Course Name: Fundamentals of Programming & Data Science	**Course Code:** CMPE-112L
Assignment Type: Lab	**Dated:** 29th January 2024
Semester: 2nd	**Session:** 2023
Lab/Project/Assignment #: 3	**CLOs to be covered: CLO 2**
Lab Title: Functions in Python	**Teacher Name:** Engr. Afeef Obaid

Lab Evaluation:

CLO 2	Practice collaboratively on large problems and provide their working solutions.					
Levels (Marks)	**Level 1**	**Level 2**	**Level 3**	**Level 4**	**Level 5**	**Level 6**
(10)						
					Total	**/10**

Rubrics for Current Lab Evaluation

Scale	**Marks**	**Level**	**Rubric**
Excellent	**5**	L1	Submitted all lab tasks, BONUS task, have good understanding.
Very Good	**4**	L2	Submitted the lab tasks but have good understanding
Good	**3**	L3	Submitted the lab tasks but have weak understanding.
Basic	**2**	L4	Submitted the lab tasks but have no understanding.
Barely Acceptable	**1**	L5	Submitted only one lab task.
Not Acceptable	**0**	L6	Did not attempt

LAB No. 3

Lab Goals/Objectives:

By reading this manual, students will be able:

- To understand the concept of user defined functions in Python
- To learn how to define and call functions in Python
- To understand anonymous function in Python
- To know about doc strings in python

Equipment Required: Computer system with Pycharm IDE and python package installed on it

Functions In Python

Functions in Python are reusable blocks of code that perform a specific task. They allow you to write code once and use it multiple times without having to rewrite the same code again and again. Functions can take input arguments, perform operations, and return output.

There are three types of functions in Python:

- Built In Functions
- User-Defined Functions (UDFs)
- Anonymous functions

Built In Functions

These functions are defined and pre-coded in python. Some examples of built-in functions are as follows:
min(), max(), len(), sum(), type(), range(), dict(), list(), tuple(), set(), print(), etc.

Example 1

```
name=input("Enter Your name ")
size=len(name)
print(f"Your name has {size} letters")
```

Example 2

```
num1=int(input("Enter 1st Number "))
num2=int(input("Enter 2nd Number "))
large=max(num1,num2)
print(f"{large} is greater between {num1} and {num2}")
```

In above example input(), len(), max() and print() are the built in functions of Python.

User-Defined Functions (UDFs)

We can create functions to perform specific tasks as per our needs. Such functions are called user-defined functions.

The four steps to defining a function in Python are the following:

- Use the keyword def to declare the function and follow this up with the function name.
- Add parameters to the function: they should be within the parentheses of the function. End your line with a colon.
- Add statements that the functions should execute.
- End your function with a return statement if the function should output something. Without the return statement, your function will return an object None

Syntax

```
def function_name(parameters):
    pass
     # Code and Statements
            return
```

Example

```
def hello():
   pass
   name=input("Enter name ")
   if name:
      print("Hello",name)
   else:
      print("Hello World")
            return
hello()
```

The return Statement

In Python, the return statement is used to return a value from a function. When the return statement is executed, the function terminates immediately, and the value specified in the return statement is returned to the caller.

Example

```
def factorial(a):
fact=1
for i in range(fact,a+1):
   fact=fact*i
return fact
num=int(input("Enter number "))
f=factorial(num)
print(f"Factorial of {num} is {f}")
```

In this example, the factorial function takes one parameter a, calculates its factorial and stores it in the fact variable. Then it uses the return statement to return the value of fact to the caller.
After that we call the factorial function with one argument num, and store the result in the f variable. Finally, we print the value of f, which is 8.

Call a Function

In Python, you can call a function by using its name followed by parentheses () and any arguments that the function takes, if any.

In general, when you call a function, you need to make sure that you pass the correct number of arguments, and that the arguments have the correct data type and format that the function expects.

In above example, we define a function **factorial** that takes one argument **a** and calculate its factorial and return the value of **fact** to the caller. We then call the **factorial** function and pass the argument "num" to it.

Function Arguments in Python

There are four types of arguments that we can provide in a function:

- Default Arguments
- Keyword Arguments
- Required Arguments
- Variable length Arguments

Default Arguments

Default arguments are those that take a default value if no argument value is passed during the function call. You can assign this default value by with the assignment operator =, just like in the following example:

Example

```
def area(l=5,b=6):
   Area=l*b
   return Area
def perimeter(l=5,b=6):
   Perimeter=2*(l+b)
   return Perimeter
length=int(input("Enter Length of Rectangle "))
Breadth=int(input("Enter Breadth of Rectangle "))
rect_area=area()
rect_perimeter=perimeter()
print(f"Length:{length}   Breadth:{Breadth}   Area:{rect_area}")
print(f"Length:{length}   Breadth:{Breadth}   Area:{rect_perimeter}")
rect_area=area(length,Breadth)
rect_perimeter=perimeter(length,Breadth)
print(f"Length:{length}   Breadth:{Breadth}   Area:{rect_area}")
print(f"Length:{length}   Breadth:{Breadth}   Area:{rect_perimeter}")
```

Keyword Arguments

We can provide arguments with key = value, this way the interpreter recognizes the arguments by the parameter name. Hence, the order in which the arguments are passed does not matter.

Example

```
def area(l,b):
    print(f"Length:{l}")
    print(f"Breadth:{b}")
    Area=l*b
    print(f"Area:{Area}")
    return
def perimeter(l,b):
    print(f"Length:{l}")
    print(f"Breadth:{b}")
    Perimeter=2*(l+b)
    print(f"Perimeter:{Perimeter}")
    return
length=int(input("Enter Length of Rectangle "))
Breadth=int(input("Enter Breadth of Rectangle "))
area(b=Breadth,l=length)
perimeter(l=length,b=Breadth)
```

Required Arguments

As the name kind of gives away, the required arguments of a UDF are those that have to be in there. These arguments need to be passed during the function call and in precisely the right order, just like in the following example:

Example

```
def area(l,b):
    print(f"Length:{l}")
    print(f"Breadth:{b}")
    Area=l*b
    print(f"Area:{Area}")
    print("==========")
    return
def perimeter(l,b):
    print(f"Length:{l}")
    print(f"Breadth:{b}")
    Perimeter=2*(l+b)
    print(f"Perimeter:{Perimeter}")
    return
length=int(input("Enter Length of Rectangle "))
Breadth=int(input("Enter Breadth of Rectangle "))
area(length,Breadth)
perimeter(length,Breadth)
```

Variable length Arguments

Sometimes we may need to pass more arguments than those defined in the actual function. This can be done using variable-length arguments.

While creating a function, pass a * before the parameter name while defining the function. The function accesses the arguments by processing them in the form of tuple.

Example

```
def area(*dimentions):
    print(f"Length:{dimentions[0]}")
    print(f"Breadth:{dimentions[1]}")
```

```
 Area=dimentions[0]*dimentions[1]
    print(f"Area:{Area}")
    return
length=int(input("Enter Length of Rectangle "))
Breadth=int(input("Enter Breadth of Rectangle "))
dimentions=(length,Breadth)
area(*dimentions)
```

Anonymous functions

Anonymous functions, which are also called lambda functions because they are not declared with the standard def keyword.

Syntax

lambda arguments: expression

Lambda functions are often used in situations where a small function is required for a short period of time. They are commonly used as arguments to higher-order functions, such as map, filter, and reduce.

Example

```
def double(x):
  return x * 2
lambda x: x * 2
```

The above lambda function has the same functionality as the double function defined earlier. However, the lambda function is anonymous, as it does not have a name.

Lambda functions can have multiple arguments, just like regular functions. Here is an example of a lambda function with multiple arguments:

Example

```
def multiply(x, y):
    return x * y
lambda x, y: x * y
```

Lambda functions can also include multiple statements, but they are limited to a single expression.

Example # 1

```
lambda x, y: print(f'{x} * {y} = {x * y}')
```

Example # 2

```
b=lambda x:"Even" if (x%2==0) else "Odd"
print(f"{b(3)}")
```

we can use lambda functions in another user defined functions as well. Below is the example

Example # 3

```
avg=lambda x,y,z:(x+y+z)/3
def square(a=avg(3,4,5)):
    return a*a
print(square())
```

Docstrings in Python

Python docstrings are the string literals that appear right after the definition of a function, method, class, or module.

Example # 1

```
def square(n):
    '''Takes in a number n, returns the square of n'''
    print(n**2)
square(5)
```

Example # 2

```
def add(num1, num2):
    """
    Add up two integer numbers.
    This function simply wraps the ``+`` operator, and does not
    do anything interesting, except for illustrating what
    the docstring of a very simple function looks like.
    Parameters
    ----------
    num1 : int
        First number to add.
    num2 : int
        Second number to add.
    Returns
    -------
    int
        The sum of ``num1`` and ``num2``.
"""
    return num1 + num2
```

Python Comments vs Docstrings

- **Python Comments**

Comments are descriptions that help programmers better understand the intent and functionality of the program. They are completely ignored by the Python interpreter.

- **Python docstrings**

As mentioned above, Python docstrings are strings used right after the definition of a function, method, class, or module. They are used to document our code.

We can access these docstrings using the **doc** attribute.

Example

```
print(add.__doc__)
```

Lab Tasks

- Write a program that calculates the Permutation and Combination of `**n**` and `**r**` by using user defined function

- Write a program that takes two numbers from the user and calculate the large number between them using lambda function and then write a UDF that takes that large number from lambda function as an argument and write its table to a specified range.

- Write a program that takes the string from the user and convert it to uppercase using lambda function and write a UDF name as invert that get uppercased string from the lambda function as an argument and print it in reverse order.

- Write a program that converts Fahrenheit to Celsius and Celsius to Fahrenheit using user defined functions.

- Write a program to calculate the Grade point Average GPA for a semester using user defined function. GPA for a semester is calculated by:

    a. Multiply Grade Point with the credit hours in each course to obtain total Grade Points.

    b. Add up the total grade points to cumulative Grade Points and divide by the total number of credit hours in order to calculate the GPA for a semester.

  Ask the user about the number of subjects he studied in the semester and then ask for the Grade Point in these subjects and their credit hours in order to calculate the GPA.