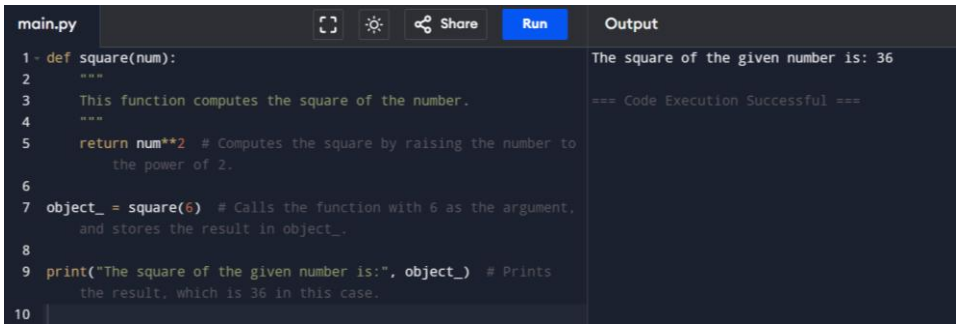
PYTHON PROJECT

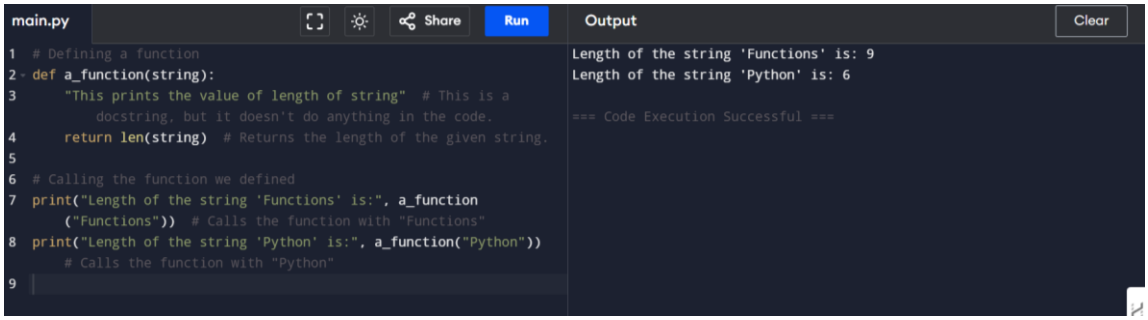
Python Functions

A collection of related assertions that carry out a mathematical, analytical, or evaluative operation is known as a function. An assortment of proclamations called Python Capabilities returns the specific errand. Python functions are necessary for intermediate-level programming and are easy to define. Function names meet the same standards as variable names do. The objective is to define a function and group-specific frequently performed actions. Instead of repeatedly creating the same code block for various input variables, we can call the function and reuse the code it contains with different variables.

Illustration of a User-Defined Function



Calling a Function



Pass by Reference vs. Pass by Value

main.py

```python
# defining the function
def square(item_list):
    '''This function will find the square of items in the list'''
    squares = []  # Create an empty list to store the squares.
    for l in item_list:  # Loop through each element in the input list.
        squares.append(l**2)  # Square the element and append it to the 'squares' list.
    return squares  # Return the list of squared elements.

# calling the defined function
my_list = [17, 52, 8]  # Initial list.
my_result = square(my_list)  # Pass the list to the function and store the result in 'my_result'.

print("Squares of the list are:", my_result)  # Print the resulting list of squares.
```

Output

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

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

Function Arguments

1) Default Arguments

main.py

```python
# defining a function
def function(n1, n2=20):  # The second argument n2 has a default value of 20.
    print("number 1 is:", n1)  # Prints the value of n1
    print("number 2 is:", n2)  # Prints the value of n2

# Calling the function and passing only one argument
print("Passing only one argument")
function(30)  # Calls the function with n1=30 and uses the default value for n2 (20)

# Now giving two arguments to the function
print("Passing two arguments")
function(50, 30)  # Calls the function with n1=50 and n2=30 (no default value used for n2)
```

Output

```
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 ===
```

2) Keyword Arguments

main.py

```python
# Defining a function
def function(n1, n2):  # The function takes two parameters: n1 and n2
    print("number 1 is:", n1)  # Prints the value of n1
    print("number 2 is:", n2)  # Prints the value of n2

# Calling function and passing arguments without using keyword
print("Without using keyword")
function(50, 30)  # This is a positional argument call, passing 50 for n1 and 30 for n2.

# Calling function and passing arguments using keyword
print("With using keyword")
function(n2=50, n1=30)  # This is a keyword argument call, specifying which value goes to which parameter.
```

Output

```
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 ===
```

3) Required Arguments

main.py

```python
# Defining a function
def function(n1, n2):  # Function accepts two arguments: n1 and n2
    print("number 1 is:", n1)  # Prints the value of n1
    print("number 2 is:", n2)  # Prints the value of n2

# Calling function and passing two arguments out of order, we need num1 to be 20 and num2 to be 30
print("Passing out of order arguments")
function(30, 20)  # Positional arguments: n1 gets 30, n2 gets 20

# Calling function and passing only one argument
print("Passing only one argument")
try:
    function(30)  # Only passing one argument; this will raise an error because function needs two arguments
except:
    print("Function needs two positional arguments")  # Handle the error when the second argument is missing
```

Output

```
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 ===
```

4) Variable-Length Arguments

main.py

```python
# Defining a function using *args (variable-length positional arguments)
def function(*args_list):
    ans = []  # List to store uppercase words
    for l in args_list:  # Loop through each element in args_list
        ans.append(l.upper())  # Convert each element to uppercase and append to ans
    return ans  # Return the list of uppercase words

# Passing *args arguments
object = function('Python', 'Functions', 'tutorial')  # Pass three words
print(object)  # Output will be ['PYTHON', 'FUNCTIONS', 'TUTORIAL']

# Defining a function using **kwargs (variable-length keyword arguments)
def function(**kargs_list):
    ans = []  # List to store key-value pairs
    for key, value in kargs_list.items():  # Loop through each key-value pair
        ans.append([key, value])  # Append the key-value pair to ans as a list
    return ans  # Return the list of key-value pairs

# Passing **kwargs arguments
object = function(First="Python", Second="Functions", Third="Tutorial")  # Pass keyword arguments
print(object)  # Output will be [['First', 'Python'], ['Second', 'Functions'], ['Third', 'Tutorial']]
```

Output

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

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

return Statement

main.py

```python
# Defining a function with return statement
def square(num):
    return num**2  # Returns the square of the number

# Calling function and passing arguments.
print("With return statement")
print(square(52))  # This will print the square of 52

# Defining a function without return statement
def square(num):
    num**2  # This will calculate the square but doesn't return it

# Calling function and passing arguments.
print("Without return statement")
print(square(52))  # This will print 'None' because the function doesn't return a value
```

Output

```
With return statement
2704
Without return statement
None

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

The Anonymous Functions

main.py

```python
# Defining a lambda function
lambda_ = lambda argument1, argument2: argument1 + argument2

# Calling the lambda function and passing values
print("Value of the function is: ", lambda_(20, 30))  # The sum of 20 and 30
print("Value of the function is: ", lambda_(40, 50))  # The sum of 40 and 50
```

Output

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

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

Scope and Lifetime of Variables

main.py

```python
# Defining a function to print a number
def number():
    num = 50  # Local variable
    print("Value of num inside the function:", num)

# Global variable
num = 10

# Calling the function
number()

# Accessing the global variable outside the function
print("Value of num outside the function:", num)
```

Output

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

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

Python Capability inside Another Capability

main.py

```python
# Defining a nested function
def word():
    string = 'Python functions tutorial'  # Outer variable
    x = 5  # Outer variable

    # Defining an inner function
    def number():
        print(string)  # Accessing the outer variable 'string'
        print(x)  # Accessing the outer variable 'x'

    # Calling the inner function
    number()

# Calling the outer function
word()
```

Output

```
Python functions tutorial
5

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

Python Built-in Functions

Python abs() Function Example

main.py

```python
# Integer number
integer = -20
print('Absolute value of -20 is:', abs(integer))

# Floating number
floating = -20.83
print('Absolute value of -20.83 is:', abs(floating))
```

Output

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

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

Python all() Function

main.py

```python
# All values true
k = [1, 3, 4, 6]
print(all(k))  # All values are truthy, so it returns True

# All values false
k = [0, False]
print(all(k))  # All values are falsy, so it returns False

# One false value
k = [1, 3, 7, 0]
print(all(k))  # Contains a falsy value (0), so it returns False

# One true value
k = [0, False, 5]
print(all(k))  # Contains a truthy value (5), so it returns False

# Empty iterable
k = []
print(all(k))  # Empty iterable, so it returns True
```

Output

```
True
False
False
False
True

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

Python bin() Function

main.py

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

Output

```
0b1010

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

Python bool()

main.py

```python
test1 = []
print(test1, 'is', bool(test1))  # Empty list is falsy

test1 = [0]
print(test1, 'is', bool(test1))  # Non-empty list is truthy, even if it contains 0

test1 = 0.0
print(test1, 'is', bool(test1))  # 0.0 is falsy

test1 = None
print(test1, 'is', bool(test1))  # None is falsy

test1 = True
print(test1, 'is', bool(test1))  # True is truthy

test1 = 'Easy string'
print(test1, 'is', bool(test1))  # Non-empty string is truthy
```

Output

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

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

Python bytes()

main.py

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

Output

```
b'Hello World.'

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

Python callable() Function

main.py

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

Output

```
False

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

Python compile() Function

main.py

```python
# compile string source to code
code_str = 'x=5\ny=10\nprint("sum =",x+y)'

# compile the string into a code object
code = compile(code_str, 'sum.py', 'exec')

# Print the type of the code object
print(type(code))  # This will print <class 'code'>, as it's a code object

# Execute the compiled code
exec(code)  # This will print: sum = 15

# Attempting to execute variable 'x' will raise an error because 'x' is an integer
# exec(x)  # This line will result in a TypeError as 'x' is not callable
```

Output

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

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

Python exec() Function

main.py

```python
x = 8
exec('print(x==8)')  # Checks if x is equal to 8 and prints the result (True)
exec('print(x+4)')  # Adds 4 to x and prints the result (12)
```

Output

```
True
12

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

Python sum() Function

main.py

```python
s = sum([1, 2, 4])
print(s)  # This prints the sum of the list: 7

s = sum([1, 2, 4], 10)
print(s)  # This prints the sum of the list plus the initial value: 17
```

Output

```
7
17

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

Python any() Function

main.py

```python
l = [4, 3, 2, 0]
print(any(l))  # Checks if any element is truthy

l = [0, False]
print(any(l))  # Checks if any element is truthy

l = [0, False, 5]
print(any(l))  # Checks if any element is truthy

l = []
print(any(l))  # Checks if any element is truthy
```

Output

```
True
False
True
False

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

Python ascii() Function

main.py

```python
normalText = 'Python is interesting'
print(ascii(normalText))  # Prints the string as is, since it only contains ASCII characters.

otherText = 'Pyth\xffdn is interesting'
print(ascii(otherText))  # Prints the string with non-ASCII characters replaced by escape sequences.

print('Pyth\xf6n is interesting')  # Directly prints the string with the non-ASCII character (ö).
```

Output

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

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

Python bytearray()

main.py

```python
string = "Python is a programming language."

# string with encoding 'utf-8'
arr = bytearray(string, 'utf-8')
print(arr)
```

Output

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

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

Python eval() Function

main.py

```python
x = 8
print(eval('x + 1'))  # Evaluates the expression 'x + 1' and prints the result (9)
```

Output

```
9

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

Python float()

main.py

```python
# for integers
print(float(9))  # Converts integer 9 to float 9.0

# for floats
print(float(8.19))  # Converts the float 8.19 to float 8.19

# for string floats
print(float("-24.27"))  # Converts string "-24.27" to float -24.27

# for string floats with whitespaces
print(float(" -17.19\n"))  # Converts string " -17.19\n" to float -17.19 (ignores whitespace and newline)

# string float error
print(float("xyz"))  # Raises ValueError as "xyz" is not a valid number
```

Output

```
9.0
8.19
-24.27
-17.19
ERROR!
Traceback (most recent call last):
  File "<main.py>", line 14, in <module>
ValueError: could not convert string to float: 'xyz'

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

Python format() Function

main.py

```python
# integer
print(format(123, "d"))  # Formats the integer 123 as a decimal (default format)

# float arguments
print(format(123.4567898, "f"))  # Formats the float 123.4567898 as a fixed-point number

# binary format
print(format(12, "b"))  # Converts the integer 12 to its binary representation

```

Output

```
123
123.456790
1100

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

Python frozenset()

main.py

```python
# tuple of letters
letters = ('m', 'r', 'o', 't', 's')

# Create a frozenset from the tuple
fSet = frozenset(letters)

# Print the frozenset
print('Frozen set is:', fSet)

# Print an empty frozenset
print('Empty frozen set is:', frozenset())

```

Output

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

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

Python getattr() Function

main.py

```python
class Details:
    age = 22
    name = "Phill"

# Create an object of the Details class
details = Details()

# Accessing the 'age' attribute using getattr
print('The age is:', getattr(details, "age"))

# Accessing the 'age' attribute using direct access
print('The age is:', details.age)

```

Output

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

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

Python globals() Function

main.py

```python
age = 22

# Setting age through globals() function
globals()['age'] = 22

# Printing the value of age
print('The age is:', age)

```

Output

```
The age is: 22

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

Python hasattr() Function

main.py

```python
l = [4, 3, 2, 0]
print(any(l))  # Checking if any value in the list is truthy

l = [0, False]
print(any(l))  # Checking if any value in the list is truthy

l = [0, False, 5]
print(any(l))  # Checking if any value in the list is truthy

l = []
print(any(l))  # Checking if any value in the empty list is truthy
```

Output

```
True
False
True
False

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

Python iter() Function

main.py

```python
# list of numbers
list = [1, 2, 3, 4, 5]

# create an iterator from the list
listIter = iter(list)

# prints '1'
print(next(listIter))

# prints '2'
print(next(listIter))

# prints '3'
print(next(listIter))

# prints '4'
print(next(listIter))

# prints '5'
print(next(listIter))
```

Output

```
1
2
3
4
5

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

Python list()

main.py

```python
# empty list
print(list())  # Output: []

# string
String = 'abcde'
print(list(String))  # Output: ['a', 'b', 'c', 'd', 'e']

# tuple
Tuple = (1, 2, 3, 4, 5)
print(list(Tuple))  # Output: [1, 2, 3, 4, 5]

# list
List = [1, 2, 3, 4, 5]
print(list(List))  # Output: [1, 2, 3, 4, 5]
```

Output

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

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

Python locals() Function

main.py

```python
def localsAbsent():
    return locals()  # Returns a dictionary of local variables in the current scope (function)

def localsPresent():
    present = True  # Define a local variable 'present'
    return locals()  # Returns a dictionary of local variables, including 'present'

print('localsNotPresent:', localsAbsent())  # Prints the dictionary of local variables in 'localsAbsent'
print('localsPresent:', localsPresent())   # Prints the dictionary of local variables in 'localsPresent'
```

Output

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

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

Python map() Function

main.py

```python
def calculateAddition(n):
    return n + n  # This function doubles the input number

numbers = (1, 2, 3, 4)  # A tuple of numbers

# Applying calculateAddition function to each element in the numbers tuple using map
result = map(calculateAddition, numbers)

# Print the map object (which is an iterator)
print(result)

# Convert the map object to a set and print the result
numbersAddition = set(result)
print(numbersAddition)
```

Output

```
<map object at 0x7a3795b31450>
{8, 2, 4, 6}

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

Python memoryview() Function

main.py

```python
# A random bytearray
randomByteArray = bytearray('ABC', 'utf-8')

# Create a memoryview of the bytearray
mv = memoryview(randomByteArray)

# Access the memory view's zeroth index
print(mv[0])  # Output: 65, the byte representation of 'A' (ASCII value)

# Create a byte object from a slice of the memory view (first two elements)
print(bytes(mv[0:2]))  # Output: b'AB' (byte representation of 'A' and 'B')

# Create a list from the memory view (first three elements)
print(list(mv[0:3]))  # Output: [65, 66, 67] (ASCII values of 'A', 'B', and 'C')
```

Output

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

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

Python object()

main.py

```python
python = object()

print(type(python))
print(dir(python))
```

Output

```
<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

main.py

```python
# Calling function
result = chr(102)  # It returns string representation of a char
result2 = chr(112)

# Displaying result
print(result)
print(result2)

# Verify, is it string type?
print("is it string type:", type(result) is str)
```

Output

```
f
p
is it string type: True

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

Python complex()

main.py

```python
# Python complex() function example
# Calling function
a = complex(1)  # Passing single parameter
b = complex(1, 2)  # Passing both parameters

# Displaying result
print(a)
print(b)
```

Output

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

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

Python delattr() Function

main.py

```python
class Student:
    id = 101
    name = "Pranshu"
    email = "pranshu@abc.com"

    # Declaring function
    def getinfo(self):
        print(self.id, self.name, self.email)

s = Student()
s.getinfo()

delattr(Student,'course')  # Removing attribute which is not available

s.getinfo()  # error: throws an error
```

Output

```
101 Pranshu pranshu@abc.com
ERROR!
Traceback (most recent call last):
  File "<main.py>", line 13, in <module>
AttributeError: type object 'Student' has no attribute 'course'

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

Python dir() Function

main.py

```python
# Calling function
att = dir()

# Displaying result
print(att)
```

Output

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

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

Python divmod() Function

main.py

```python
# Python divmod() function example
# Calling function
result = divmod(10, 2)

# Displaying result
print(result)
```

Output

```
(5, 0)

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

Python enumerate() Function

main.py

```python
# Calling function
result = enumerate([1, 2, 3])

# Displaying result
print(result)
print(list(result))
```

Output

```
<enumerate object at 0x7f09696736a0>
[(0, 1), (1, 2), (2, 3)]

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

Python dict()

main.py

```python
# Calling function
result = dict()  # returns an empty dictionary
result2 = dict(a=1, b=2)

# Displaying result
print(result)
print(result2)
```

Output

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

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

Python filter() Function

main.py

```python
# Python filter() function example
def filterdata(x):
    if x > 5:
        return x

# Calling function
result = filter(filterdata, (1, 2, 6))

# Displaying result
print(list(result))
```

Output

```
[6]

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

Python hash() Function

main.py

```python
# Python hash() Function Example
result = hash(21)  # integer value
result2 = hash(22.2)  # decimal value
print(result)
print(result2)
```

Output

```
21
461168601842737174

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

Python help() Function

main.py

```python
# Python help() Function Example
info = help()  # No argument
print(info)
```

Output

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

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

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

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

help>
```

Python min() Function

main.py

```python
# Python min() Function Example
small = min(2225, 325, 2025)  # returns smallest element
small2 = min(1000.25, 2025.35, 5625.36, 10052.50)
print(small)
print(small2)
```

Output

```
325
1000.25

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

Python set() Function

main.py

```python
# Python set() Function Example
result = set()  # empty set
result2 = set('12')
result3 = set('javatpoint')
print(result)
print(result2)
print(result3)
```

Output

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

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

Python hex() Function

main.py

```python
# Python hex() Function Example
result = hex(1)
result2 = hex(342)
print(result)
print(result2)
```

Output

```
0x1
0x156

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

Python id() Function

main.py

```python
# Python id() Function Example
val = id("Javatpoint")  # string object
val2 = id(1200)  # integer object
val3 = id([25, 336, 95, 236, 92, 3225])  # List object
print(val)
print(val2)
print(val3)
```

Output

```
135731578803312
135731580153360
135731581979008

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

Python setattr() Function

main.py

```python
# Python setattr() Function Example
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)
# Adding new attribute
setattr(student, 'email', 'sohan@abc.com')
print(student.email)
```

Output

```
102
Sohan
sohan@abc.com

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

Python slice() Function

main.py

```python
# Python slice() Function Example
result = slice(5)  # returns slice object
result2 = slice(0, 5, 3)  # returns slice object
print(result)
print(result2)
```

Output

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

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

Python sorted() Function

main.py

```python
# Python sorted() Function Example
str = "javatpoint"  # declaring string
sorted1 = sorted(str)  # sorting string
print(sorted1)
```

Output

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

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

Python next() Function

main.py

```python
# Python next() Function Example
number = iter([256, 32, 82])  # Creating iterator
item = next(number)
print(item)
item = next(number)
print(item)
item = next(number)
print(item)
```

Output

```
256
32
82

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

Python input() Function

main.py

```python
# Python input() Function Example
val = input("Enter a value: ")
print("You entered:", val)
```

Output

```
Enter a value: 45
You entered: 45

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

Python int() Function

main.py

```python
# Python int() Function Example
val = int(10)  # integer value
val2 = int(10.52)  # float value
val3 = int('10')  # string value
print("integer values:", val, val2, val3)
```

Output

```
integer values: 10 10 10

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

Python isinstance() Function

main.py

```python
# Python isinstance() Function Example
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))  # isinstance of Student class
print(isinstance(lst, Student))
```

Output

```
True
False

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

Python oct() Function

main.py

```python
# Python oct() Function Example
val = oct(10)
print("Octal value of 10:", val)
```

Output

```
Octal value of 10: 0o12

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

Python ord() Function

main.py

```python
# Python ord() Function Example
print(ord('8'))
print(ord('R'))
print(ord('&'))
```

Output

```
56
82
38

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

Python pow() Function

main.py

```python
# Python pow() Function Example
print(pow(4, 2))
print(pow(-4, 2))
print(pow(4, -2))
print(pow(-4, -2))
```

Output

```
16
16
0.0625
0.0625

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

Python print() Function

main.py

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

Output

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

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

Python range() Function

main.py

```python
# Python range() Function Example
print(list(range(0)))
print(list(range(4)))
print(list(range(1, 7)))
```

Output

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

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

Python reversed() Function

main.py

```python
# Python reversed() Function Example
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)))
```

Output

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

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

Python round() Function

main.py

```python
# Python round() Function Example
print(round(10))
print(round(10.8))
print(round(6.6))
```

Output

```
10
11
7

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

Python issubclass() Function

main.py

```python
# Python issubclass() Function Example
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)))
```

Output

```
True
False
True
True

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

Python str

main.py

```python
# Python str() Function Example
print(str('4'))
```

Output

```
4

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

Python tuple() Function

main.py

```python
# Python tuple() Function Example
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)
```

Output

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

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

Python type()

main.py

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

Output

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

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

Python vars() function

main.py

```python
# Python vars() Function Example
class Python:
    def __init__(self, x = 7, y = 9):
        self.x = x
        self.y = y

InstanceOfPython = Python()
print(vars(InstanceOfPython))
```

Output

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

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

Python zip() Function

main.py

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

Output

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

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

Python Lambda Functions
Code 1:

main.py

```python
# Code to demonstrate how we can use a lambda function for adding 4 to the input number
add = lambda num: num + 4
print(add(6))
```

Output

```
10

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

Program Code 2:

main.py

```python
# Code to demonstrate a function using def to add 4 to the input number
def add(num):
    return num + 4
print(add(6))
```

Output

```
10

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

Program Code 3:

main.py

```python
# Code to demonstrate a lambda function that multiplies two numbers
a = lambda x, y : (x * y)
print(a(4, 5))
```

Output

```
20

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

Program Code 4:

main.py

```python
# Code to demonstrate a lambda function that adds three numbers
a = lambda x, y, z : (x + y + z)
print(a(4, 5, 5))
```

Output

```
14

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

What's the Distinction Between Lambda and Def Functions?

main.py

```python
# Python code to show the reciprocal of the given number
def reciprocal(num):
    return 1 / num

lambda_reciprocal = lambda num: 1 / num

print("Def keyword: ", reciprocal(6))
print("Lambda keyword: ", lambda_reciprocal(6))
```

Output

```
Def keyword:  0.16666666666666666
Lambda keyword:  0.16666666666666666

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

Using Lambda Function with filter()

main.py

```python
# Code to filter the odd numbers from the given list using lambda function
    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 number is:', odd_list)
```

Output

```
The list of odd number is: [35, 69, 55, 75, 73]

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

Using Lambda Function with map()

main.py

```python
# Code to square all the entries in a list using the map() function
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)
```

Output

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

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

Using Lambda Function with List Comprehension

main.py

```python
# Code to calculate the square of each number using list comprehension with
    lambda
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="
        ")
```

Output

```
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 ===
```

Using Lambda Function with if-else

main.py

```python
# Code to use lambda function with if-else block
Minimum = lambda x, y: x if (x < y) else y
print('The greater number is:', Minimum(35, 74))
```

Output

```
The greater number is: 35

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

Using Lambda with Multiple Statements

```python
# Code to find the third largest number from each sublist using lambda functions
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 Statement

```python
# 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
```

```python
# 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

```python
# 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

```python
# 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

```python
#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

```python
[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

```python
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

```python
# 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

```python
# 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

```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

```python
# 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.

```python
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:

```python
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:

```python
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

```python
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

```python
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
```

```python
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
```

1. Generator Function Example

```python
def simple():
    for i in range(10):
        if i % 2 == 0:
            yield i

for i in simple():
    print(i)
```

Output:

```
0
2
4
6
8

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

2. Using Multiple yield Statements

```python
def multiple_yield():
    str1 = "First String"
    yield str1

    str2 = "Second string"
    yield str2

    str3 = "Third String"
    yield str3

obj = multiple_yield()
print(next(obj))
print(next(obj))
print(next(obj))
```

Output:

```
First String
Second string
Third String


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

3. Generator Expression vs. List Comprehension

main.py

```python
lst = [1, 2, 3, 4, 5, 6, 7]

z = [x**3 for x in lst]

a = (x**3 for x in lst)

print(a)
print(z)
```

Output:

```
<generator object <genexpr> at 0x715265bdb9f0>
[1, 8, 27, 64, 125, 216, 343]


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

4. Using next() with Generator Expression

main.py

```python
lst = [1, 2, 3, 4, 5, 6]

z = (x**3 for x in lst)

print(next(z))  # 1
print(next(z))  # 8
print(next(z))  # 27
print(next(z))  # 64
```

Output:

```
1
8
27
64


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

5. Printing Multiplication Table Using a Generator

main.py

```python
def table(n):
    for i in range(1, 11):
        yield n * i

for i in table(15):
    print(i)
```

Output:

```
15
30
45
60
75
90
105
120
135
150
```

6. Comparing Memory Usage (List vs. Generator)

main.py

```python
import sys

nums_squared_list = [i * 2 for i in range(1000)]
print("Memory in Bytes:", sys.getsizeof(nums_squared_list))

nums_squared_gc = (i ** 2 for i in range(1000))
print("Memory in Bytes:", sys.getsizeof(nums_squared_gc))
```

Output:

```
Memory in Bytes: 8856
Memory in Bytes: 200


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

7. Generating an Infinite Sequence

main.py

```python
def infinite_sequence():
    num = 0
    while True:
        yield num
        num += 1

for i in infinite_sequence():
    print(i)
```

Output:

```
89947
89948
89949
89950
89951
89952
89953
89954
89955
89956
89957
89958
89959
89960
89961
89962
89963
89964
89965
89966
89967
8^C91142
Traceback (most recent call last):
  File "/home/main.py", line 8, in <module>
    print(i)
KeyboardInterrupt


...Program finished with exit code 2
Press ENTER to exit console.
```