Lab 6

September 13, 2023

[1]:
```
# Example Python Code for User-Defined function
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
```

[2]:
```
def a_function( string ):
    "This prints the value of length of string"
    return len(string)

print( "Length of the string Functions is: ", a_function( "Functions" ) )
print( "Length of the string Python is: ", a_function( "Python" ) )
```

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

[3]:
```
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
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]
```

[4]:
```
# Python code to demonstrate the use of default arguments
# defining a function
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
```

[5]:
```
# Python code to demonstrate the use of keyword arguments
  # Defining a function
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
```

[6]:
```
# Python code to demonstrate the use of default arguments
# Defining a function
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
```

[7]:
```
# Python code to demonstrate the use of variable-length arguments
# Defining a function
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']]
```

[8]:
```
# Python code to demonstrate the use of return statements
# Defining a function with return statement
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
```

[9]:
```
# Python code to demonstrate ananymous functions
# Defining a function
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
```

[10]:
```
# tuple of letters
letters = ('m', 'r', 'o', 't', 's')

fSet = frozenset(letters)
print('Frozen set is:', fSet)
print('Empty frozen set is:', frozenset())
```

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

[11]:
```
# Code to demonstrate how we can use a lambda function for adding 4 numbers
add = lambda num: num + 4
print( add(6) )
```

```
10
```

[12]:
```
# Python code to show the reciprocal of the given number to highlight the␣
 ↪difference between def() and lambda().
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) )
```

```
Def keyword:  0.16666666666666666
Lambda keyword:  0.16666666666666666
```

[13]:
```
# This code used to filter the odd numbers from the given list
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]
```

[14]:
```
#Code to calculate the square of each number of 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 )
```

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

[15]:
```
#Code to calculate square of each number of lists using list comprehension
squares = [lambda num = num: num ** 2 for num in range(0, 11)]
for square in squares:
      print('The square value of all numbers from 0 to 10:',square(), end = "
")
```

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

[16]:
```
# Code to use lambda function with if-else
Minimum = lambda x, y : x if (x < y) else y
print('The greater number is:', Minimum( 35, 74 ))
```

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

[17]:
```
# Code to print the third largest number of the given list using the lambda
function

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]
```

[ ]:

[ ]: