Practical No 2

Aim : Write A Program To Implement Different Methods In Python.

1. Class Method In Python

This code snippet defines a class called `ClassMethod` in Python. The class has a class method named `call` which is decorated with `@classmethod`. A class method is a method that is bound to the class rather than an instance of the class. It can be called on the class itself, without creating an instance of the class.

In this code, the `call` method simply returns the string "I Am Class Method Example". The code then creates an instance of the `ClassMethod` class called `obj`. Finally, it calls the `call` method on the instance `obj` and prints the result.

2.Static Method In Python

This code defines a class `Static` with a static method `call()`. A static method is a method that belongs to a class, not an instance of the class. It's called using the class name or an instance of the class.

Here, an object `obj` is created from the `Static` class, and the `call()` method is called on that object. The method returns the string "I Am Static Method", which is then printed.

Note that since `call()` is a static method, it could also be called directly on the class, like this: `print(Static.call())`.

3. Single Inheritance

This Python code demonstrates inheritance, where the \`Child\` class inherits the methods of the \`Parent\` class. The \`Child\` class can call methods from both its own class and the \`Parent\` class.

In this example, \`object.func1()\` calls the method from the \`Parent\` class, and \`object.func2()\` calls the method from the \`Child\` class.

4. Multiple Inheritance

This code snippet demonstrates multiple inheritance in Python. It defines three classes: \`Mother\`, \`Father\`, and \`Child\`.

The \`Mother\` class has a \`motherName\` attribute and a method \`motherNameDisplay\` that returns the value of \`motherName\`.

The \`Father\` class has a \`fatherName\` attribute and a method \`fatherNameDisplay\` that returns the value of \`fatherName\`.

The `Child` class inherits from both `Mother` and `Father` classes, indicating multiple inheritance. It has a method `getParents` that prints the mother's and father's names by calling the corresponding methods from the inherited classes.

Finally, an instance of the `Child` class named `son` is created, and the `motherName` and `fatherName` attributes are set. The `getParents` method is then called to display the names of the mother and father.

5. Multi Level Inheritance

1. The code starts with the definition of the `Grandfather` class. This class has an `__init__` method that initializes an attribute `grandFatherName` when an instance of the class is created.

2. Next, the `Father` class is defined. It inherits from the `Grandfather` class using the `class Father(Grandfather)` syntax. The `Father` class also has an `__init__` method that initializes an attribute `fatherName` and calls the `__init__` method of the `Grandfather` class using the syntax `Grandfather.__init__(self, grandFatherName)`. This allows the `Father` class to inherit the `grandFatherName` attribute from the `Grandfather` class.

3. The `Son` class is defined. It inherits from the `Father` class using the `class Son(Father)` syntax. The `Son` class also has an `__init__` method that initializes an attribute `sonName` and calls the `__init__` method of the `Father` class using the syntax `Father.__init__(self, fatherName, grandFatherName)`. This allows the `Son` class to inherit the `fatherName` and `grandFatherName` attributes from the `Father` class.

4. The `Son` class also has a method `displayFamily` that prints the names of the son, father, and grandfather.

5. Finally, an instance of the `Son` class is created with the names "Himesh", "Dipesh", and "Alpesh". The `displayFamily` method is called to print the family details.

This code demonstrates how multi-level inheritance works in Python. It allows classes to inherit attributes and methods from their parent classes, creating a hierarchical structure.

6. Calling Of Method in Different Class

This code snippet is a simple example of how to use static methods in Python. It demonstrates the ability to call a static method from a different class.

In this code, `ClassA` is defined with a static method `method_in_class_a()`. This method simply prints the message "Method in classA" when called.

`ClassB` is defined with an instance method `call_method_from_class_a()`. This method first calls the static method `method_in_class_a()` from `ClassA` using the class name `ClassA.method_in_class_a()`. Then, it prints the message "Method in classB".

When an instance of `ClassB` is created and its method `call_method_from_class_a()` is called, both messages are printed.

7 . Hierarchical inheritance

This code demonstrates hierarchical inheritance in Python, where multiple child classes (`Child1`, `Child2`, `Child3`, `Child4`) inherit from a single parent class (`Parent`). Each child class has its own method (`callChild`) and inherits the `callParent` method from the parent class. The code creates objects of each child class and calls both the inherited and child-specific methods.

8 . Hybrid inheritance

This code snippet demonstrates the concept of hybrid inheritance in Python. It defines a parent class called `School` with a method `schoolName()` that prints "This is a school".

Then, it defines three child classes: `Student1`, `Student2`, and `Student3`. `Student1` and `Student2` inherit from the `School` class, while `Student3` inherits from `School`, `Student1`, and `Student2`.

Each child class has its own `studentName()` method that prints a specific student's name.

The code creates objects for each student type (`s1`, `s2`, and `s3`) and calls their respective `studentName()` and `schoolName()` methods to print the names and the school name.

9 . Super Keyboard

he `super()` keyword in Python is used to access methods and properties of a parent or sibling class. It allows a child class to call methods of its parent class, even if the child class has overridden those methods.

Why Use Super?

There are several reasons to use `super()`:

- **Method overriding**: When a child class overrides a method of its parent class, it can use `super()` to call the original method of the parent class.
- **Method extension**: A child class can use `super()` to extend the behavior of a method of its parent class.
- **Avoiding duplicate code**: By using `super()`, you can avoid duplicating code in a child class that is already present in its parent class.

9 . Method Overloading

this is an example of method overloading, then the `add` method is overloaded to handle different numbers of arguments.

In this case, the `add` method is overloaded to have three different "versions":

1. `add(a, b, c)`: This version takes three arguments and returns their sum.
2. `add(a, b)`: This version takes two arguments and returns their sum.
3. `add()`: This version takes no arguments and returns 0 (although this is not explicitly shown in the code, it is implied by the default values of `a`, `b`, and `c`).