EXPERIMENT - 2

GROUP MEMBERS NAME:
Harshal Jain - 60004190041
Junaid Girkar - 60004190057
Khushi Chavan - 60004190061
Megh Dedhia - 60004190067

AIM: Creating functions, classes and objects using python.

FUNCTIONS:
Python Functions is a block of related statements designed to perform a computational, logical, or evaluative task. The idea is to put some commonly or repeatedly done tasks together and make a function so that instead of writing the same code again and again for different inputs, we can do the function calls to reuse code contained in it over and over again.
Functions can be both built-in or user-defined. It helps the program to be concise, non-repetitive, and organized.
SYNTAX:

```
def function_name(parameters):
    """docstring"""
    statement(s)
    return expression
```

Creating a Function
We can create a Python function using the def keyword.
Example: Python Creating Function
CODE:

```
def hello(name):#function def
    '''This function says hello to the person passed in as a
parameter'''
    print("Hello, " + name)
```

Calling a Function
After creating a function we can call it by using the name of the function followed by parenthesis containing parameters of that particular function.
Example: Python Calling Function
CODE:

```
hello("Jack")#function call
```

OUTPUT:

```
Hello, Jack
```

Arguments of a Function
Arguments are the values passed inside the parentheses of the function. A function can have any number of arguments separated by a comma.

Example: Python Function with arguments
In this example, we will create a simple function to check whether the number passed as an argument to the function is even or odd.
CODE:

```
def num(a):#function def
    #This function passes a number as a parameter and checks
whether number is odd or even
    if(a%2==0):
        print(a,"is an even number")
    else:
        print(a,"is an odd number")

num(5)#function call
```

OUTPUT:

```
5 is an odd number
```

Types of Arguments
Python supports various types of arguments that can be passed at the time of the function call. Let's discuss each type in detail.

Default arguments
A default argument is a parameter that assumes a default value if a value is not provided in the function call for that argument. The following example illustrates Default arguments. Like C++ default arguments, any number of arguments in a function can have a default value. But once we have a default argument, all the arguments to its right must also have default values.

Keyword arguments
The idea is to allow the caller to specify the argument name with values so that the caller does not need to remember the order of parameters.
CODE:

```
def full_name(fname, lname):#this function is for the full name of
a person
  print("My name is : ",fname+" "+ lname)
full_name(fname="Jack", lname="Fernandes")
```

OUTPUT:

```
My name is :  Jack Fernandes
```

Variable-length arguments

In Python, we can pass a variable number of arguments to a function using special symbols. There are two special symbols:

- *args (Non-Keyword Arguments)
- **kwargs (Keyword Arguments)

Example 1: Variable length non-keywords argument

CODE:

```
def kid(*kids):
  print("The youngest child is " + kids[1])

kid("Jack", "Joy", "Jose")
```

OUTPUT:

```
The youngest child is Joy
```

Example 2: Variable length keyword arguments

CODE:

```
def kid(**kids):
  print("His name is " + kids["fname"])

kid(fname = "Jack", lname = "Fernandes")
```

OUTPUT:

```
His name is Jack
```

Docstring

The first string after the function is called the Document string or Docstring in short. This is used to describe the functionality of the function. The use of docstring in functions is optional but it is considered a good practice.

The below syntax can be used to print out the docstring of a function:

SYNTAX:

```
print(function_name.__doc__)
```

Example: Adding Docstring to the function

CODE:

```
def hello(name):#function def
    '''This function says hello to the person passed in as a
parameter'''
    print("Hello, " + name)
print(hello.__doc__)
```

OUTPUT:

```
This function says hello to the person passed in as a parameter
```

The return statement

The function return statement is used to exit from a function and go back to the function caller and return the specified value or data item to the caller.

SYNTAX:

```
return [expression_list]
```

The return statement can consist of a variable, an expression, or a constant which is returned to the end of the function execution. If none of the above is present with the return statement a None object is returned.

Example: Python Function Return Statement

CODE:

```
def square(x):#finds the square of the number
  return x*x
print("Square is ",square(5))
```

OUTPUT:

```
Square is  25
```

Is Python Function Pass by Reference or pass by value?

One important thing to note is, in Python every variable name is a reference. When we pass a variable to a function, a new reference to the object is created. Parameter passing in Python is the same as reference passing in Java.

CODE:

```
Example:
my_string = "Python"
def test(my_string):
    my_string = "PythonGuides"
    print("Inside the function:",my_string)
test(my_string)
```

```
print("Outside the function:",my_string)
```

OUTPUT:

```
Inside the function: PythonGuides
Outside the function: Python
```

Python Function within Functions

A function that is defined inside another function is known as the inner function or nested function. Nested functions are able to access variables of the enclosing scope. Inner functions are used so that they can be protected from everything happening outside the function.

CODE:

```
def f1():
    s = 'I love Python

    def f2():
        print(s)

    f2()

# Driver's code
f1()
```

OUTPUT:

```
I love Python
```

ANONYMOUS FUNCTION:

Python Lambda Functions are anonymous functions meaning that the function is without a name. As we already know that the def keyword is used to define a normal function in Python. Similarly, the lambda keyword is used to define an anonymous function in Python.

SYNTAX:

```
lambda arguments: expression
```

lambda arguments: expression

- This function can have any number of arguments but only one expression, which is evaluated and returned.
- One is free to use lambda functions wherever function objects are required.
- You need to keep in your knowledge that lambda functions are syntactically restricted to a single expression.
- It has various uses in particular fields of programming besides other types of expressions in functions.

Example: Lambda Function Example

CODE:

```
double = lambda x: x * 2

print("Double is ",double(4))
```

OUTPUT:

```
Double is  8
```

Difference Between Lambda functions and def defined function

Let's look at this example and try to understand the difference between a normal def defined function and lambda function. This is a program that returns the cube of a given value:

Example 1: Python Lambda Function with List Comprehension

CODE:

```
choc=["5star", "silk", "milkybar"]
newchoc=[a for a in choc if "s" in a]
print("Chocolates containing letter s is",newchoc)
```

OUTPUT:

```
Chocolates containing letter s is ['5star', 'silk']
```

Example 2: Python Lambda Function with if-else

CODE:

```
eve_odd = lambda x : "even" if (x%2==0) else "odd"
print("The number is ",eve_odd(3))
```

OUTPUT:

```
The number is  odd
```

Example 3: Python Lambda with Multiple statements

CODE:

```
list1=[[2,3,4],[1, 4, 16, 64],[3, 6, 9, 12]]
sort=lambda x:(sorted(y) for y in x)
get_small = lambda x, f : [y[0] for y in f(x)]
smallest=get_small(list1, sort)
print("Smallest numbers in each list are",smallest)
```

OUTPUT:

```
Smallest numbers in each list are [2, 1, 3]
```

Using lambda() Function with filter()

The filter() function in Python takes in a function and a list as arguments. This offers an elegant way to filter out all the elements of a sequence “sequence”, for which the function returns True. Here is a small program that returns the even numbers from an input list:

CODE:

```
list1 = [1, 5, 4, 6, 8, 11, 3, 12]
even = list(filter(lambda x: (x%2 == 0) , list1))#only filters
even numbers
print("Even numbers in the list are ",even)
```

OUTPUT:

```
Even numbers in the list are  [4, 6, 8, 12]
```

Using lambda() Function with map()

The map() function in Python takes in a function and a list as an argument. The function is called with a lambda function and a list and a new list is returned which contains all the lambda modified items returned by that function for each item.

CODE:

```
list1 = [1, 5, 4, 6, 8, 11, 3, 12]
double = list(map(lambda x: (x*2) , list1))#only filters even
numbers
print("doubled the numbers in the list are ",double)
```

OUTPUT:

```
doubled the numbers in the list are  [2, 10, 8, 12, 16, 22, 6, 24]
```

Using lambda() Function with reduce()

The reduce() function in Python takes in a function and a list as an argument. The function is called with a lambda function and an iterable and a new reduced result is returned. This performs a repetitive operation over the pairs of the iterable. The reduce() function belongs to the functools module.

CODE:

```
from functools import reduce
list1 = [1, 2, 3, 4, 5]
sum = reduce((lambda x, y: x+y), list1)
print ("Sum of number in the list is",sum)
```

OUTPUT:

```
Sum of number in the list is 15
```

RECURSION:

The term Recursion can be defined as the process of defining something in terms of itself. In simple words, it is a process in which a function calls itself directly or indirectly.

Advantages of using recursion

- A complicated function can be split down into smaller subproblems utilizing recursion.
- Sequence creation is simpler through recursion than utilizing any nested iteration.
- Recursive functions render the code look simple and effective.

Disadvantages of using recursion

- A lot of memory and time is taken through recursive calls which makes it expensive for use.
- Recursive functions are challenging to debug.
- The reasoning behind recursion can sometimes be tough to think through.

SYNTAX:

```
def func(): <--
              |
              | (recursive call)
              |
    func() ----
```

CODE:

```
def factorial(k):
  if(k > 0):
    res=k*factorial(k-1)
    print(res)
  else:
    res=1
  return res

print("Factorial till 5:")
factorial(5)
```

OUTPUT:

```
Factorial till 5:
1
2
```

```
6
24
120
```

OBJECTS IN PYTHON:

Python is an object-oriented programming language. Almost everything in Python is an object, with its properties and methods.

A Class is like an object constructor or a "blueprint" for creating objects.

CODE:

```
class Student :
    pass

s1 = Student()
s2 = Student()
s3 = Student()
```

Class attributes

Class attributes belong to the class itself; they will be shared by all the instances. Such attributes are defined in the class body parts usually at the top, for legibility.

CODE:

```
class Student :
    ## Class attributes
    totalStudents = 20
    classTeacherName = 'Komal'


 ## Class Attributes belong to class
print(Student.__dict__)
```

OUTPUT:

```
{'__module__': '__main__', 'totalStudents': 20,
'classTeacherName': 'Komal', '__dict__': <attribute '__dict__' of
'Student' objects>, '__weakref__': <attribute '__weakref__' of
'Student' objects>, '__doc__': None}
```

CODE:

```
## All are referring to same class Attribute
print(Student.totalStudents)
print(s1.totalStudents)
print(s2.totalStudents)
```

OUTPUT:

```
20
20
20
```

CODE:

```
## class Attributes does not belong to particular Instance
print(s1.__dict__)
print(s2.__dict__)
```

OUTPUT:

```
{}
{}
```

CODE:

```
## Modifying class Attribute
Student.totalStudents = 30
```

Class method

A class method is a method that is bound to a class rather than its object. It doesn't require creation of a class instance.

CODE:

```
class Student :
    ## Class attributes
    __totalStudents = 20
    classTeacherName = 'Komal'
    ## Class Method
    @classmethod
    def getTotalStudents(cls) :
        return Student.__totalStudents

print(Student.getTotalStudents())
```

OUTPUT:

```
20
```

Instance Attributes

Unlike class attributes, instance attributes are not shared by objects. Every object has its own copy of the instance attribute (In case of class attributes all objects refer to a single copy).
CODE:

```
s1.name = "Mohit"
s1.age = 20
print(s1.__dict__)
```

OUTPUT:

```
{'name': 'Mohit', 'age': 20}
```

SYNTAX:

```
<object-name> = <class-name>(<arguments>)
```

The self Parameter

The self parameter is a reference to the current instance of the class, and is used to access variables that belong to the class.
It does not have to be named self , you can call it whatever you like, but it has to be the first parameter of any function in the class:

```
class Person:
  def __init__(xyz, name, age):
    xyz.name = name
      xyz.age = age

  def myfunc(abc):
    print("Hello my name is " + abc.name)

p1 = Person("Junaid", 20)
p1.myfunc()
```

OUTPUT:

```
Hello my name is Junaid
```

Modify Object Properties

You can modify properties on objects like this:

```
p1.age = 40
```

Delete Object Properties

You can delete properties on objects or full objects itself by using the del keyword:

```
del p1.age

del p1
```

Accessing Invalid Instance Attributes will give an error.

CODE:

```
s2 = Student()
s2.rollNumber = 101

print(s2.__dict__)
print(s2.name)
```

OUTPUT:

```
AttributeError                            Traceback (most recent
call last)
<ipython-input-12-e81b27c01c56> in <module>()
----> 1 print(s2.name)
AttributeError: 'Student' object has no attribute 'name'
```

Methods:

CODE:

```
print(hasattr(s1, 'name'))
print(hasattr(s2, 'name'))
print(getattr(s1, 'name'))
print(getattr(s2, 'name', 'test'))
```

OUTPUT:

```
True
False
Mohit
test
```

CODE:

```
delattr(s1, 'name')
print(s1.__dict__)
```

OUTPUT:

```
{'age': 20}
```

Instance Method

Instance attributes are those attributes that are not shared by objects. Every object has its own copy of the instance attribute.

CODE:

```
class Student :

    ## Instance methods
    def printHello(self) :
        print("Hello")

    def print(self, str) :
        print(str)

    def printName(self) :
        print(self.name)
```

Conclusion:

We learnt about different types of functions, classes and object that are available in python and implemented each of them in a python script.