Python Functions

Illustration of a User-Defined Function

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

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

Calling a Function

```python
def a_function( string ):
    "This prints the value of length of string"
    return len(string)

# Calling the function we defined
print( "Length of the string Functions is: ", a_function( "Functions" ) )
print( "Length of the string Python is: ", a_function( "Python" ) )
```

input

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

Pass by Reference vs. Pass by Value

```python
def square( item_list ):
    '''''''This function will find the square of items in the list'''
    squares = [ ]
    for l in item_list:
        squares.append( l**2 )
    return squares

# calling the defined function
my_list = [17, 52, 8];
my_result = square( my_list )
print( "Squares of the list are: ", my_result )
```

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

Function Arguments

1) Default Arguments

```python
def function( n1, n2 = 20 ):
    print("number 1 is: ", n1)
    print("number 2 is: ", n2)


# Calling the function and passing only one argument
print( "Passing only one argument" )
function(30)

# Now giving two arguments to the function
print( "Passing two arguments" )
function(50,30)
```

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

2) Keyword Arguments

```python
def function( n1, n2 ):
    print("number 1 is: ", n1)
    print("number 2 is: ", n2)

# Calling function and passing arguments without using keyword
print( "Without using keyword" )
function( 50, 30)

# Calling function and passing arguments using keyword
print( "With using keyword" )
function( n2 = 50, n1 = 30)
```

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

3) Required Arguments

```python
def function( n1, n2 ):
    print("number 1 is: ", n1)
    print("number 2 is: ", 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 )

# Calling function and passing only one argument
print( "Passing only one argument" )
try:
    function( 30 )
except:
    print( "Function needs two positional arguments" )
```

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

4) Variable-Length Arguments

```python
def function( *args_list ):
    ans = []
    for l in args_list:
        ans.append( l.upper() )
    return ans
# Passing args arguments
object = function('Python', 'Functions', 'tutorial')
print( object )

# defining a function
def function( **kargs_list ):
    ans = []
    for key, value in kargs_list.items():
        ans.append([key, value])
    return ans
# Paasing kwargs arguments
object = function(First = "Python", Second = "Functions", Third = "Tutorial")
print(object)
```

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

return Statement

```python
def square( num ):
    return num**2

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

# Defining a function without return statement
def square( num ):
    num**2

# Calling function and passing arguments.
print( "Without return statement" )
print( square( 52 ) )
```

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

The Anonymous Functions

```python
lambda_ = lambda argument1, argument2: argument1 + argument2;

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

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

Scope and Lifetime of Variables

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

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

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

Python Capability inside Another Capability

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

    number()
word()
```

```
Python functions tutorial
5
```

Python Built-in Functions

Python abs() Function

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

#  floating number
floating = -20.83
print('Absolute value of -40.83 is:', abs(floating))
```

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

Python all() Function

Python all() Function Example

```python
k = [1, 3, 4, 6]
print(all(k))

# all values false
k = [0, False]
print(all(k))

# one false value
k = [1, 3, 7, 0]
print(all(k))

# one true value
k = [0, False, 5]
print(all(k))

# empty iterable
k = []
print(all(k))
```

```
True
False
False
False
True
```

Python bin() Function

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

```
0b1010
```

Python bool()

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

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

Python compile() Function

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

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

Python exec() Function

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

```
True
12
```

Python any() Function

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

l = [0, False]
print(any(l))

l = [0, False, 5]
print(any(l))

l = []
print(any(l))
```

```
True
False
True
False
```

Python float()

```python
# for integers
print(float(9))

# for floats
print(float(8.19))

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

# for string floats with whitespaces
print(float("     -17.19\n"))

# string float error
print(float("xyz"))
```

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

Python format() Function

```python
# d, f and b are a type

# integer
print(format(123, "d"))

# float arguments
print(format(123.4567898, "f"))

# binary format
print(format(12, "b"))
```

```
123
123.456790
1100
```

Python hasattr() Function

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

l = [0, False]
print(any(l))

l = [0, False, 5]
print(any(l))

l = []
print(any(l))
```

```
True
False
True
False
```

Python iter() Function Example

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

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))
```

```
1
2
3
4
5
```

Python len() Function

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

```
6
```

Python list()

```python
# empty list
print(list())

# string
String = 'abcde'
print(list(String))

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

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

Python memoryview() Function

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

mv = memoryview(randomByteArray)

# access the memory view's zeroth index
print(mv[0])

# It create byte from memory view
print(bytes(mv[0:2]))

# It create list from memory view
print(list(mv[0:3]))
```

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

Python object()

```python
python = object()

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

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

Python hash() Function

```python
# Calling function
result = hash(21) # integer value
result2 = hash(22.2) # decimal value
# Displaying result
print(result)
print(result2)
```

```
21
461168601842737174
```

Python pow() Function

```python
# positive x, positive y (x**y)
print(pow(4, 2))

# negative x, positive y
print(pow(-4, 2))

# positive x, negative y (x**-y)
print(pow(4, -2))

# negative x, negative y
print(pow(-4, -2))
```

```
16
16
0.0625
0.0625
```

Python reversed() Function

```python
# for string
String = 'Java'
print(list(reversed(String)))

# for tuple
Tuple = ('J', 'a', 'v', 'a')
print(list(reversed(Tuple)))

# for range
Range = range(8, 12)
print(list(reversed(Range)))

# for list
List = [1, 2, 7, 5]
print(list(reversed(List)))
```

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

Python issubclass() Function

```python
class Rectangle:
  def __init__(rectangleType):
    print('Rectangle is a ', rectangleType)

class Square(Rectangle):
  def __init__(self):
    Rectangle.__init__('square')

print(issubclass(Square, Rectangle))
print(issubclass(Square, list))
print(issubclass(Square, (list, Rectangle)))
print(issubclass(Rectangle, (list, Rectangle)))
```

```
True
False
True
True
```

Python tuple() Function

```python
t1 = tuple()
print('t1=', t1)

# creating a tuple from a list
t2 = tuple([1, 6, 9])
print('t2=', t2)

# creating a tuple from a string
t1 = tuple('Java')
print('t1=',t1)

# creating a tuple from a dictionary
t1 = tuple({4: 'four', 5: 'five'})
print('t1=',t1)
```

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

Python zip() Function

```python
numList = [4,5, 6]
strList = ['four', 'five', 'six']

# No iterables are passed
result = zip()

# Converting itertor to list
resultList = list(result)
print(resultList)

# Two iterables are passed
result = zip(numList, strList)

# Converting itertor to set
resultSet = set(result)
print(resultSet)
```

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

Python Lambda Functions

```python
add = lambda num: num + 4
print( add(6) )
```

```
10
```

```python
def add( num ):
    return num + 4
print( add(6) )
```

```
10
```

```python
a = lambda x, y : (x * y)
print(a(4, 5))
```

```
10
```

```python
a = lambda x, y, z : (x + y + z)
print(a(4, 5, 5))
```

```
14
```

What's the Distinction Between Lambda and Def Functions?

```python
def reciprocal( num ):
    return 1 / num

lambda_reciprocal = lambda num: 1 / num

# using the function defined by def keyword
print( "Def keyword: ", reciprocal(6) )

# using the function defined by lambda keyword
print( "Lambda keyword: ", lambda_reciprocal(6) )
```

```
14
```

Using Lambda Function with filter()

```python
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)
```

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

Using Lambda Function with map()

```python
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 )
```

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

Using Lambda Function with List Comprehension

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

```
input
The square value of all numbers from 0 to 10: 0 The square value of all numbers from 0 to 10: 1 The square value of all numbers from 0 to 10: 4 The square value of all
numbers from 0 to 10: 9 The square value of all numbers from 0 to 10: 16 The square value of all numbers from 0 to 10: 25 The square value of all numbers from 0 to 10
: 36 The square value of all numbers from 0 to 10: 49 The square value of all numbers from 0 to 10: 64 The square value of all numbers from 0 to 10: 81 The square valu
e of all numbers from 0 to 10: 100
```

Using Lambda Function with if-else

```python
Minimum = lambda x, y : x if (x < y) else y
print('The greater number is:', Minimum( 35, 74 ))
```

```
The greater number is: 35
```

Using Lambda with Multiple Statements

```python
my_List = [ [3, 5, 8, 6], [23, 54, 12, 87], [1, 2, 4, 12, 5] ]
# sorting every sublist of the above list
sort_List = lambda num : ( sorted(n) for n in num )
# Getting the third largest number of the sublist
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]
```

Python Modules

example_module.py main_program.py

python > main_program.py

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

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Code

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

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

Importing and also Renaming

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

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

Python from...import Statement

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

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

Import all Names - From import * Statement

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

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

Locating Path of Modules

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

```
Path of the sys module in the system is: ['/home', '/usr/lib/python312.zip', '/usr/lib/python3.12', '/usr/lib/python3.12/lib-dynload', '/usr/local/lib/python3.12/dist-packages', '/usr/lib/python3/dist-packages']
```

The dir() Built-in Function

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

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

Namespaces and Scoping

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

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

PYTHON ARRAYS

1. **Accessing array elements**

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

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

1. **Deleting the elements from Array**

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

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

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

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

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


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

1. **To find the length of array**

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

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

Python Decorator

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

input

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

Inner Function

main.py

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

input

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

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

```
9
21
```

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

```
Hello
```

Decorating functions with parameters

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

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

```
Hello
```

Syntactic Decorator

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


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

```

```
2.0
```

Reusing Decorator

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

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

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

Python Decorator with Argument

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

PROBLEMS 1   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

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

Returning Values from Decorated Functions

```
main.py 1, U    hello.py 1, U
123 > hello.py > ...
from decorator import do_twice
@do_twice
def return_greeting(name):
    print("We are created greeting")
    return f"Hi {name}"
hi_adam = return_greeting("Adam")
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS

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

Fancy Decorators

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

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

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

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

```
Hello Peter
Hello Peter
```

Decorator with Arguments

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

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

    return decorator_repeat

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

function1("John")
```

```
John
John
John
John
John
```

Stateful Decorators

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

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

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

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

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

input

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

Classes as Decorators

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

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

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

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

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

input

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

Python Generators

How to Create Generator function in Python?

main.py

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

#Successive Function call using for loop
for i in simple():
    print(i)
```

```
0
2
4
6
8
```

yield vs. return

```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))
```

```
First String
Second string
Third String
```

Generator Expression

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

# List Comprehension
z = [x**3 for x in list]

# Generator expression
a = (x**3 for x in list)

print(a)
print(z)
```

```
<generator object <genexpr> at 0x7475d12279f0>
[1, 8, 27, 64, 125, 216, 343]
```

```python
z = (x**3 for x in list)

print(next(z))

print(next(z))

print(next(z))

print(next(z))
```

```
1
8
27
64
```

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

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

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

```python
import sys
# List comprehension
nums_squared_list = [i * 2 for i in range(1000)]
print("Memory in Bytes:", sys.getsizeof(nums_squared_list))

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

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

Generate Infinite Sequence

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

for i in infinite_sequence():
    print(i)

```

```
4644
4645
4646
4647
4648
4649
4650
4651
4652
4653
4654
4655
4656
4657
4658
4659
4660
4661
4662
4663
4664
4665
4666
4667
4668
4669
4670
4671
4672
4673
4674
4675
4676
4677
4678
4679
```