Python Functions:

Illustration of a User-Defined Function

main.py Output

```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_ )
```

main.py Output

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

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

Calling a Function

main.py Output

```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" ) )
```

main.py Output

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

Pass by Reference vs. Pass by Value

main.py Output

```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 )
```

main.py Output

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

Function Arguments

1. Default arguments

main.py Output

```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)
```

main.py Output

```
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

main.py Output

```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)
```

main.py Output

```
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

main.py Output

```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" )
```

main.py Output

```
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

main.py Output

```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)
```

main.py Output

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

return Statement

main.py Output

```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 ) )
```

main.py Output

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

The Anonymous Functions

main.py | Output

```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 ) )
```

main.py | Output

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

Scope and Lifetime of Variables

main.py | Output

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

main.py | Output

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

Python Capability inside Another Capability

main.py | Output

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

main.py Output

```
Python functions tutorial
5
```

Python abs() Function

main.py Output

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

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

main.py Output

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

Python all() Function

main.py Output

```python
# all values true
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))
```

main.py Output

```
True
False
False
False
True
```

Python bin() Function

main.py Output

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

main.py Output

```
0b1010
```

Python bool()

main.py Output

```
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))
```

main.py Output

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

Python bytes()

main.py Output

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

main.py Output

```
b'Hello World.'
```

Python callable() Function

main.py

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

Output

```
False

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

Python compile() Function

main.py

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

Output

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

Python exec() Function

main.py

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

Output

```
True
12
```

Python sum() Function

main.py

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

Output

```
7
17

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

Python any() Function

main.py

```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))
```

Output

```
True
False
True
False

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

Python ascii() Function

main.py

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

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 'uti
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'))
```

Output

```
9

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

Python float()

main.py

```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"))
```

```
Output
9.0
8.19
-24.27
-17.19
ERROR!
```

Python format() Function

```
main.py                                   Run        Output
# d, f and b are a type                              123
# integer                                            123.456790
print(format(123, "d"))                              1100
# float arguments
print(format(123.4567898, "f"))                      === Code Execution Successful ===
# binary format
print(format(12, "b"))
```

Python frozenset()

```
main.py                                   Share  Run  Output
# tuple of letters                                   Frozen set is: frozenset({'m', 't', 'o', 's', 'r'})
letters = ('m', 'r', 'o', 't', 's')                  Empty frozen set is: frozenset()
fSet = frozenset(letters)
print('Frozen set is:', fSet)                        === Code Execution Successful ===
print('Empty frozen set is:', frozenset())
```

Python getattr() Function

```
main.py                                   Share  Run  Output
class Details:                                       The age is: 22
    age = 22                                         The age is: 22
    name = "Phill"
details = Details()                                  === Code Execution Successful ===
print('The age is:', getattr(details, "age"))
print('The age is:', details.age)
```

Python globals() Function

```
main.py                                   Run        Output
age = 22                                             The age is: 22
globals()['age'] = 22
print('The age is:', age)                            === Code Execution Successful ===
```

Python hasattr() Function

main.py

```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))
```

Output

```
True
False
True
False

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

Python iter() Function

main.py

```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))
```

Output

```
1
2
3
4
5

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

Python len() Function

main.py

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

Output

```
6

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

Python list()

main.py

```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))
```

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

Output

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

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

Python map() Function

main.py

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

Output

```
<map object at 0x7f9c03f3d450>
{8, 2, 4, 6}

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

Python memoryview() Function

main.py

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

# Create a memory view
mv = memoryview(randomByteArray)

# Access the memory view's zeroth index
print(mv[0])  # Output: 65 (ASCII code for 'A')

# Create bytes from memory view
print(bytes(mv[0:2]))  # Output: b'AB'

# Create list from memory view
print(list(mv[0:3]))  # Output: [65, 66, 67]
```

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__', '_
    '__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 dir() Function

main.py

```python
# Calling function
att = dir()
# Displaying result
print(att)
```

Output

```
['__annotations__', '__builtins__', '__doc__', '__loader__', '__name__', '__p
    , '__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 0x7adcf65276a0>
[(0, 1), (1, 2), (2, 3)]

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

Python hash() Function

main.py

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

Output

```
21
461168601842737174

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

Python help() Function

main.py

```python
# Calling function
info = help() # No argument
# Displaying result
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>
=== Session Ended. Please Run the code again ===
```

Python min() Function

main.py

```python
# Calling function
small = min(2225,325,2025) # returns smallest element
small2 = min(1000.25,2025.35,5625.36,10052.50)
# Displaying result
print(small)
print(small2)
```

Output

```
325
1000.25

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

Python set() Function

main.py

```python
# Calling function
result = set() # empty set
result2 = set('12')
result3 = set('javatpoint')
# Displaying result
print(result)
print(result2)
print(result3)
```

Output

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

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

Python hex() Function

main.py

```python
# Calling function
result = hex(1)
# integer value
result2 = hex(342)
# Displaying result
print(result)
print(result2)
```

Output

```
0x1
0x156

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

Python id() Function

main.py

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

Output

```
138688839734192
138688841100816
138688842942912

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

Python setattr() Function

main.py

```python
class Student:
    id = 0
    name = ""

    def __init__(self, id, name):
        self.id = id
        self.name = name

student = Student(102,"meghana")
print(student.id)
print(student.name)
#print(student.email) product error
setattr(student, 'email' ,'meghana@gmail.com') # adding new attribute
print(student.email)
```

Output

```
102
meghana
meghana@gmail.com

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

Python slice() Function

main.py

```python
# Calling function
result = slice(5) # returns slice object
result2 = slice(0,5,3) # returns slice object
# Displaying result
print(result)
print(result2)
```

Output

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

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

Python sorted() Function

main.py

```python
str = "javatpoint" # declaring string
# Calling function
sorted1 = sorted(str) # sorting string
# Displaying result
print(sorted1)
```

Output

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

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

Python next() Function

main.py

```python
number = iter([256, 32, 82])
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
# Calling function
val = input("Enter a value: ")
# Displaying result
print("You entered:",val)
```

Output

```
Enter a value: 44
You entered: 44

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

Python int() Function

main.py

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

Output

```
integer values : 10 10 10

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

Python isinstance() Function

main.py

```python
class Student:
    id = 101
    name = "John"
    def __init__(self, id, name):
        self.id=id
        self.name=name

student = Student(1010,"John")
lst = [12,34,5,6,767]
# Calling function
print(isinstance(student, Student)) # isinstance of Student class
print(isinstance(lst, Student))
```

Output

```
True
False

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

Python oct() Function

main.py

```python
# Calling function
val = oct(10)
# Displaying result
print("Octal value of 10:",val)
```

Output

```
Octal value of 10: 0o12

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

Python ord() Function

main.py

```python
# Code point of an integer
print(ord('8'))

# Code point of an alphabet
print(ord('R'))

# Code point of a character
print(ord('&'))
```

Output

```
56
82
38

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

Python pow() Function

main.py

```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))
```

Output

```
16
16
0.0625
0.0625

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

Python print() Function

main.py

```python
print("Python is programming language."
    )

x = 7
# Two objects passed
print("x =", x)

y = x
# Three objects passed
print('x =', x, '= y')
```

Output

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

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

Python range() Function

main.py

```python
# empty range
print(list(range(0)))

# using the range(stop)
print(list(range(4)))

# using the range(start, stop)
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
# 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)))
```

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
#  for integers
print(round(10))

#  for floating point
print(round(10.8))

#  even choice
print(round(6.6))
```

Output

```
10
11
7

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

Python issubclass() Function

main.py

```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)))
```

Output

```
True
False
True
True

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

Python tuple() Function

main.py

```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)
```

Output

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

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

Python type()

main.py

```python
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
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
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)
```

Output

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

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

Python Lambda Functions

Example 1:

main.py

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

Output

```
10
```

Example 2:

main.py

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

Output

```
10

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

Example 3:

main.py

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

Output

```
20

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

Example 4:

main.py

```python
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
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
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
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 if-else

main.py

```python
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

main.py

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

Output

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

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

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/dis
t-packages', '/usr/lib/python3/dist-packages']
```

The dir() Built-in Function

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

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

2. **Deleting the elements from Array**

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

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

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

```
main.py   File (Ctrl+M)
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.
```

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

```
main.py
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
```

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

Inner Function

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

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

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

```
9
21
```

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

```
Hello
```

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

```
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
```

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

```
Hello Peter
Hello Peter
```

Decorator with Arguments

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

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