Q1. What is the purpose of Python's OOP?

In python Object Oriented Programming deals with the concept of class and Object. It is used to structure the python program into simple and reusable form(classes) of which we can create instance/object. It aims to implement to implement inheritance, abstraction, polymorphism, encapsulation.

Q2. Where does an inheritance search look for an attribute?

1. In inheritance what ever methods, variables, constructors parent class has by-default available to child class, there is not required to redefine them in child class. In case of code execution inheritance search look for attributes of parent class.
2. We use inheritance for code reusability and extendability.

Q3. How do you distinguish between a class object and an instance object?

1. Class object is a real object which is the blue print and built from a class.
2. Instance object is a virtual copy of the object
3. You can distinguish them by the address of the object. For instance object the address will be always same. But in case of class object the address of the object is different always.

Q4. What makes the first argument in a class’s method function special?

1. The first argument is a class method should be object itself. Self is the variable which indicates current class object.
2. It is not a keyword, only a naming convention.

Q5. What is the purpose of the \_\_init\_\_ method?

1. \_\_init\_\_ is the constructor. It is a special type of method which gets executed when an object is created. The main purpose of constructor is to initialize the instance variables.
2. It takes the first argument as self. In python name of the constructor is always \_\_init\_\_.
3. In python constructor is optional, if we are not creating ant constructor then default constructor will be given by PVM.

Eg :

class A:

def \_\_init\_\_(self):

self.B = 10

a = A()

print(a.\_\_dict\_\_) #displays the properties of the class

class Employee:

def \_\_init\_\_(self, eno, esal):

self.eno = eno

self.esal = esal

e = Employee()

print(e.eno)

print(e.esal)

Q6. What is the process for creating a class instance?

In python to create an object we have only one process. First we have to create a class by giving the keyword and class name. Then we need to create and object by giving paranthesis after the name of the class. To access it we need to assign it to a variable.

Eg :

class A:

def function(self, ch):

return ch

a = A()

print(a.function(‘S’))

Q7. What is the process for creating a class?

In python to create a class we need to write class name followed by class keyword and the we need to provide colon. As per OOP concept the class name should start with capital letter and then by small letter.

Eg:

class A:

def function(self, ch):

return ch

Q8. How would you define the superclasses of a class?

1. To define a super class of a class we need to write the super class name in parantheses after child class name while creating the child class. By this process the child class will extend the parent class attributes.
2. To access the attributes of parent class from child class, you need to create an object of child class.

Eg:

class Parent:

def p(self):

print('Parent method')

class Child(Parent):

def c(self):

print('Child Method')

c = Child()

print(c.c())

print(c.p())

Q1. What is the relationship between classes and modules?

1. To define a class we use class keyword.
2. Module is the combination of classes, functions and other modules
3. We can modify the attributes of the class by using inheritance. But we can simply use the codes written in a module by importing it.

Q2. How do you make instances and classes?

1. To make classes we use class keyword.
2. To make instance of a class you need to define \_\_init\_\_() method inside class. Then create object by passing the arguments it’s constructor takes then refer it to an instance.

Eg :

class Instance:

def \_\_init\_\_(self, name)

name = name

i = Instance()

print(i.name)

Q3. Where and how should be class attributes created?

Class attributes belong to class itself, they will be shared by all the instances. They are defined at the class body parts.

Q4. Where and how are instance attributes created?

Instance attributes are defined inside constructor which is defined directly inside a class using parameter self.

Q5. What does the term "self" in a Python class mean?

Self is a reference variable which indicates current class instance/object. And it is used to access variables which belong to the particular class, inside the class.

Q6. How does a Python class handle operator overloading?

In python + operator works as overloaded operator. It performs addition when we use it with int class. It performs concatenation when we use it with string and list class. To handle operator overloading python has inbuilt magic functions. (eg: \_\_add\_\_(self, other),\_\_sub\_\_(self, other), \_\_mul\_\_(self, other), \_\_truediv\_\_(self, other),\_\_floordiv\_\_(self, other), \_\_mod\_\_(self, other), \_\_pow\_\_(self, other), \_\_rshift\_\_(self, other), \_\_lshift\_\_(self, other), \_\_and\_\_(self, other), \_\_or\_\_(self, other), \_\_xor\_\_(self, other))

Q7. When do you consider allowing operator overloading of your classes?

Allowing operator overloading means using operators (+,-,\*,<,>,&,|,^) for different classes to perform different operations. We will allow operator overloading for the classes when we are dealing with same type classes (int-int, str-str, list-list) to perform either addition or concatenation operation.

Q8. What is the most popular form of operator overloading?

The most popular form of operator loading is +. Which is used for both addition and concatenation.

Q9. What are the two most important concepts to grasp in order to comprehend Python OOP code?

The most important concepts to grasp Python OOP code are

1. Inheritance
2. Polymorphism

Inheritance :

Acquiring the properties of one class to another class is the concept of inheritance.

It is used for code reusability and extendability.

Eg:

class Employee:

def \_\_init\_\_(self, eno, ename, esal):

self.eno = eno

self.ename = ename

self.esal = esal

def display(self):

print('Employee Number : ', self.eno)

print('Employee Name : ', self.ename)

print('Employee Salary : ', self.esal)

e = Employee(1118, 'Pragati', 100000)

e.display()

1. What is the concept of an abstract superclass?

i. Abstract means partially implemented.

ii. A class is called as abstract class if it contains at-least one abstract method. An abstract method is a method which is only declared but not implemented. It’s next level child class is responsible for implementation.

iii. Abstract class may not be instantiated.

Eg :

class ABC:

def m(self):

pass

class XYZ(ABC):

def m(self,a,b):

return a+b

1. What happens when a class statement's top level contains a basic assignment statement?

If the class statement’s top level contains a basic assignment stamen then it can be usable in further codes if we are using the same variable without re-assigning any other value. In case we are re-assigning any value PVM will consider the recent assigned value.

1. Why does a class need to manually call a superclass's \_\_init\_\_ method?

When we will create an object of the child class it will only execute the constructor present in particular class. If a class extends to a super class and we want to execute super class constructor then we need to call super class constructor manually from child class constructor with respwective arguments.

Eg :

class Parent:

def \_\_init\_\_(self,B,C):

print(B, C)

self.B = B

self.C = C

class Child(Parent):

def \_\_init\_\_(self, B, C):

super().\_\_init\_\_(B,C)

c = Child(10,20)

1. How can you augment, instead of completely replacing, an inherited method?

We can augment a method instead of completely replacing it by performing different operations.

Eg :

class ABC:

def m(self):

global a

a = 10

class XYZ(ABC):

def m(self):

b = 20

c = 30

return b+c

x = XYZ()

print(x.m())

5. How is the local scope of a class different from that of a function?

i. Local variables have the scope with in the function, method. It will get destroyed by destructor once the execution completes. If we will try to access them out side of that block we will get error.

ii. If we are creating a function then we can call it within the module, as it is provides functionality all across the module.