Lab No :4

Python Objects and Classes

Objectives:

In this lab, student will be able to

1. Understand the python classes
2. Learn how to use class objects and methods

Python is an object oriented programming language. Unlike procedure oriented programming, where the main emphasis is on functions, object oriented programming stresses on objects.

An object is simply a collection of data (variables) and methods (functions) that act on those data. Similarly, a class is a blueprint for that object.

We can think of class as a sketch (prototype) of a house. It contains all the details about the floors, doors, windows etc. Based on these descriptions we build the house. House is the object.

As many houses can be made from a house's blueprint, we can create many objects from a class. An object is also called an instance of a class and the process of creating this object is called **instantiation**.

Defining a Class in Python

Like function definitions begin with the def keyword in Python, class definitions begin with a class keyword.

The first string inside the class is called docstring and has a brief description about the class. Although not mandatory, this is highly recommended.

Here is a simple class definition.

```
class MyNewClass:
   '''This is a docstring. I have created a new class'''
pass
```

A class creates a new local namespace where all its attributes are defined. Attributes may be data or functions.

There are also special attributes in it that begins with double underscores `__`. For example, `__doc__` gives us the docstring of that class.

As soon as we define a class, a new class object is created with the same name. This class object allows us to access the different attributes as well as to instantiate new objects of that class.

```
class Person:
    "This is a person class"
age = 10

    def greet(self):
print('Hello')


# Output: 10
print(Person.age)

# Output: <function Person.greet>
print(Person.greet)
```

```
# Output: 'This is my second class'
print(Person.__doc__)
```

Output

```
10
<function Person.greet at 0x7fc78c6e8160>
This is a person class
```

Creating an Object in Python

We saw that the class object could be used to access different attributes.

It can also be used to create new object instances (instantiation) of that class. The procedure to create an object is similar to a function call.

```
>>> harry = Person()
```

This will create a new object instance named harry. We can access the attributes of objects using the object name prefix.

Attributes may be data or method. Methods of an object are corresponding functions of that class.

This means to say, since `Person.greet` is a function object (attribute of class), `Person.greet` will be a method object.

```
class Person:
    "This is a person class"
age = 10

    def greet(self):
print('Hello')


# create a new object of Person class
harry = Person()

# Output: <function Person.greet>
print(Person.greet)

# Output: <bound method Person.greet of <__main__.Person object>>
print(harry.greet)

# Calling object's greet() method
# Output: Hello
harry.greet()
```

Output

```
<function Person.greet at 0x7fd288e4e160>
<bound method Person.greet of <__main__.Person object at 0x7fd288e9fa30>>
Hello
```

You may have noticed the `self` parameter in function definition inside the class but we called the method simply as harry.greet() without any arguments. It still worked.

This is because, wh harry.greet() translates into Person.greet(harry) lf is passed as the first argument. So, .

In general, calling a method with a list of n arguments is equivalent to calling the corresponding function with an argument list that is created by inserting the method's object before the first argument.

For these reasons, the first arg self nt of the function in class must be the object itself. This is conventionally called . It can be named otherwise but highly recommend to follow the convention.

Another Example:

```
# We use the "class" statement to create a class
class Human:

    # A class attribute. It is shared by all instances of this class
species = "H. sapiens"

    # Basic initializer, this is called when this class is instantiated.
    # Note that the double leading and trailing underscores denote objects
    # or attributes that are used by Python but that live in user-controlled
    # namespaces. Methods(or objects or attributes) like: __init__, __str__,
    # __repr__ etc. are called special methods (or sometimes called dunder methods)
# You should not invent such names on your own.     def __init__(self, name):
        # Assign the argument to the instance's name attribute
self.name = name

        # Initialize property
self._age = 0

    # An instance method. All methods take "self" as the first argument
def say(self, msg):
        print("{name}: {message}".format(name=self.name, message=msg))

    # Another instance method
    def sing(self):
        return 'yo... yo... microphone check... one two... one two...'
    # A class method is shared among all instances
    # They are called with the calling class as the first argument
    @classmethod
def get_species(cls):
return cls.species

    # A static method is called without a class or instance reference
    @staticmethod
def grunt():
return "*grunt*"
```

```
    # A property is just like a getter.
    # It turns the method age() into an read-only attribute of the same name.
    # There's no need to write trivial getters and setters in Python, though.
    @property
    def age(self):
        return self._age

    # This allows the property to be set
    @age.setter
def age(self, age):
self._age = age
    # This allows the property to be deleted
    @age.deleter
def age(self):
del self._age

# When a Python interpreter reads a source file it executes all its code. # This
__name__ check makes sure this code block is only executed when this #
module is the main program.

if __name__ == '__main__':
    # Instantiate a class
```

```
i = Human(name="Ian")
   i.say("hi")                    # "Ian: hi"
j = Human("Joel")
   j.say("hello")                 # "Joel: hello"
   # i and j are instances of type Human, or in other words: they are Human objects

   # Call our class method
   i.say(i.get_species())         # "Ian: H. sapiens"
   # Change the shared attribute
   Human.species = "H. neanderthalensis"
   i.say(i.get_species())         # => "Ian: H. neanderthalensis"
   j.say(j.get_species())         # => "Joel: H. neanderthalensis"
   # Call the static method
print(Human.grunt())              # => "*grunt*"

   # Cannot call static method with instance of object
   # because i.grunt() will automatically put "self" (the object i) as an argument
   print(i.grunt())               # => TypeError: grunt() takes 0 positional arguments but 1
was given

   # Update the property for this instance
   i.age = 42     #
Get the property
   i.say(i.age)                   # => "Ian: 42"
   j.say(j.age)                  # => "Joel: 0"
# Delete the property     del i.age
   # i.age                       # => this would raise an AttributeError
```

Example: Write a Python program to convert an integer to a roman numeral.

```
class py_solution:     def
int_to_Roman(self, num):
        val = [
            1000, 900, 500, 400,
            100, 90, 50, 40,
            10, 9, 5, 4,
            1
            ]
        syb = [
            "M", "CM", "D", "CD",
            "C", "XC", "L", "XL",
            "X", "IX", "V", "IV",
            "I"
            ]
        roman_num = ''
        i = 0         while  num > 0:
for _ in range(num // val[i]):
                roman_num += syb[i]
                num -= val[i]
i += 1
        return roman_num


print(py_solution().int_to_Roman(1)) print(py_solution().int_to_Roman(4000))
```

Exercises:

1. Write a Python class to get all possible unique subsets from a set of distinct integers Input:[4,5,6]
   Output : [[], [6], [5], [5, 6], [4], [4, 6], [4, 5], [4, 5, 6]]

2. Write a Python class to find a pair of elements (indices of the two numbers) from
   a given array whose sum equals a specific target number.
   Input: numbers= [10,20,10,40,50,60,70], target=50
   Output: 3, 4
3. Write a Python class to implement pow(x, n).
4. Write a Python class which has two methods get_String and print_String. get_String accept a string from the user and print_String print the string in upper case.

Additional Exercises:

1. Write a Python class to find validity of a string of parentheses, '(', ')', '{', '}', '[' and ']. These brackets must be close in the correct order, for example "()" and "()[]{}" are valid but "[)", "({[)]" and "{{{" are invalid.
2. Write a Python class to reverse a string word by word.
3. Write a Python class named Circle constructed by a radius and two methods which will compute the area and the perimeter of a circle.