

EXPERIMENT 5

Q1. Python program to create and list of employees using Employee class. Program should also print total number of employees.

Employee class should have

- empcount(class variable)
- id and name(instance variable)
- constructor to set id, * set_name(), get_name (), get_id() methods (instance method)
- set_emp_count ()(class method)

CLASSES AND OBJECT:

A class is a user-defined blueprint or prototype from which objects are created. Classes provide a means of bundling data and functionality together. Creating a new class creates a new type of object, allowing new instances of that type to be made. Each class instance can have attributes attached to it for maintaining its state. Class instances can also have methods(defined by their class) for modifying their state.

- Class creates a user-defined data structure, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object.

Some points on Python **class**:

- Classes are created by keyword class.
- Attributes are the variables that belong to a class.
- Attributes are always public and can be accessed using the dot (.) operator. Eg.:Myclass.Myattribute

CLASS OBJECTS:

- An Object is an instance of a Class. A class is like a blueprint while an instance is a copy of the class with actual values. It's not an idea anymore, it's an actual dog, like a dog of breed pug who's seven years old. You can have many dogs to create many different instances, but without the class as a guide, you would be lost, not knowing what information is required.

An object consists of:

- State: It is represented by the attributes of an object. It also reflects the properties of an object.
- Behavior: It is represented by the methods of an object. It also reflects the response of an object to other objects.
- Identity: It gives a unique name to an object and enables one object to interact with other objects.

SELF:

Class methods must have an extra first parameter in the method definition. We do not give a value for this parameter when we call the method, Python provides it.

If we have a method that takes no arguments, then we still have to have one argument. This is similar to this pointer in C++ and this reference in Java.

init method : The init method is similar to constructors in C++ and Java. Constructors are used to initialize the object's state. Like methods, a constructor also contains a collection of statements (i.e., instructions) that are executed at the time of Object creation. It is run as soon as an object of a class is instantiated. The method is useful to do any initialization you want to do with your object.

CLASS VARIABLE: All objects share class or static variables. An instance or non-static variables are different for different objects (every object has a copy). For example, let a Computer Science Student be represented by class CSStudent. The class may have a static variable whose value is "cse" for all objects. And class may also have non-static members like name and roll. In C++ and Java, we can use static keywords to make a variable a class variable. The variables which don't have a preceding static keyword are instance variables. See this for Java example and this for C++ example. The Python approach is simple; it doesn't require a static keyword.

SYNTAX : `variable_name = value`

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, much like staticmethod. The difference between a static method and a class method is: Static method knows nothing about the class and just deals with the parameters.

SYNTAX :

@classmethod

def method_name(cls, ...):

function code

INSTANCE VARIABLE : Instance variables are owned by instances of the class. This means that for each object or instance of a class, the instance variables are different.

SYNTAX : self.variable_name = value

INSTANCE METHOD A class is a user-defined blueprint or prototype from which objects are created. Classes provide a means of bundling data and functionality together. Creating a new class creates a new type of object, allowing new instances of that type to be made. Each class instance can have attributes attached to it for maintaining its state. Class instances can also have methods (defined by its class) for modifying its state.

An instance method can access and even modify the value of attributes of an instance. It has one default parameter:-

self – It is a keyword which points to the current passed instance. But it need not be passed every time while calling an instance method. EXAMPLE :

def method_name(self,):

CODE:

```
class Employee:
    empCount = 0
    def __init__(self, ID):
        self.set_emp_count()
        self.ID = ID

    def set_name(self, name):
        self.name = name

    def get_name(self):
        return self.name

    def get_id(self):
        return self.ID

    @classmethod
    def set_emp_count(cls):
        cls.empCount += 1
n = int(input('ENTER NUMBER OF EMPLOYEES: '))
emps = []
print('Enter the name of Employees: ')
for i in range(n):
    temp = Employee(i+1)
    emp_name = input()
    temp.set_name(emp_name)
    emps.append(temp)

print('\n\nTOTAL NO OF EMPLOYEES : {} \n\n'.format(Employee.empCount))
print('***** DATA OF EMPLOYEES *****\n')
print('\tEmployee ID \t\t Employee Name')
for e in emps :
    print('        {} \t\t {}'.format(e.ID, e.name))
```

```

PS D:\Harsh\SEM 4\PYTHON\Assingment\EXP 5> python -u "d:\Harsh\SEM 4\PYTHON\Assingment\EXP 5\P1.PY"
ENTER NUMBER OF EMPLOYEES: 3
Enter the name of Employees:
Harsh
Mayank
Tanish

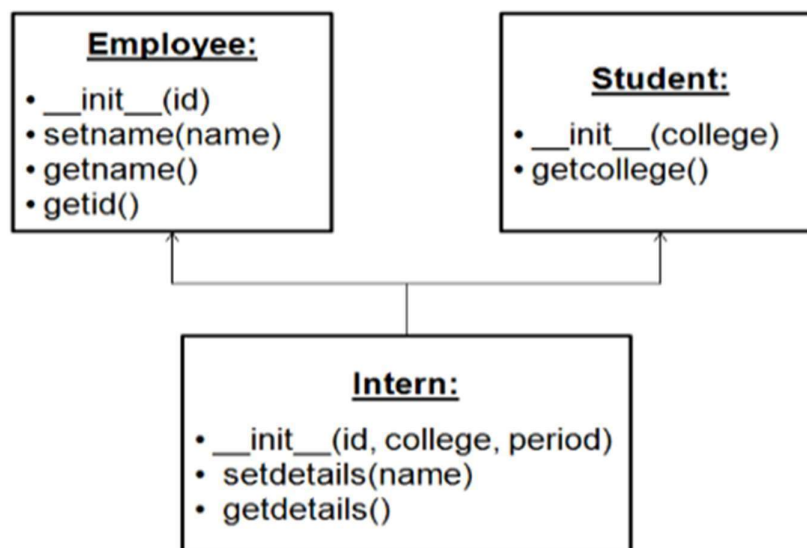
TOTAL NO OF EMPLOYEES : 3

***** DATA OF EMPLOYEES *****

Employee ID      Employee Name
1                Harsh
2                Mayank
3                Tanish
PS D:\Harsh\SEM 4\PYTHON\Assingment\EXP 5> 

```

Q2 Python program to demonstrate Multiple Inheritance



MULTIPLE INHERITANCE:

Inheritance is the mechanism to achieve the re-usability of code as one class(child class) can derive the properties of another class(parent class). It also provides transitivity i.e. if class C inherits from P then all the sub-classes of C would also inherit from P.

Multiple Inheritance When a class is derived from more than one base class it is called multiple Inheritance. The derived class inherits all the features of the base case.

CODE:

```
class Employee:
```

```
    def __init__(self, ID):  
        self.ID = ID
```

```
    def set_name(self, name):  
        self.name = name
```

```
    def get_name(self):  
        return self.name
```

```
    def get_id(self):  
        return self.ID
```

```
class Student:
```

```
    def __init__(self, college):  
        self.college
```

```
    def get_college():  
        return self.college
```

```
class Intern(Employee, Student):
```

```
    def __init__(self, ID, college, period):  
        self.ID = ID  
        self.college = college  
        self.period = period
```

```
    def set_details(self, name):  
        self.name = name
```

```
    def get_details(self):  
        print("***** Intern Details *****\n")
```

```

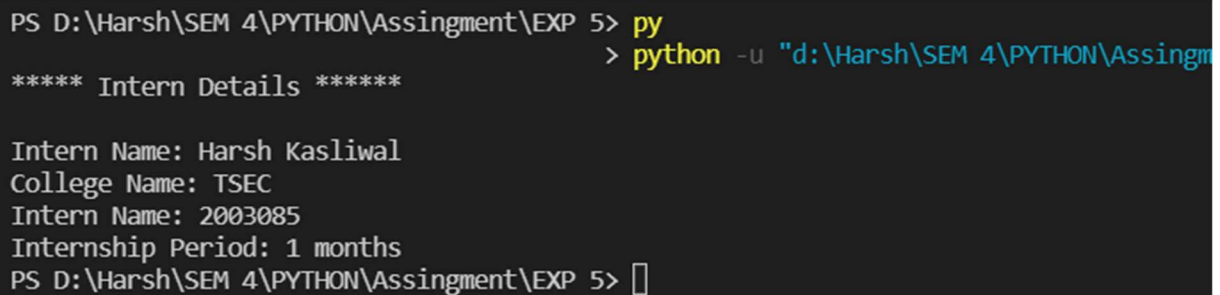
print('Intern Name:', self.name)
print('College Name:', self.college)
print('Intern Name:', str(self.ID))
print('Internship Period:', str(self.period), 'months ')

```

```

I1 = Intern(2003085, 'TSEC', 1)
I1.set_details('Harsh Kasliwal')
I1.get_details()

```



```

PS D:\Harsh\SEM 4\PYTHON\Assingment\EXP 5> py
> python -u "d:\Harsh\SEM 4\PYTHON\Assingment\EXP 5\intern.py"
***** Intern Details *****
Intern Name: Harsh Kasliwal
College Name: TSEC
Intern Name: 2003085
Internship Period: 1 months
PS D:\Harsh\SEM 4\PYTHON\Assingment\EXP 5>

```

Q3 Python program to overload greater than (>) operator to make it act on user defined classobjects

OPERATOR OVERLOADING:

Operator Overloading means giving extended meaning beyond their predefined operational meaning. For example operator + is used to add two integers as well as join two strings and merge two lists. It is achievable because '+' operator is overloaded by int class and str class. You might have noticed that the same built-in operator or function shows different behavior for objects of different classes, this is called Operator Overloading. Consider that we have two objects which are a physical representation of a class (user-defined data type) and we have to add two objects with binary '+' operator it throws an error, because compiler don't know how to add two objects. So we define a method for an operator and that process is called operator overloading. We can overload all existing operators but we can't create a new operator. 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, when we use + operator, the magic method add is automatically invoked in which the operation for + operator is defined.

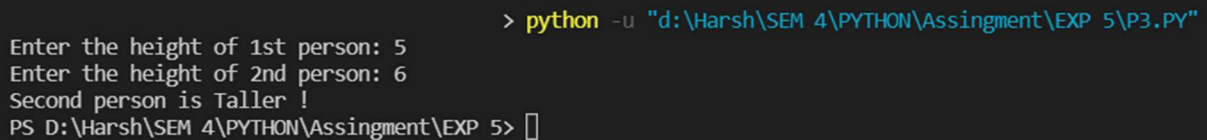
CODE:

```
class Person :
    def __init__(self, height) :
        self.height = height

    def __ge__(self, other) :
        return self.height >= other.height
h1 = int(input('Enter the height of 1st person: '))
h2 = int(input('Enter the height of 2nd person: '))

p1 = Person(h1)
p2 = Person(h2)

if(p1 >= p2) :
    print('First person is Taller ! ')
else :
    print('Second person is Taller ! ')
```



```
> python -u "d:\Harsh\SEM 4\PYTHON\Assingment\EXP 5\P3.PY"
Enter the height of 1st person: 5
Enter the height of 2nd person: 6
Second person is Taller !
PS D:\Harsh\SEM 4\PYTHON\Assingment\EXP 5> █
```

Q4 Python program to demonstrate concept of Interfaces. Program contains Printer interface and subclasses to send text to any printer.

INTERFACES:

1. **@abstractmethod**: An abstract class can be considered as a blueprint for other classes. It allows you to create a set of methods that must be created within any child classes built from the abstract class. A class which contains one or more abstract methods is called an abstract class. An abstract method is a method that has a declaration but does not have an implementation. While we are designing large functional units we use an abstract class. When we want to provide a common interface for different implementations of a component, we use an abstract class. By default, Python does not provide abstract classes. Python comes with a module that provides the base for defining Abstract Base classes (ABC) and that module name is ABC. ABC works by decorating methods of the base class as abstract and then registering concrete classes as implementations of the abstract base. A method becomes abstract when decorated with the keyword **@abstractmethod**.

2. pass: In Python programming, the pass statement is a null statement. The difference between a comment and a pass statement in Python is that while the interpreter ignores a comment entirely, pass is not ignored. However, nothing happens when the pass is executed. It results in no operation (NOP).

CODE:

```
from abc import *

class Printer(ABC):
    @abstractmethod
    def printit(self, text):
        pass

    @abstractmethod
    def disconnect(self):
        pass

class IBM(Printer):
    def printit(self, text):
        print(text)

    def disconnect(self):
        print('Printing completed on IBM Printer')

class HP(Printer):
    def printit(self, text):
        print(text)

    def disconnect(self):
        print('Printing completed on HP printer')

class Myclass:
    str = input("Enter the Printer name: ")
    classname = globals()[str]
    x = classname()
    x.printit('Hello, this is sent to printer')
    x.disconnect()
```

```
> python -u "d:\Harsh\SEM 4\PYTHON\Assingment\EXP 5\p4.py"
Enter the Printer name: HP
Hello, this is sent to printer
Printing completed on HP printer
PS D:\Harsh\SEM 4\PYTHON\Assingment\EXP 5> █
```