Python OOPS Interview Questions

1. How will you check if a class is a child of another class?

This is done by using a method called **issubclass()** provided by python. The method tells us if any class is a child of another class by returning true or false accordingly.

For example:

```
class Parent(object):
   pass

class Child(Parent):
   pass

# Driver Code
print(issubclass(Child, Parent))    #True
print(issubclass(Parent, Child))    #False
```

- We can check if an object is an instance of a class by making use of **isinstance()** method:

```
obj1 = Child()
obj2 = Parent()
print(isinstance(obj2, Child))    #False
print(isinstance(obj2, Parent))   #True
```

2. What is init method in python?

The **init** method works similarly to the constructors in Java. The method is run as soon as an object is instantiated. It is useful for initializing any attributes or default behaviour of the object at the time of instantiation.
For example:

```
class InterviewbitEmployee:

   # init method / constructor
   def __init__(self, emp_name):
       self.emp_name = emp_name

   # introduce method
   def introduce(self):
       print('Hello, I am ', self.emp_name)

emp = InterviewbitEmployee('Mr Employee')    # __init__ method is called he
emp.introduce()
```

3. Why is finalize used?

Finalize method is used for freeing up the unmanaged resources and clean up before the garbage collection method is invoked. This helps in performing memory management tasks.

4. Differentiate between new and override modifiers.

The new modifier is used to instruct the compiler to use the new implementation and not the base class function. The Override modifier is useful for overriding a base class function inside the child class.

5. How is an empty class created in python?

An empty class does not have any members defined in it. It is created by using the pass keyword (the pass command does nothing in python). We can create objects for this class outside the class.

For example-

```
class EmptyClassDemo:
   pass
obj=EmptyClassDemo()
obj.name="Interviewbit"
print("Name created= ",obj.name)
```

Output:

Name created = Interviewbit

6. Is it possible to call parent class without its instance creation?

Yes, it is possible if the base class is instantiated by other child classes or if the base class is a static method.

7. Are access specifiers used in python?

Python does not make use of access specifiers specifically like private, public, protected, etc. However, it does not derive this from any variables. It has the concept of imitating the behaviour of variables by making use of a single (protected) or double underscore (private) as prefixed to the variable names. By default, the variables without prefixed underscores are public.

Example:

```
# to demonstrate access specifiers
class InterviewbitEmployee:

    # protected members
    _emp_name = None
    _age = None

    # private members
    __branch = None

    # constructor
    def __init__(self, emp_name, age, branch):
        self._emp_name = emp_name
        self._age = age
        self.__branch = branch

    #public member
    def display():
        print(self._emp_name +" "+self._age+" "+self.__branch)
```

8. How do you access parent members in the child class?

Following are the ways using which you can access parent class members within a child class:

- **By using Parent class name:** You can use the name of the parent class to access the attributes as shown in the example below:

```
class Parent(object):
  # Constructor
  def __init__(self, name):
     self.name = name

class Child(Parent):
  # Constructor
  def __init__(self, name, age):
     Parent.name = name
     self.age = age

  def display(self):
     print(Parent.name, self.age)

# Driver Code
obj = Child("Interviewbit", 6)
obj.display()
```

- **By using super():** The parent class members can be accessed in child class using the super keyword.

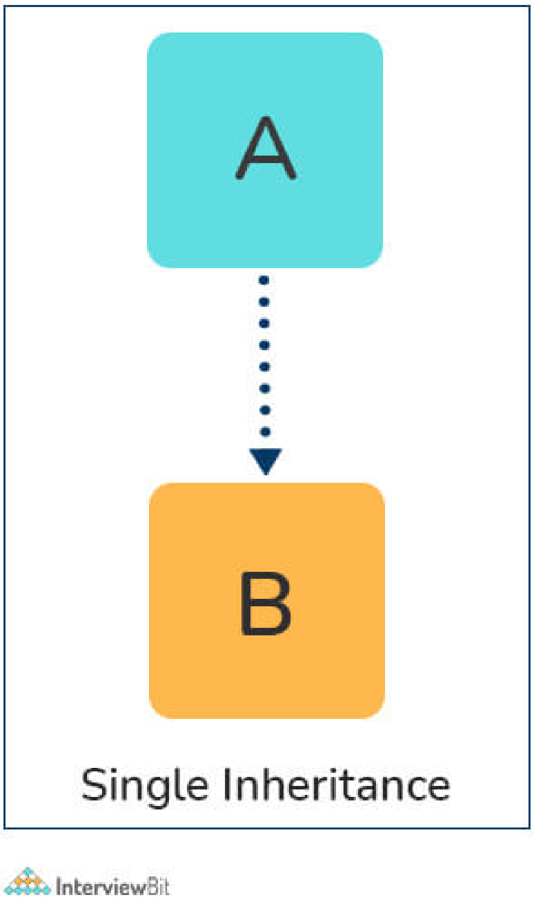
```
class Parent(object):
  # Constructor
  def __init__(self, name):
        self.name = name

class Child(Parent):
    # Constructor
    def __init__(self, name, age):
        '''
        In Python 3.x, we can also use super().__init__(name)
        '''
        super(Child, self).__init__(name)
        self.age = age

    def display(self):
        # Note that Parent.name cant be used
        # here since super() is used in the constructor
        print(self.name, self.age)

# Driver Code
obj = Child("Interviewbit", 6)
obj.display()
```

9. How does inheritance work in python? Explain it with an example.

Inheritance gives the power to a class to access all attributes and methods of another class. It aids in code reusability and helps the developer to maintain applications without redundant code. The class inheriting from another class is a child class or also called a derived class. The class from which a child class derives the members are called parent class or superclass.

Python supports different kinds of inheritance, they are:

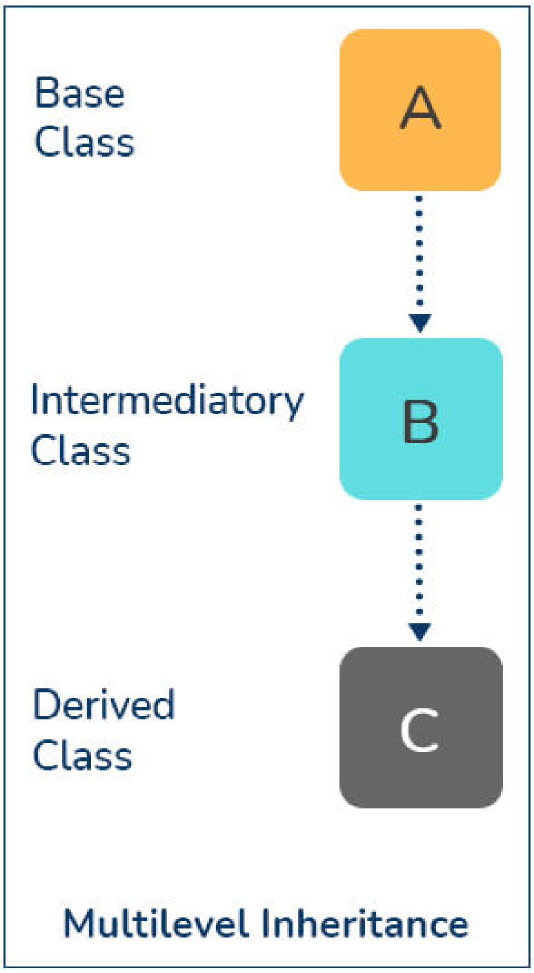
- **Single Inheritance**: Child class derives members of one parent class.



```
# Parent class
class ParentClass:
    def par_func(self):
        print("I am parent class function")

# Child class
class ChildClass(ParentClass):
    def child_func(self):
        print("I am child class function")

# Driver code
obj1 = ChildClass()
```

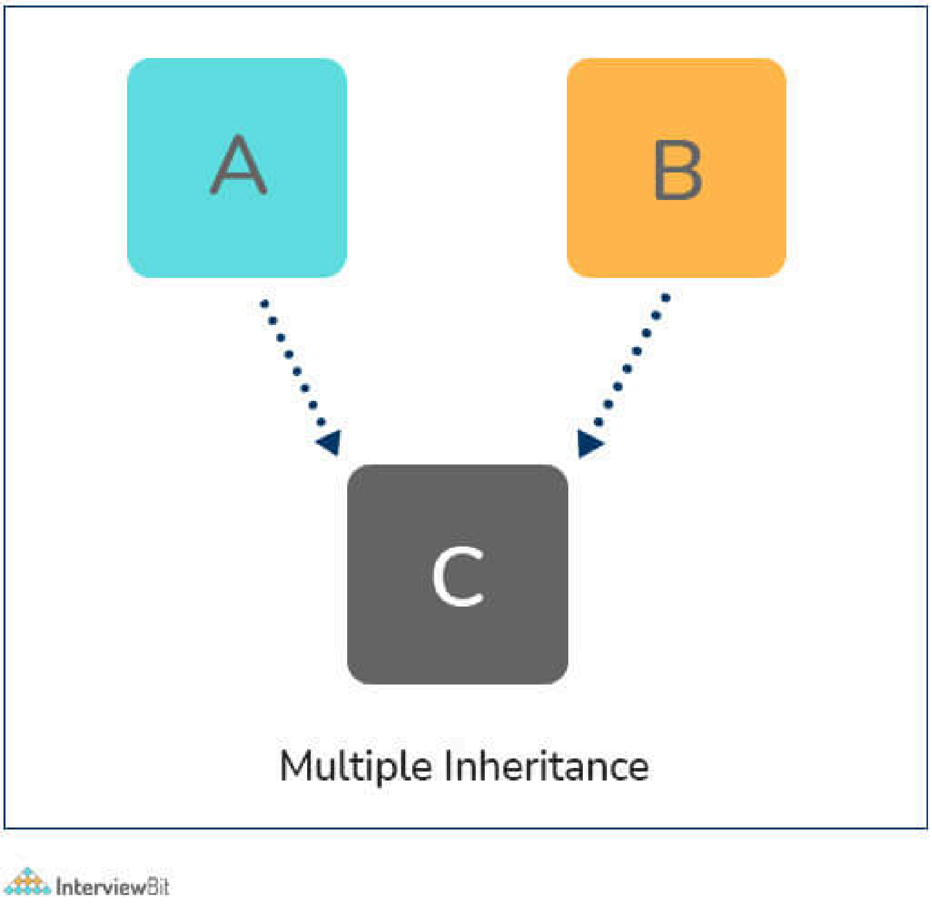
```
obj1.par_func()
obj1.child_func()
```

- **Multi-level Inheritance:** The members of the parent class, A, are inherited by child class which is then inherited by another child class, B. The features of the base class and the derived class are further inherited into the new derived class, C. Here, A is the grandfather class of class C.



```
# Parent class
class A:
  def __init__(self, a_name):
        self.a_name = a_name

# Intermediate class
class B(A):
    def __init__(self, b_name, a_name):
        self.b_name = b_name
        # invoke constructor of class A
        A.__init__(self, a_name)

# Child class
class C(B):
    def __init__(self,c_name, b_name, a_name):
        self.c_name = c_name
        # invoke constructor of class B
        B.__init__(self, b_name, a_name)

    def display_names(self):
        print("A name : ", self.a_name)
        print("B name : ", self.b_name)
        print("C name : ", self.c_name)

#  Driver code
obj1 = C('child', 'intermediate', 'parent')
print(obj1.a_name)
obj1.display_names()
```

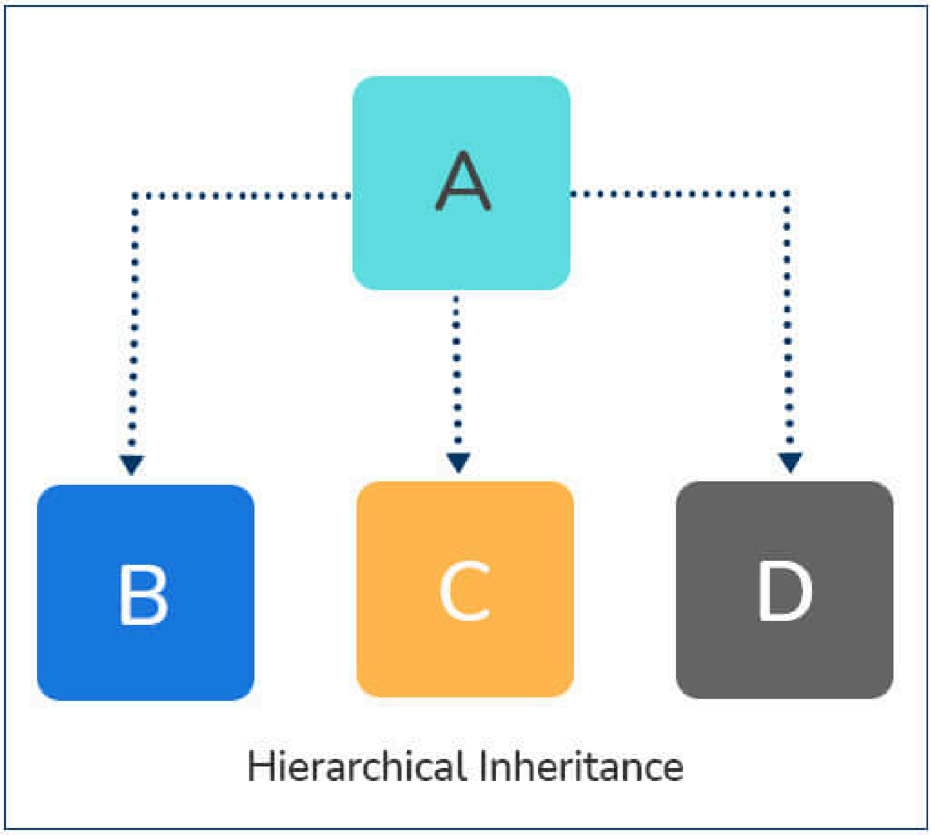
- **Multiple Inheritance:** This is achieved when one child class derives members from more than one parent class. All features of parent classes are inherited in the child class.



```
# Parent class1
class Parent1:
   def parent1_func(self):
       print("Hi I am first Parent")

# Parent class2
class Parent2:
   def parent2_func(self):
       print("Hi I am second Parent")

# Child class
class Child(Parent1, Parent2):
   def child_func(self):
       self.parent1_func()
       self.parent2_func()
```

```
# Driver's code
obj1 = Child()
obj1.child_func()
```

- **Hierarchical Inheritance:** When a parent class is derived by more than one child class, it is called hierarchical inheritance.



InterviewBit

```
# Base class
class A:
    def a_func(self):
        print("I am from the parent class.")

# 1st Derived class
class B(A):
    def b_func(self):
        print("I am from the first child.")

# 2nd Derived class
class C(A):
    def c_func(self):
        print("I am from the second child.")

# Driver's code
obj1 = B()
obj2 = C()
obj1.a_func()
obj1.b_func()    #child 1 method
obj2.a_func()
obj2.c_func()    #child 2 method
```

10. How do you create a class in Python?

To create a class in python, we use the keyword “class” as shown in the example below:

```
class InterviewbitEmployee:
   def __init__(self, emp_name):
       self.emp_name = emp_name
```

To instantiate or create an object from the class created above, we do the following:

```
emp_1=InterviewbitEmployee("Mr. Employee")
```

To access the name attribute, we just call the attribute using the dot operator as shown below:

```
print(emp_1.emp_name)
# Prints Mr. Employee
```

To create methods inside the class, we include the methods under the scope of the class as shown below:

```
class InterviewbitEmployee:
   def __init__(self, emp_name):
       self.emp_name = emp_name

   def introduce(self):
       print("Hello I am " + self.emp_name)
```

The self parameter in the init and introduce functions represent the reference to the current class instance which is used for accessing attributes and methods of that class. The self parameter has to be the first parameter of any method defined inside the class. The method of the class InterviewbitEmployee can be accessed as shown below:

```
emp_1.introduce()
```

The overall program would look like this:

```
class InterviewbitEmployee:
   def __init__(self, emp_name):
       self.emp_name = emp_name

   def introduce(self):
       print("Hello I am " + self.emp_name)
```

```
# create an object of InterviewbitEmployee class
emp_1 = InterviewbitEmployee("Mr Employee")
print(emp_1.emp_name)    #print employee name
emp_1.introduce()        #introduce the employee
```