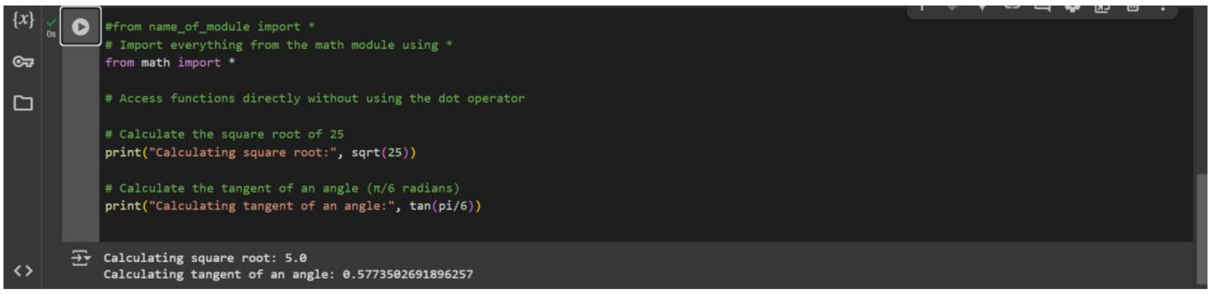
Python from...import Statement



```
# Import Euler's number (e) directly from the math module
from math import e

# Print the value of Euler's number
print("The value of Euler's number is:", e)
```

```
The value of Euler's number is: 2.718281828459045
```

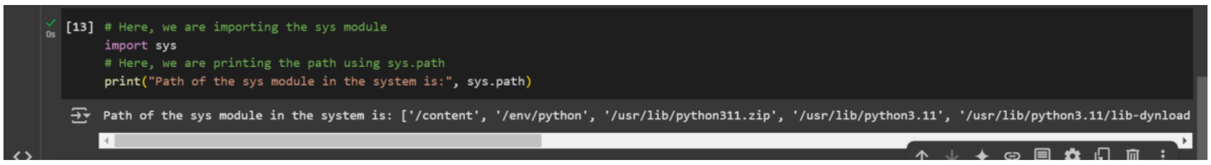
Import all Names - From import * Statement



```
#from name_of_module import *
# Import everything from the math module using *
from math import *

# Access functions directly without using the dot operator

# Calculate the square root of 25
print("Calculating square root:", sqrt(25))

# Calculate the tangent of an angle (π/6 radians)
print("Calculating tangent of an angle:", tan(pi/6))
```

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

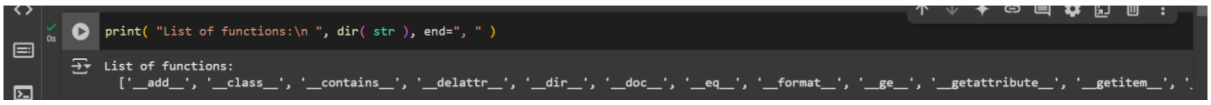
Locating Path of Modules



```
[13] # Here, we are importing the sys module
     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: ['/content', '/env/python', '/usr/lib/python311.zip', '/usr/lib/python3.11', '/usr/lib/python3.11/lib-dynload
```

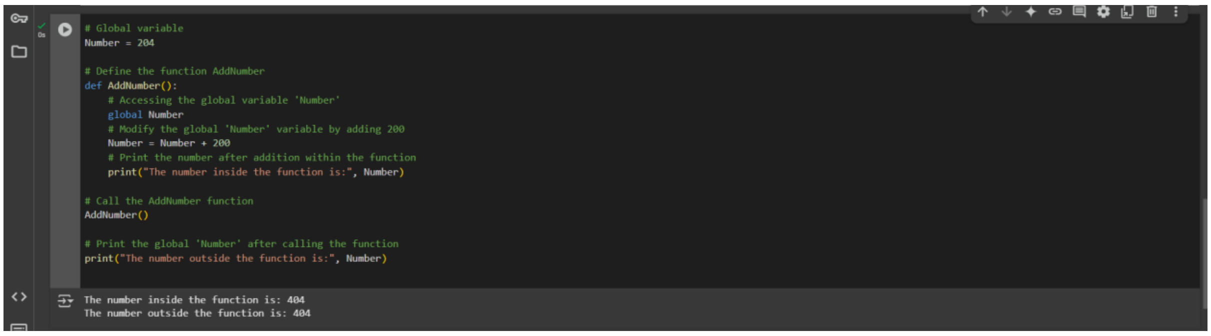
The dir() Built-in Function



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

```
List of functions:
  ['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '
```

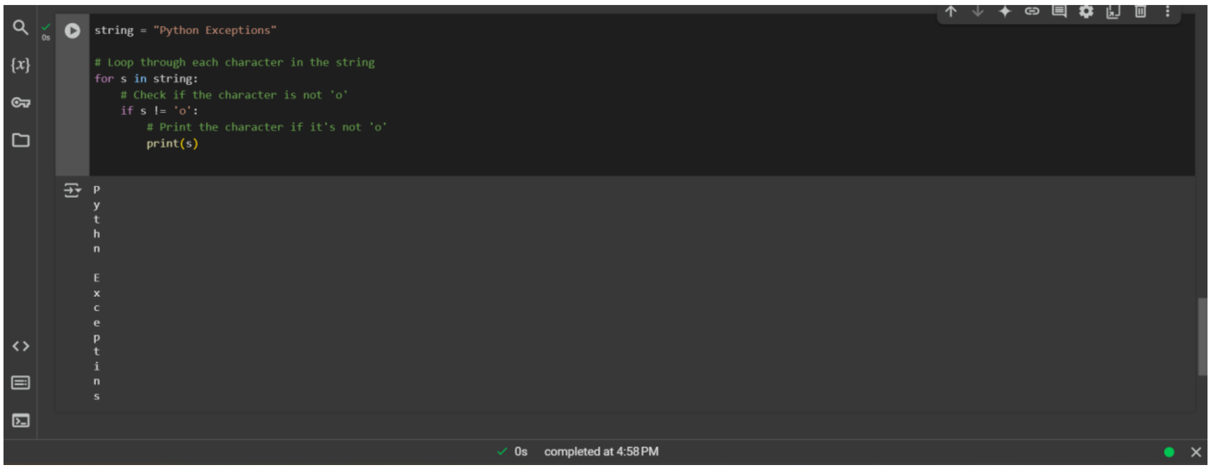
Namespaces and Scoping



```
# Global variable
Number = 204

# Define the function AddNumber
def AddNumber():
    # Accessing the global variable 'Number'
    global Number
    # Modify the global 'Number' variable by adding 200
    Number = Number + 200
    # Print the number after addition within the function
    print("The number inside the function is:", Number)

# Call the AddNumber function
AddNumber()

# Print the global 'Number' after calling the function
print("The number outside the function is:", Number)
```

```
The number inside the function is: 404
The number outside the function is: 404
```

Python Exceptions

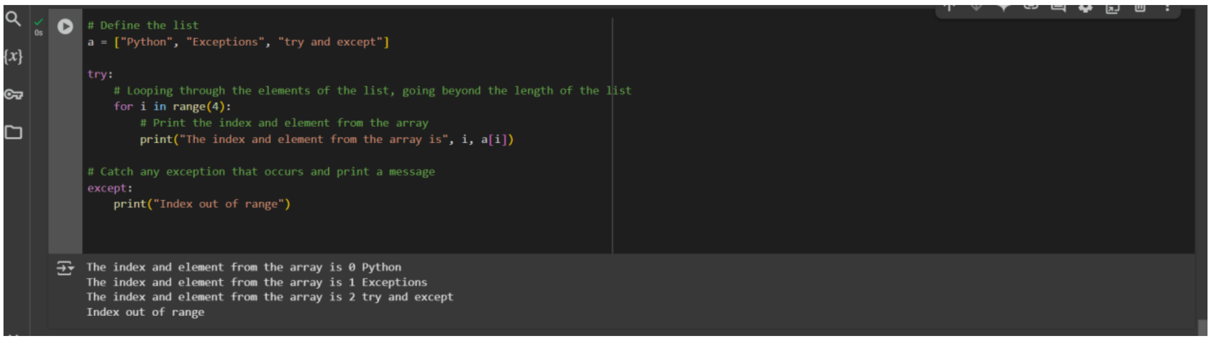


```python
string = "Python Exceptions"

# Loop through each character in the string
for s in string:
    # Check if the character is not 'o'
    if s != 'o':
        # Print the character if it's not 'o'
        print(s)
```

```
P
y
t
h
n

E
x
c
e
p
t
i
n
s
```

0s completed at 4:58 PM

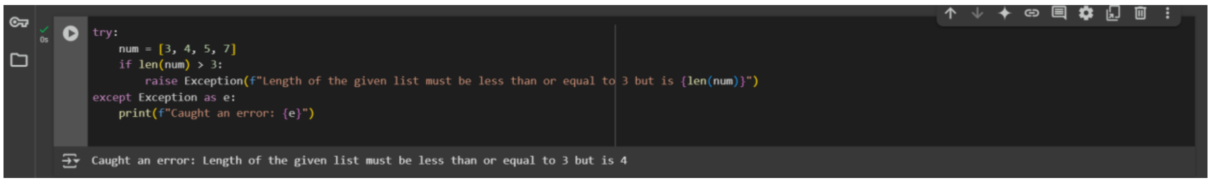
Try and Except Statement - Catching Exceptions



```python
# Define the list
a = ["Python", "Exceptions", "try and except"]

try:
    # Looping through the elements of the list, going beyond the length of the list
    for i in range(4):
        # Print the index and element from the array
        print("The index and element from the array is", i, a[i])

# Catch any exception that occurs and print a message
except:
    print("Index out of range")
```

```
The index and element from the array is 0 Python
The index and element from the array is 1 Exceptions
The index and element from the array is 2 try and except
Index out of range
```

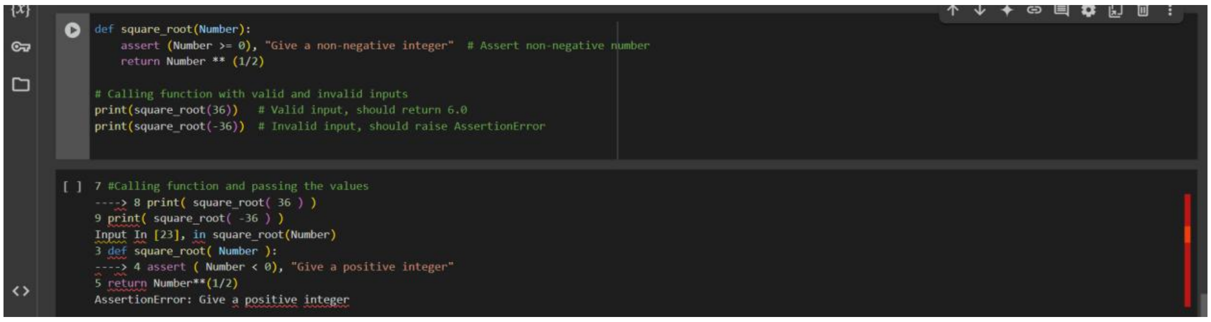
How to Raise an Exception



```python
try:
    num = [3, 4, 5, 7]
    if len(num) > 3:
        raise Exception(f"Length of the given list must be less than or equal to 3 but is {len(num)}")
except Exception as e:
    print(f"Caught an error: {e}")
```

```
Caught an error: Length of the given list must be less than or equal to 3 but is 4
```

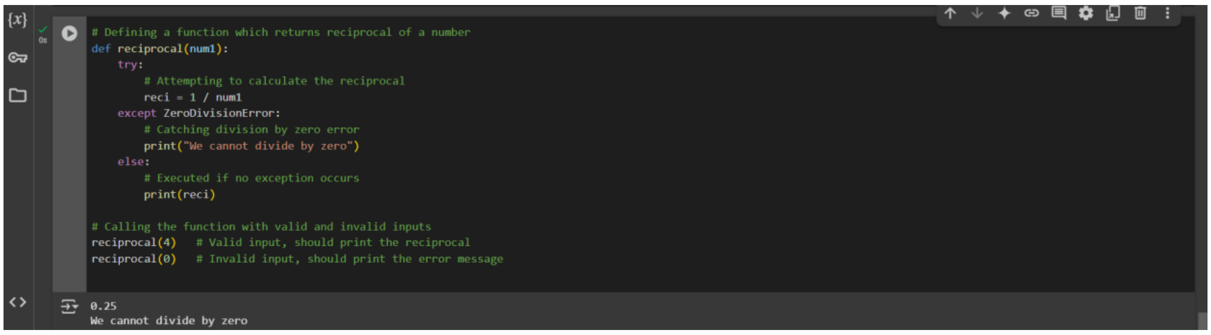
Assertions in Python



```python
def square_root(Number):
    assert (Number >= 0), "Give a non-negative integer"  # Assert non-negative number
    return Number ** (1/2)

# Calling function with valid and invalid inputs
print(square_root(36))   # Valid input, should return 6.0
print(square_root(-36))  # Invalid input, should raise AssertionError
```

```
[ ] 7 #Calling function and passing the values
    ----> 8 print( square_root( 36 ) )
    9 print( square_root( -36 ) )
    Input In [23], in square_root(Number)
    3 def square_root( Number ):
    ----> 4 assert ( Number < 0), "Give a positive integer"
    5 return Number**(1/2)
    AssertionError: Give a positive integer
```

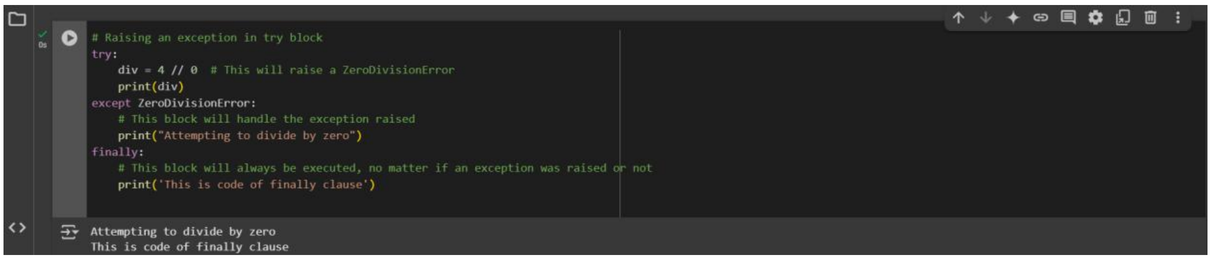
Try with Else Clause



```
# Defining a function which returns reciprocal of a number
def reciprocal(num1):
    try:
        # Attempting to calculate the reciprocal
        reci = 1 / num1
    except ZeroDivisionError:
        # Catching division by zero error
        print("We cannot divide by zero")
    else:
        # Executed if no exception occurs
        print(reci)

# Calling the function with valid and invalid inputs
reciprocal(4)   # Valid input, should print the reciprocal
reciprocal(0)   # Invalid input, should print the error message
```

```
0.25
We cannot divide by zero
```

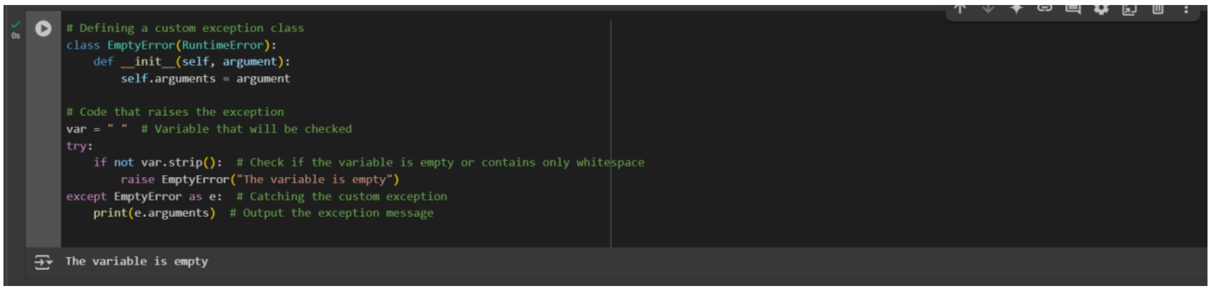
Finally Keyword in Python



```
# Raising an exception in try block
try:
    div = 4 // 0  # This will raise a ZeroDivisionError
    print(div)
except ZeroDivisionError:
    # This block will handle the exception raised
    print("Attempting to divide by zero")
finally:
    # This block will always be executed, no matter if an exception was raised or not
    print('This is code of finally clause')
```

```
Attempting to divide by zero
This is code of finally clause
```

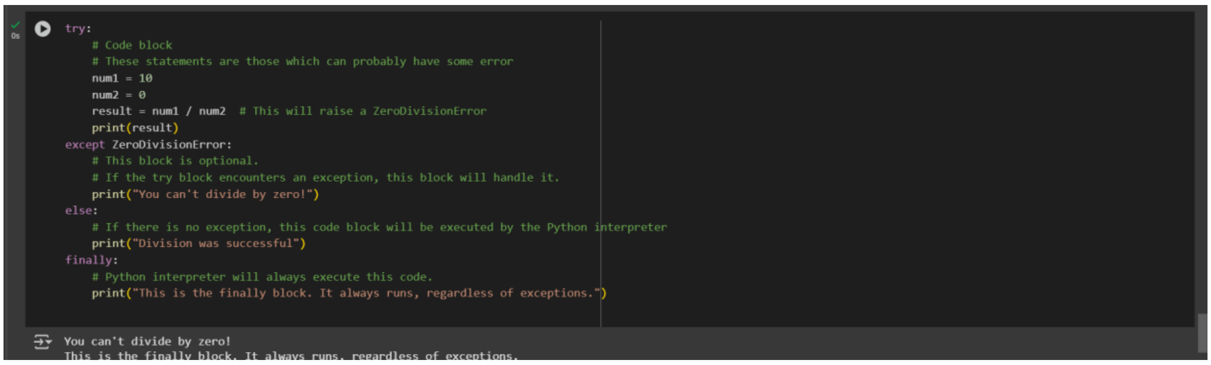
User-Defined Exceptions



```
# Defining a custom exception class
class EmptyError(RuntimeError):
    def __init__(self, argument):
        self.arguments = argument

# Code that raises the exception
var = " "  # Variable that will be checked
try:
    if not var.strip():  # Check if the variable is empty or contains only whitespace
        raise EmptyError("The variable is empty")
except EmptyError as e:  # Catching the custom exception
    print(e.arguments)  # Output the exception message
```

```
The variable is empty
```

try, except, else, and finally clauses

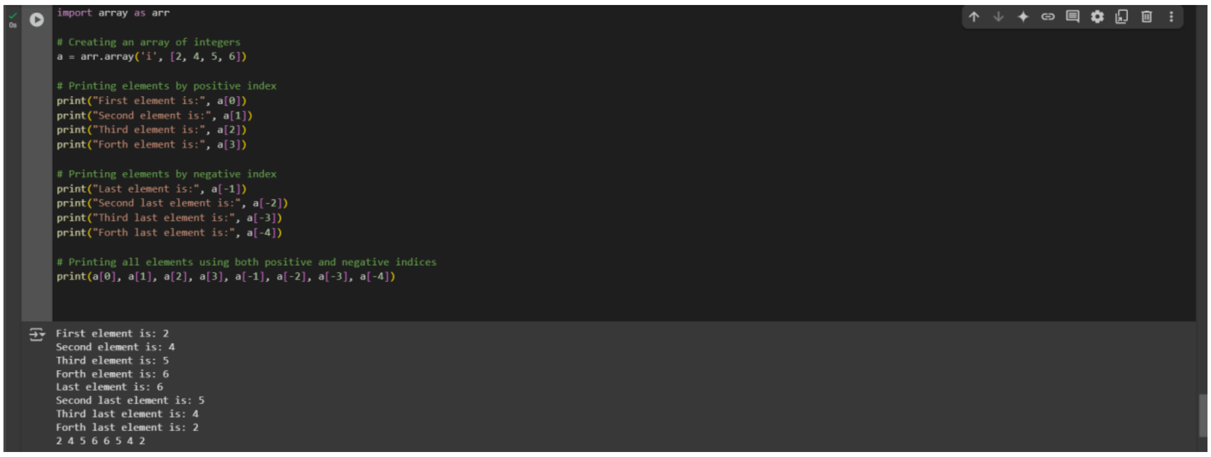


```python
try:
    # Code block
    # These statements are those which can probably have some error
    num1 = 10
    num2 = 0
    result = num1 / num2  # This will raise a ZeroDivisionError
    print(result)
except ZeroDivisionError:
    # This block is optional.
    # If the try block encounters an exception, this block will handle it.
    print("You can't divide by zero!")
else:
    # If there is no exception, this code block will be executed by the Python interpreter
    print("Division was successful")
finally:
    # Python interpreter will always execute this code.
    print("This is the finally block. It always runs, regardless of exceptions.")
```

```
You can't divide by zero!
This is the finally block. It always runs, regardless of exceptions.
```

Python Arrays

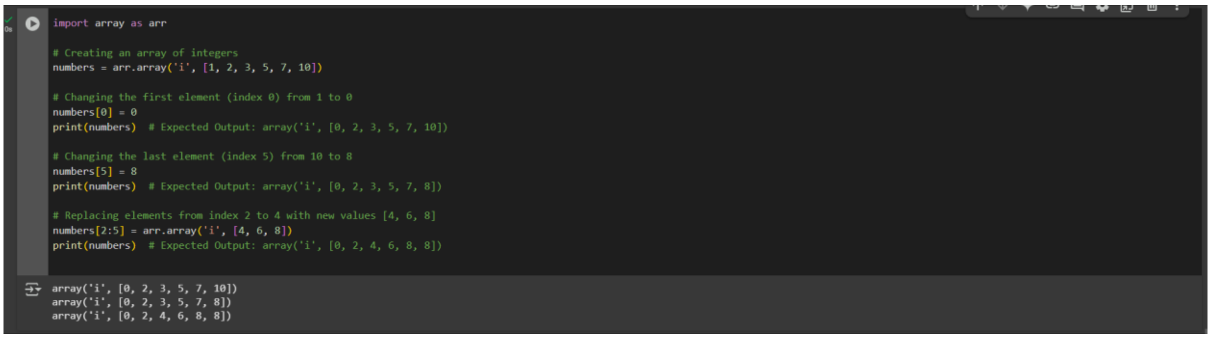
Accessing array element



```python
import array as arr

# Creating an array of integers
a = arr.array('i', [2, 4, 5, 6])

# Printing elements by positive index
print("First element is:", a[0])
print("Second element is:", a[1])
print("Third element is:", a[2])
print("Forth element is:", a[3])

# Printing elements by negative index
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])

# Printing all elements using both positive and negative indices
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
```

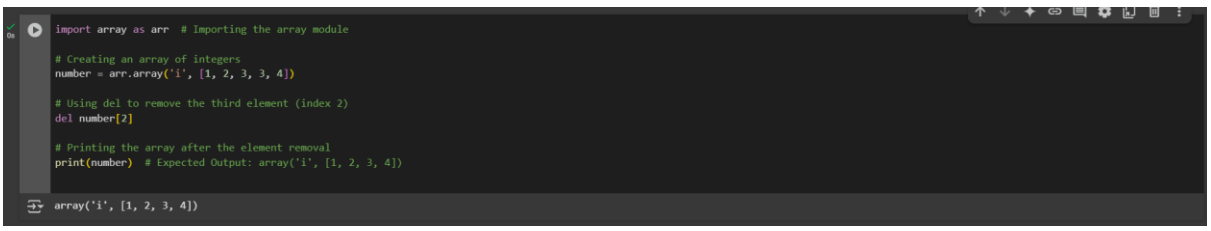
Arrays are mutable, and their elements can be changed similarly to lists.



```python
import array as arr

# Creating an array of integers
numbers = arr.array('i', [1, 2, 3, 5, 7, 10])

# Changing the first element (index 0) from 1 to 0
numbers[0] = 0
print(numbers)  # Expected Output: array('i', [0, 2, 3, 5, 7, 10])

# Changing the last element (index 5) from 10 to 8
numbers[5] = 8
print(numbers)  # Expected Output: array('i', [0, 2, 3, 5, 7, 8])

# Replacing elements from index 2 to 4 with new values [4, 6, 8]
numbers[2:5] = arr.array('i', [4, 6, 8])
print(numbers)  # Expected Output: array('i', [0, 2, 4, 6, 8, 8])
```

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

The elements can be deleted from an array using Python's del statement. If we want to delete any value from the Array, we can use the indices of a particular element.



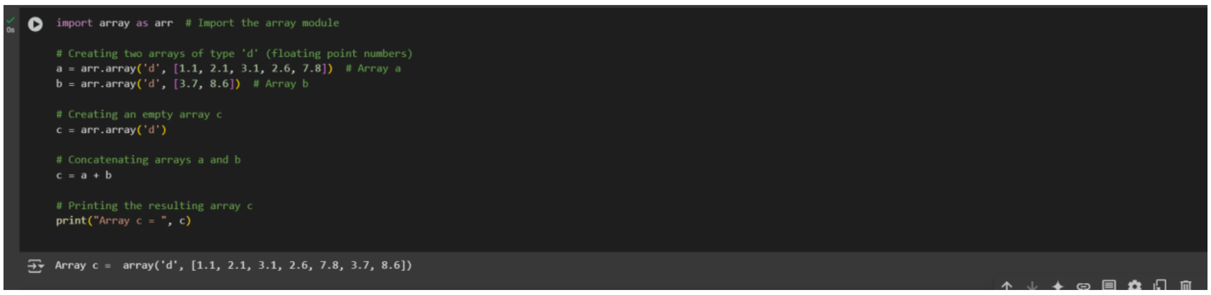
```
import array as arr  # Importing the array module

# Creating an array of integers
number = arr.array('i', [1, 2, 3, 3, 4])

# Using del to remove the third element (index 2)
del number[2]

# Printing the array after the element removal
print(number)  # Expected Output: array('i', [1, 2, 3, 4])
```

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

Array Concatenation

We can easily concatenate any two arrays using the + symbol.

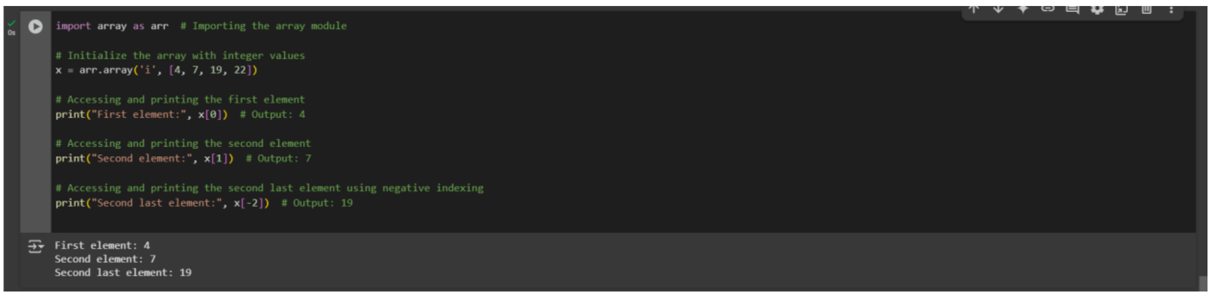
Example 1:



```
import array as arr  # Import the array module

# Creating two arrays of type 'd' (floating point numbers)
a = arr.array('d', [1.1, 2.1, 3.1, 2.6, 7.8])  # Array a
b = arr.array('d', [3.7, 8.6])  # Array b

# Creating an empty array c
c = arr.array('d')

# Concatenating arrays a and b
c = a + b

# Printing the resulting array c
print("Array c = ", c)
```

```
Array c =  array('d', [1.1, 2.1, 3.1, 2.6, 7.8, 3.7, 8.6])
```

Example 2:

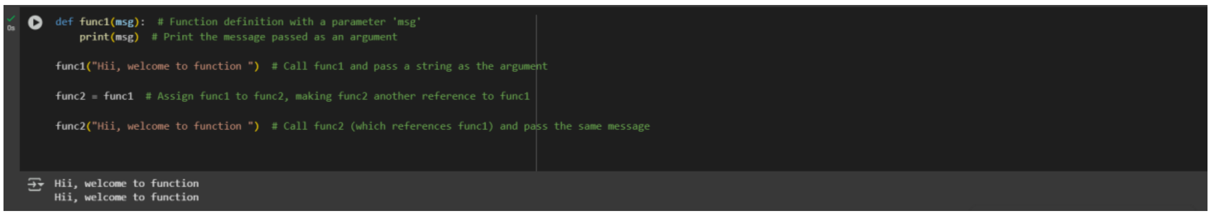


```
import array as arr  # Importing the array module

# Initialize the array with integer values
x = arr.array('i', [4, 7, 19, 22])

# Accessing and printing the first element
print("First element:", x[0])  # Output: 4

# Accessing and printing the second element
print("Second element:", x[1])  # Output: 7

# Accessing and printing the second last element using negative indexing
print("Second last element:", x[-2])  # Output: 19
```

```
First element: 4
Second element: 7
Second last element: 19
```

Python Decorator

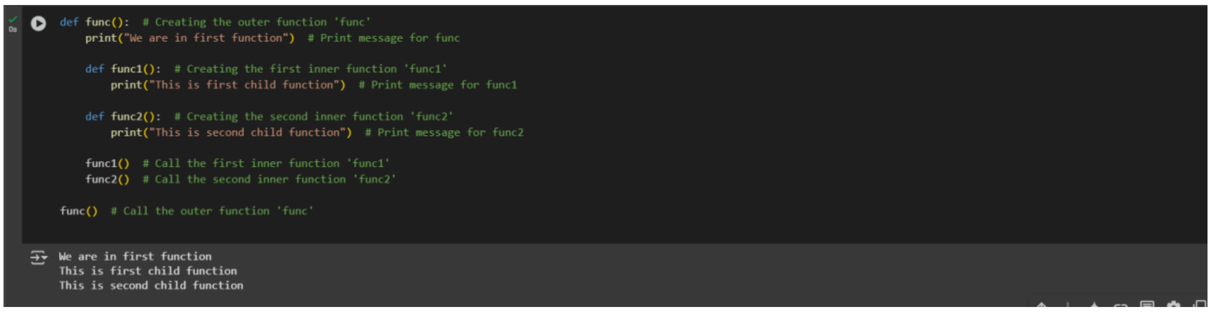
Example



```
def func1(msg):  # Function definition with a parameter 'msg'
    print(msg)  # Print the message passed as an argument

func1("Hii, welcome to function ")  # Call func1 and pass a string as the argument

func2 = func1  # Assign func1 to func2, making func2 another reference to func1

func2("Hii, welcome to function ")  # Call func2 (which references func1) and pass the same message
```

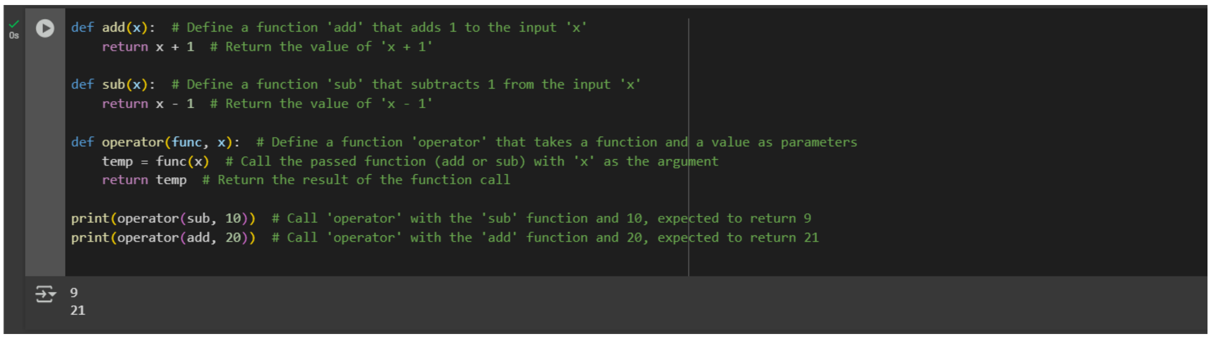
```
Hii, welcome to function
Hii, welcome to function
```

Inner Function



```
def func():  # Creating the outer function 'func'
    print("We are in first function")  # Print message for func

    def func1():  # Creating the first inner function 'func1'
        print("This is first child function")  # Print message for func1

    def func2():  # Creating the second inner function 'func2'
        print("This is second child function")  # Print message for func2

    func1()  # Call the first inner function 'func1'
    func2()  # Call the second inner function 'func2'

func()  # Call the outer function 'func'
```

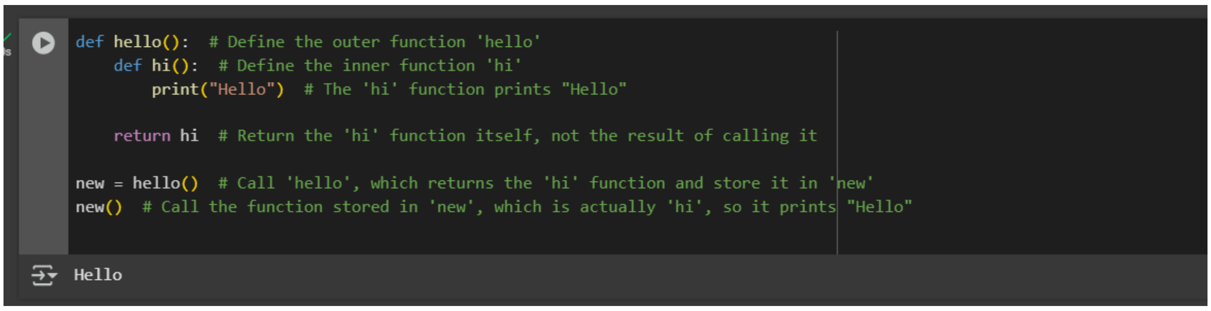
```
We are in first function
This is first child function
This is second child function
```



```
def add(x):  # Define a function 'add' that adds 1 to the input 'x'
    return x + 1  # Return the value of 'x + 1'

def sub(x):  # Define a function 'sub' that subtracts 1 from the input 'x'
    return x - 1  # Return the value of 'x - 1'

def operator(func, x):  # Define a function 'operator' that takes a function and a value as parameters
    temp = func(x)  # Call the passed function (add or sub) with 'x' as the argument
    return temp  # Return the result of the function call

print(operator(sub, 10))  # Call 'operator' with the 'sub' function and 10, expected to return 9
print(operator(add, 20))  # Call 'operator' with the 'add' function and 20, expected to return 21
```

```
9
21
```

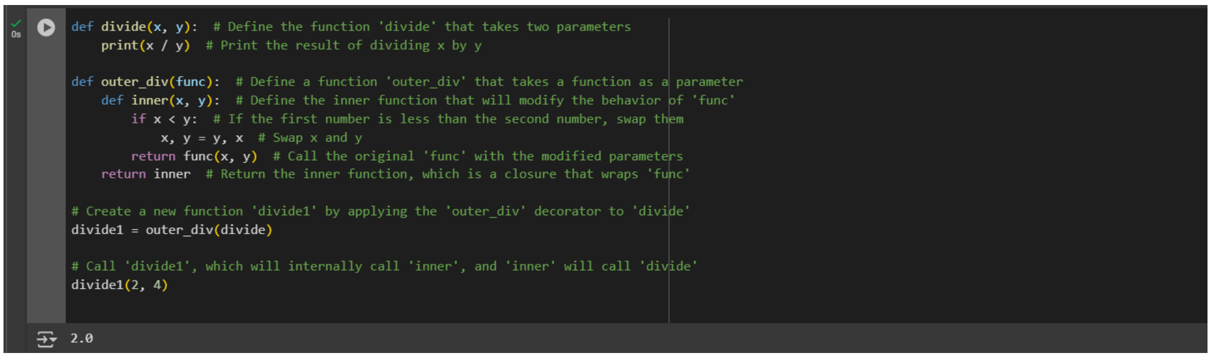
A function can return another function. Consider the below example:



```python
def hello():  # Define the outer function 'hello'
    def hi():  # Define the inner function 'hi'
        print("Hello")  # The 'hi' function prints "Hello"

    return hi  # Return the 'hi' function itself, not the result of calling it

new = hello()  # Call 'hello', which returns the 'hi' function and store it in 'new'
new()  # Call the function stored in 'new', which is actually 'hi', so it prints "Hello"
```

```
Hello
```

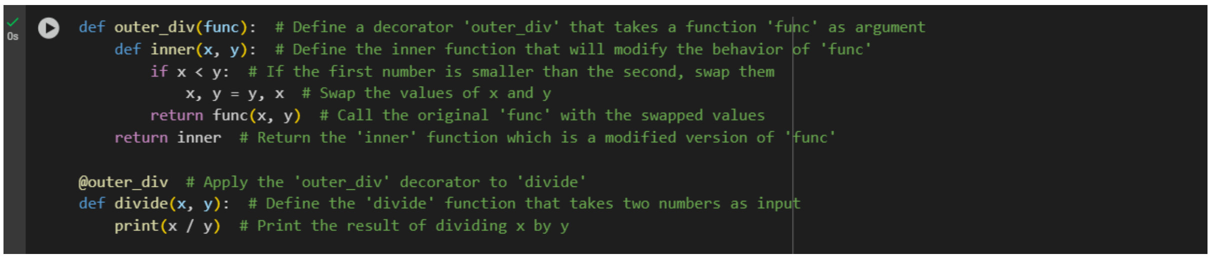
Decorating functions with parameters



```python
def divide(x, y):  # Define the function 'divide' that takes two parameters
    print(x / y)  # Print the result of dividing x by y

def outer_div(func):  # Define a function 'outer_div' that takes a function as a parameter
    def inner(x, y):  # Define the inner function that will modify the behavior of 'func'
        if x < y:  # If the first number is less than the second number, swap them
            x, y = y, x  # Swap x and y
        return func(x, y)  # Call the original 'func' with the modified parameters
    return inner  # Return the inner function, which is a closure that wraps 'func'

# Create a new function 'divide1' by applying the 'outer_div' decorator to 'divide'
divide1 = outer_div(divide)

# Call 'divide1', which will internally call 'inner', and 'inner' will call 'divide'
divide1(2, 4)
```

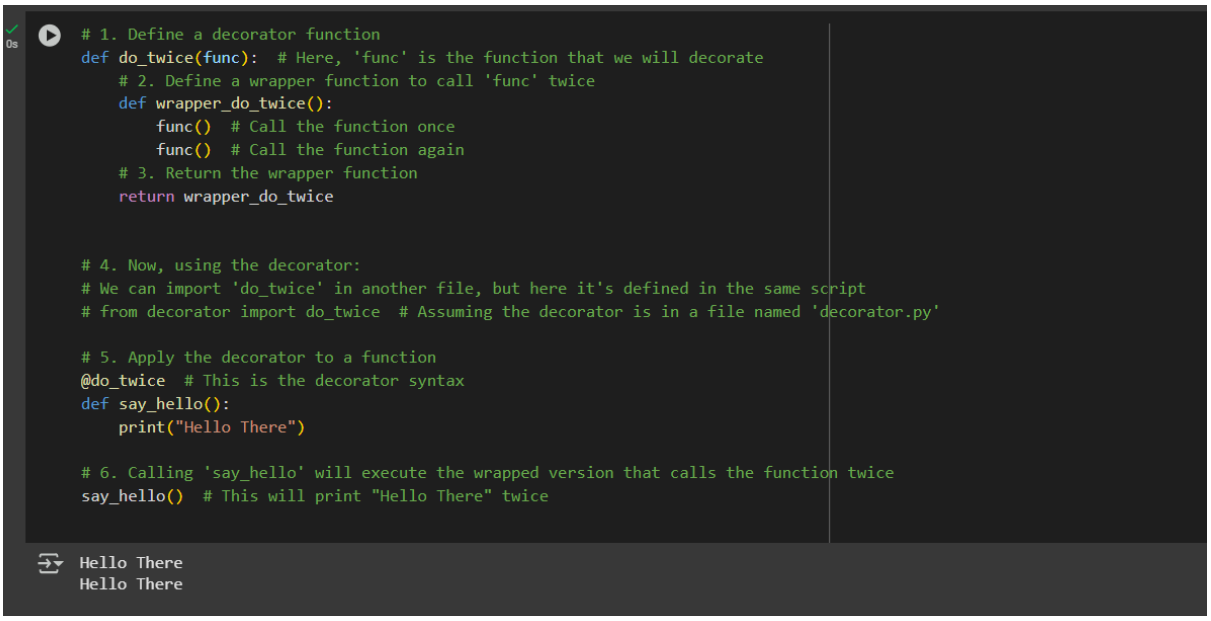
```
2.0
```

Syntactic Decorator



```python
def outer_div(func):  # Define a decorator 'outer_div' that takes a function 'func' as argument
    def inner(x, y):  # Define the inner function that will modify the behavior of 'func'
        if x < y:  # If the first number is smaller than the second, swap them
            x, y = y, x  # Swap the values of x and y
        return func(x, y)  # Call the original 'func' with the swapped values
    return inner  # Return the 'inner' function which is a modified version of 'func'

@outer_div  # Apply the 'outer_div' decorator to 'divide'
def divide(x, y):  # Define the 'divide' function that takes two numbers as input
    print(x / y)  # Print the result of dividing x by y
```

Reusing Decorator

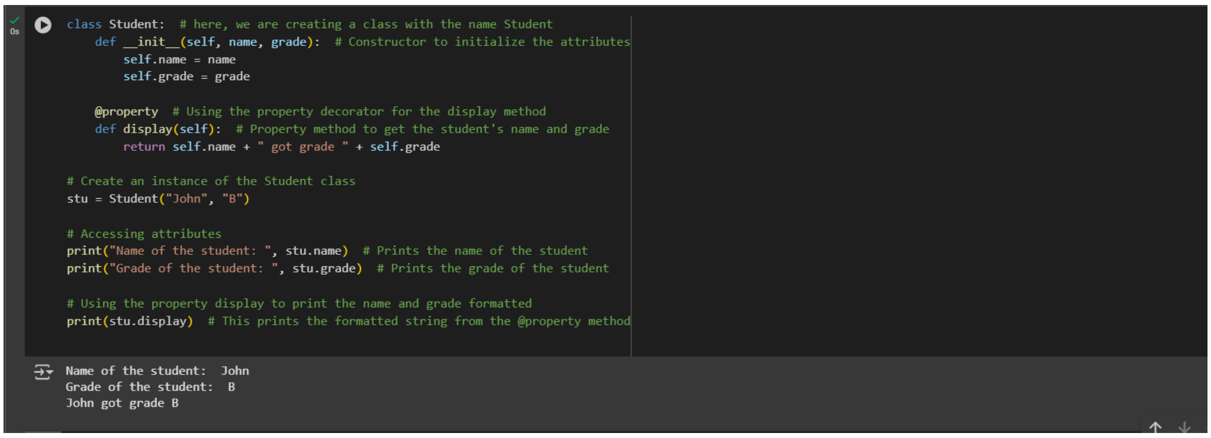


```python
# 1. Define a decorator function
def do_twice(func):  # Here, 'func' is the function that we will decorate
    # 2. Define a wrapper function to call 'func' twice
    def wrapper_do_twice():
        func()  # Call the function once
        func()  # Call the function again
    # 3. Return the wrapper function
    return wrapper_do_twice


# 4. Now, using the decorator:
# We can import 'do_twice' in another file, but here it's defined in the same script
# from decorator import do_twice  # Assuming the decorator is in a file named 'decorator.py'

# 5. Apply the decorator to a function
@do_twice  # This is the decorator syntax
def say_hello():
    print("Hello There")

# 6. Calling 'say_hello' will execute the wrapped version that calls the function twice
say_hello()  # This will print "Hello There" twice
```

```
Hello There
Hello There
```

Fancy Decorators

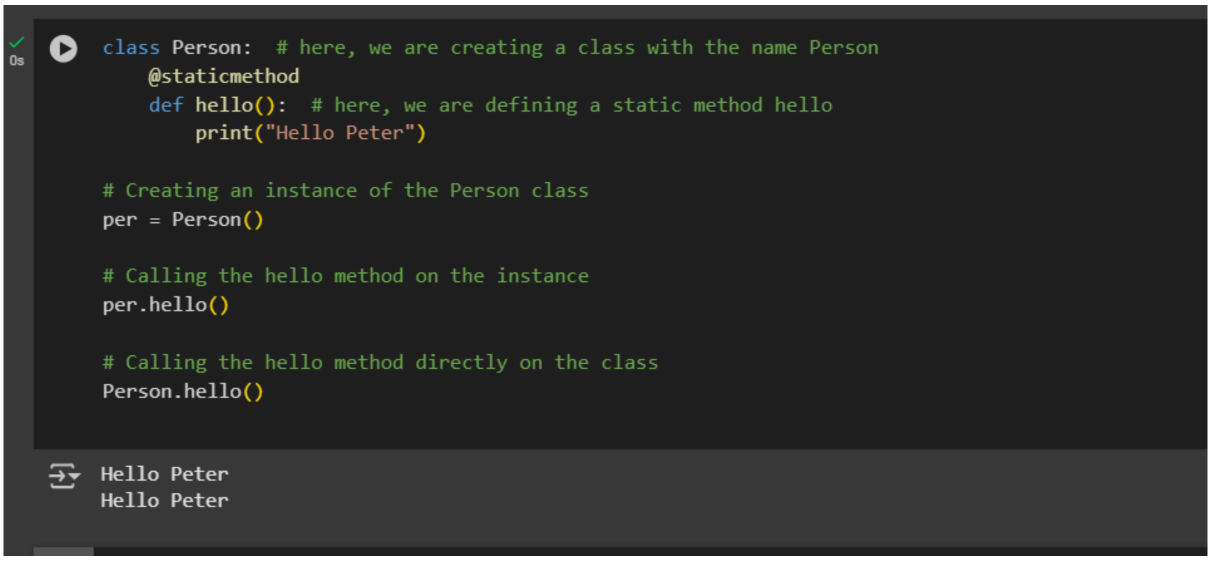
Example: 1



```python
class Student:  # here, we are creating a class with the name Student
    def __init__(self, name, grade):  # Constructor to initialize the attributes
        self.name = name
        self.grade = grade

    @property  # Using the property decorator for the display method
    def display(self):  # Property method to get the student's name and grade
        return self.name + " got grade " + self.grade

# Create an instance of the Student class
stu = Student("John", "B")

# Accessing attributes
print("Name of the student: ", stu.name)  # Prints the name of the student
print("Grade of the student: ", stu.grade)  # Prints the grade of the student

# Using the property display to print the name and grade formatted
print(stu.display)  # This prints the formatted string from the @property method
```

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

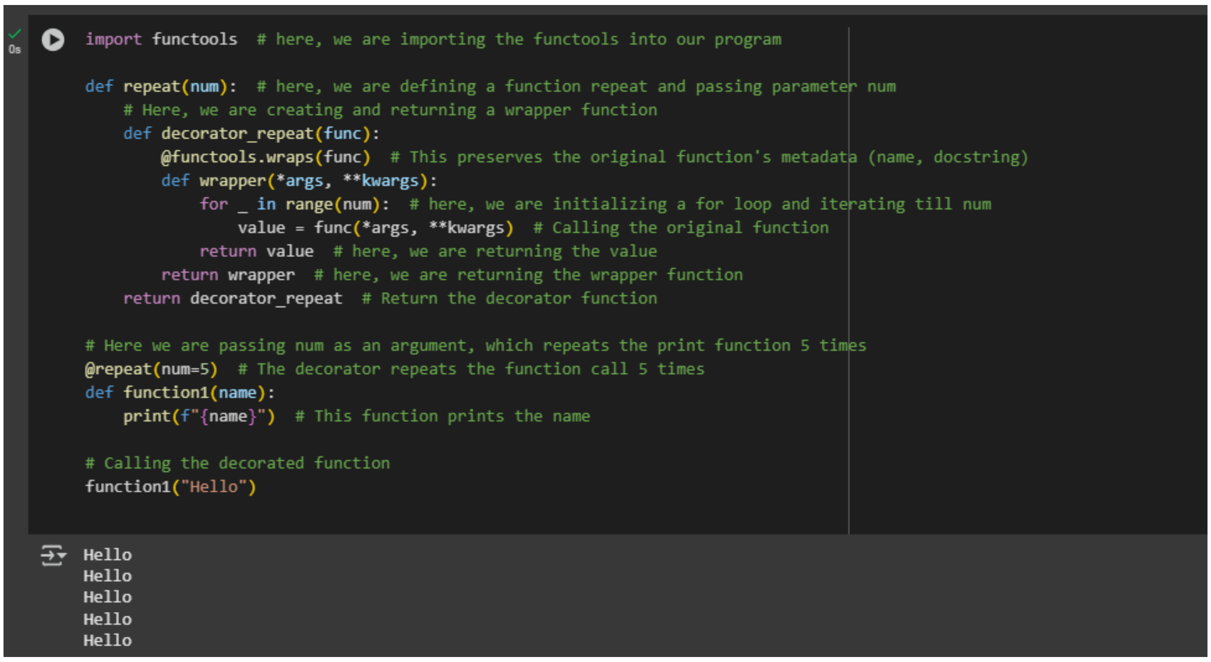
Example: 2-



```python
class Person:  # here, we are creating a class with the name Person
    @staticmethod
    def hello():  # here, we are defining a static method hello
        print("Hello Peter")

# Creating an instance of the Person class
per = Person()

# Calling the hello method on the instance
per.hello()

# Calling the hello method directly on the class
Person.hello()
```

```
Hello Peter
Hello Peter
```

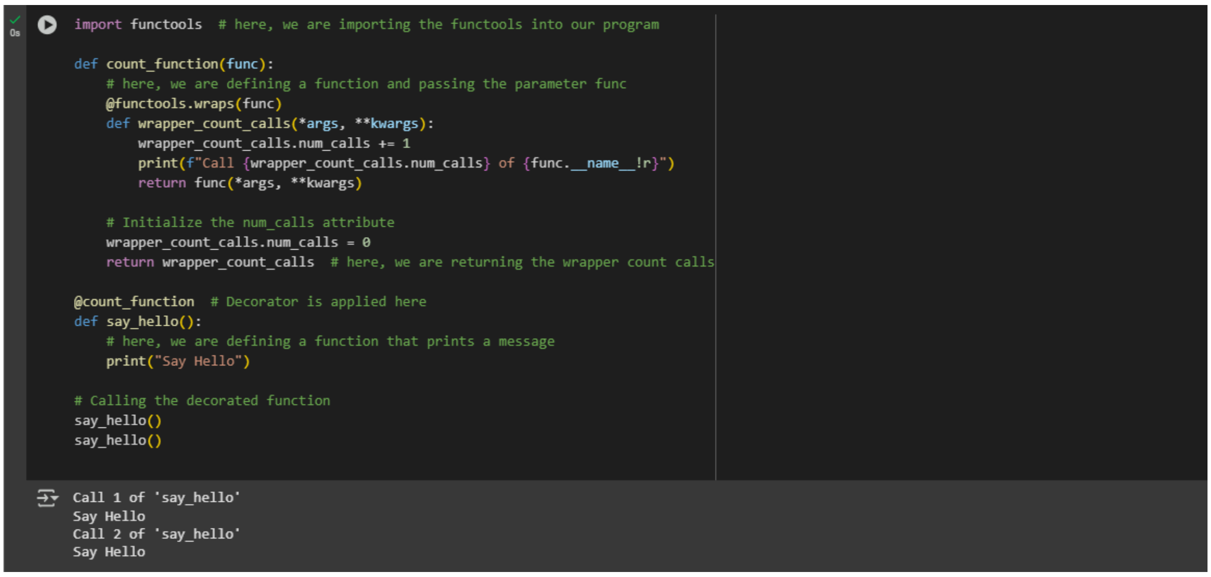
Decorator with Arguments



```python
import functools  # here, we are importing the functools into our program

def repeat(num):  # here, we are defining a function repeat and passing parameter num
    # Here, we are creating and returning a wrapper function
    def decorator_repeat(func):
        @functools.wraps(func)  # This preserves the original function's metadata (name, docstring)
        def wrapper(*args, **kwargs):
            for _ in range(num):  # here, we are initializing a for loop and iterating till num
                value = func(*args, **kwargs)  # Calling the original function
            return value  # here, we are returning the value
        return wrapper  # here, we are returning the wrapper function
    return decorator_repeat  # Return the decorator function

# Here we are passing num as an argument, which repeats the print function 5 times
@repeat(num=5)  # The decorator repeats the function call 5 times
def function1(name):
    print(f"{name}")  # This function prints the name

# Calling the decorated function
function1("Hello")
```

```
Hello
Hello
Hello
Hello
Hello
```

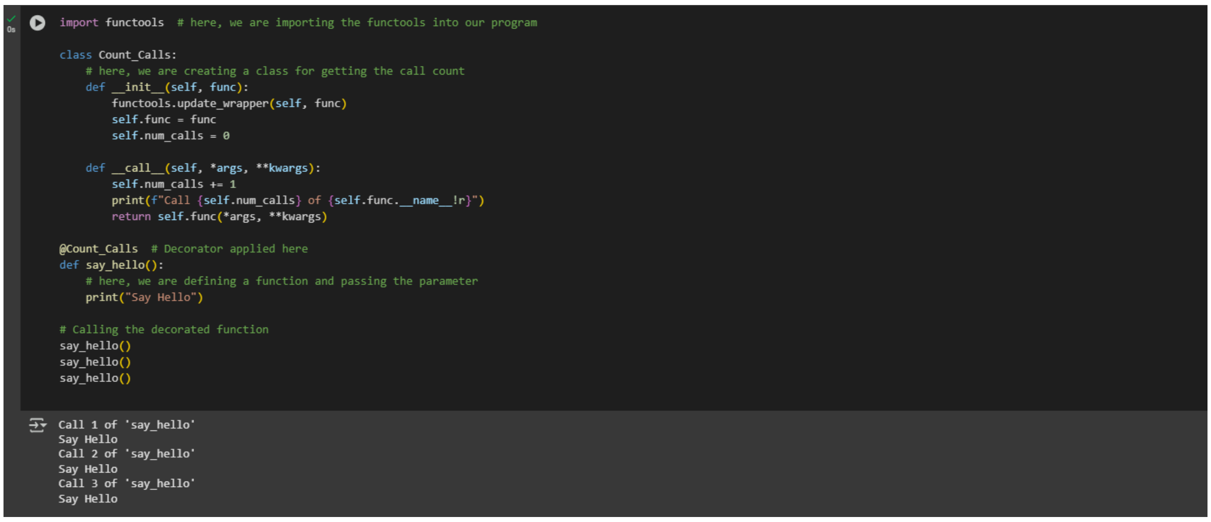
Stateful Decorators



```python
import functools  # here, we are importing the functools into our program

def count_function(func):
    # here, we are defining a function and passing the parameter func
    @functools.wraps(func)
    def wrapper_count_calls(*args, **kwargs):
        wrapper_count_calls.num_calls += 1
        print(f"Call {wrapper_count_calls.num_calls} of {func.__name__!r}")
        return func(*args, **kwargs)

    # Initialize the num_calls attribute
    wrapper_count_calls.num_calls = 0
    return wrapper_count_calls  # here, we are returning the wrapper count calls

@count_function  # Decorator is applied here
def say_hello():
    # here, we are defining a function that prints a message
    print("Say Hello")

# Calling the decorated function
say_hello()
say_hello()
```

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

Classes as Decorators



```python
import functools  # here, we are importing the functools into our program

class Count_Calls:
    # here, we are creating a class for getting the call count
    def __init__(self, func):
        functools.update_wrapper(self, func)
        self.func = func
        self.num_calls = 0

    def __call__(self, *args, **kwargs):
        self.num_calls += 1
        print(f"Call {self.num_calls} of {self.func.__name__!r}")
        return self.func(*args, **kwargs)

@Count_Calls  # Decorator applied here
def say_hello():
    # here, we are defining a function and passing the parameter
    print("Say Hello")

# Calling the decorated function
say_hello()
say_hello()
say_hello()
```

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