ASSIGNMENT 2

QUESTION 1. What is the relationship between classes and modules?
ANSWER:

- Classes let us reuse the code with the help of concepts like inheritance, but in the same program.
- A module is just a file of any Python code, and by using the Python import statement we can import any such module into our current working program. This makes the code and variables in the imported module available to our current program.

QUESTION 2. How do you make instances and classes?
ANSWER :

- To create instances of a class, call the class using the class name and pass in the arguments its `__init__` method accepts.
- A class can be created by using the keyword `class`, followed by the class name.

QUESTION 3. Where and how should class attributes be created?
ANSWER :

Class attribute : any variable that is bound in a class is a class attribute.

```
class Book:
      font_size = 11
      font_name = 'Arial'
      page_size = 'A5'
obj = Book()
```

- Inside the class, all the references to class attributes (inside a method) should be with the class name; for example, `Book.font_size`.
- Outside the class, you should qualify all references to class attributes with the class name (for example `Book.font_size`) or with an instance of the class (for example `obj.font_size`, where `obj` is an instance of the class).

QUESTION 4. Where and how are instance attributes created?
ANSWER :

```
class Book:
      font_size = 11
      font_name = 'Arial'

      def TheAlchemist(self):
            self.page_count = 163
            self.page_size = 'A5'
obj = Book()
```

- Any attribute which is defined inside a Method using the first argument (usually self) of the method is called as an instance attribute.
- In the above class `Book`, all the attributes defined inside the method `'TheAlchemist'` which are referenced using `self` are called as instance attributes, for example (`self.page_count` and `self.page_size`).
- NOTE 1 : Inside a class, all references to instance attributes should be qualified with the `self` variable.
- NOTE 2 : Outside the class, all references to instance variables should be qualified with an instance of the class.

QUESTION 5. What does the term 'self' in a Python class mean?
ANSWER :

- self represents the instance of the class. By using the `'self'` we can access the attributes and methods of the class in python. It binds the attributes with the given arguments.
```
class Book:
      def __init__(self, book_name):
            self.name = book_name
```
- Python uses the self argument to find the right object's attributes and methods by passing the **`object`** to the `method` of the **`class`** as the **`self`** parameter.

QUESTION 6. How does a Python class handle operator overloading?
ANSWER :
To perform operator overloading, Python provides some special function or magic function that is automatically invoked when it is associated with that particular operator.
For example, `__add__(self, other)`

QUESTION 7. When do you consider allowing operator overloading of your classes?
ANSWER :
```
class Employee:
      def __init__(self, name, email_id, salary):
            self.name = name
            self.email = email_id
            self.salary = salary

emp1 = Employee('Arun', 'arun@gmail.com', 100)
emp2 = Employee('Tanu', 'tanu@gmail.com', 101)
print(emp1 + emp2)
```

If we run the `print` command it will throw a `TypeError`, because the compiler won't be able to understand what to add in the two objects named `emp1` and `emp2`. Since, behind the scenes, python is using `__add__` function so we can now overload the + operator to get the desired outputs.
The code below will add just the salaries. I could have done concatenation operation on strings using + operator as well but it doesn't make sense in this particular example.

```
class Employee:
      def __init__(self, name, email_id, salary):
            self.name = name
            self.email = email_id
            self.salary = salary

      def __add__(self, other):
            return self.salary + other.salary

emp1 = Employee(‘Arun’, ‘arun@gmail.com’, 100)
emp2 = Employee(‘Tanu’, ‘tanu@gmail.com’, 101)
print(emp1 + emp2)
```

`Output = 201`

QUESTION 8. What is the most popular form of operator overloading?
ANSWER :
Addition (+) operator. For example,

```
class Name:
   def __init__(self, name):
       self.name = name

   def __add__(self, other):
       return self.name + other.name

first_name = Name('Gaurav')
last_name = Name('Rajput')
print(first_name + last_name)
```

`Output = GauravRajput`

QUESTION 9. What are the two most important concepts to grasp in order to comprehend Python OOP code?
ANSWER :

Two most important concepts to grasp in order to comprehend Python OOP code are enumerated below:

1. *Concept of class* : without class, there is no OOPs, so one must be very well versed with the application of class and basics related to class like instance variables, class variables, use of self, objects etc; concepts that give basic understanding of class.

2. *Inheritance* : It’s a great way to eliminate unnecessary repetitive code. A child class can inherit from the parent class partially or entirely.

    a. Python is quite flexible with regards to inheritance. We can add new attributes and methods as well as modify the existing ones.