OOPS

TOPICS

1. CLASS
2. OBJECT
3. ABSTRACTION
4. ENCAPSULATION
5. POLYMORPHISM
6. INHERITANCE
7. SUPER FUNCTION
8. CONSTRUCTOR
9. TYPES OF VARIABLES
10. TYPES OF METHODS

1. CLASS
    a. Class is a collection of data members and member functions
    b. Data members are also called as variables, class variables
    c. Member functions are also called as methods, class methods
    d. Both class variables and class methods together called as class members
    e. Class is a logical representation of the data
    f. Class is a blue print of the data
    g. Class doesn't hold any value

SYNTAX:

```
Class class_name:
        Variables
        Methods
```

EXPLANATION:

⇨ Class and class_name are mandatory
⇨ Variables and methods are optional

2. OBJECT
   a. Object is an instance of the class
   b. Instance means allocating memory to the class members
   c. Object is also called as reference variable
   d. With the help of the object we can access the class members
   e. Object is the physical representation of the data
   f. Object holds the value

SYNTAX:

```
reference_variable=class_name()
```

Calling the class members with the help of the object:

Syntax:

```
reference_variable.variable_name
reference_variable.method_name()
```

Ex for the class and object:

```
class Example:
    i=10
    def m1(self):
        print("iam method")
obj=Example()
print(obj.i)
obj.m1()
```

3. ABSTRACTION
   a. Hiding the implementation and showing the functionality is known as abstraction
   b. Hiding the unwanted information to the user
   c. Abstraction is possible with the help of the abstract classes

4. ENCAPSULATION
   a. Combining the related information into the single unit is known as encapsulation
   b. Encapsulation is possible with the help of the classes
5. POLYMORPHISM
   a. Poly means many, morph means many forms
   b. One task performing in different ways is known as polymorphism or one name different functionalities is known as polymorphism

TYPES OF POLYMORPHISM

There are mainly two types:

1. COMPILE TIME POLYM ORPHISM
   a. It is achieved by the overloading concept
      There are 3 types of overloading:
      1. Method overloading
      2. Constructor overloading
      3. Operator overloading

METHOD OVERLOADING

⇨ In a class we can have same method name but with
   - Different no.of parameters
   - Order of the parameters(signature)
   - Datatype of the parameters

⇨ Python doesn't support method overloading and constructor overloading, if we write forcefully the last method will be executed

Ex:

```
def add(a,b):
    print(a+b)
def add(a,b,c):
    print(a+b+c)
def add(a,b,c,d):
    print(a+b+c+d)
add(1,2,3,4)
```

OPERATOR OVERLOADING

- ⇨ It is achieved with the help of the magic methods
- ⇨ In python every operator will have the magic methods

Syntax:

```
def __add__(self,other):
    return self.variable_name+other.variable_name
```

Ex:

```
class Book:
    def __init__(self,pages):
        self.pages=pages
    def __add__(self, other):
        return self.pages+other.pages
    def __gt__(self, other):
        return self.pages>other.pages
book1=Book(300)
book2=Book(200)
print(book1+book2)
print(book1>book2)
```

2. RUN TIME POLYMORPHISM
    - ⇨ It is achieved with the help of the overriding concept
    - ⇨ Overriding means redefining the functionality of parent class method with the help of the child class method
      There are mainly two types of overriding:
      1. Method overriding
      2. Constructor overriding

METHOD OVERRIDING

Ex:

```
class Parent:
    def m1(self):
        print("iam method of parent class")
class Child(Parent):
    def m1(self):
        print("iam method of child class")
childobj=Child()
childobj.m1()
```

6. INHERITANCE
   Inheritance is the process of extracting the properties of one class to another class
   ADVANTAGES:
    - ⇨ Code reusability
    - ⇨ Reduces the code repetition
    - ⇨ It saves the time
    - ⇨ It improves the readability, scalability, maintainability

   There are mainly 5 types of inheritance:

   1. Single inheritance
   2. Multiple Inheritance
   3. Multilevel inheritance
   4. Hierarchical inheritance
   5. Hybrid inheritance

SINGLE INHERITANCE

- ⇨ Deriving a class from single base class is known as single inheritance
- ⇨ In simple words, one parent one child

Syntax:

```
Class Parent:
      Stmts
Class Child(Parent):
      Stmts
```

Ex:

```
class Parent:
   def add(self,a,b):
       print(a+b)
class Child(Parent):
   def sub(self,a,b):
       print(a-b)
obj=Child()
obj.add(1,2)
obj.sub(4,2)
```

MULTIPLE INHERITANCE

- ⇨ Deriving a class from more than one base class is known as multiple inheritance
- ⇨ In simple words, two parents one child

Syntax:

```
Class parent1:
        Stmts
Class parent2:
        Stmts
Class child(parent1,parent2):
       Stmts
```

Ex:

```
class Parent1:
    def m1(self):
        print("iam parent1")
class Parent2:
    def m2(self):
        print("iam parent2")
class Child(Parent1,Parent2):
    def m3(self):
        print("iam child of parent1 and parent2")
childobj1=Child()
childobj1.m1()
childobj1.m2()
childobj1.m3()
```

MULTILEVEL INHERITANCE

- ⇨ Deriving a class from another derived class is known as multilevel inheritance
- ⇨ In simple words, child of another child

Syntax:

```
Class Parent:
      Stmts
Class child1(Parent):
      Stmts
Class child2(child1):
     Stmts
```

Ex:

```
class GrandParent:
   def m1(self):
       print("iam the grand parent")
class Parent(GrandParent):
   def m2(self):
       print("iam the child of the grand parent")
class Child(Parent):
   def m3(self):
       print("iam the child of the Parent")
obj=Child()
obj.m1()
obj.m2()
obj.m3()
```

HIERARCHICAL INHERITANCE

⇨ Deriving more than one classes from single base class
⇨ In simpe words, one parent more childrens

Syntax:

```
Class Parent:
    Stmts
Class Child1(Parent):
    Stmts
Class Child2(Parent):
    Stmts
```

Ex:

```
class Parent:
    def m1(self):
        print("iam the parent")
class Child1(Parent):
    def m2(self):
        print("iam the first child")
class Child2(Parent):
    def m3(self):
        print("iam the second child")
parentobj=Parent()
child1obj=Child1()
child2obj=Child2()
parentobj.m1()
child1obj.m2()
child2obj.m3()
```

HYBRID INHERITANCE

⇨ The combination of more than one inheritance is known as hybrid inheritance like hierarchical and multiple etc

Syntax:

```
Class Parent:
    Stmts
Class Child1(Parent):
    Stmts
Class Child2(Parent):
    Stmts
Class Child3(Child1,Child2):
    Stmts
```

Ex:

```
class Child1(Parent):
    def m2(self):
        print("iam the first child")
class Child2(Parent):
    def m3(self):
        print("iam the second child")
class Child3(Child1,Child2):
    def m4(self):
        print("iam the child of child1,child2")
Child3obj=Child3()
Child3obj.m1()
Child3obj.m2()
Child3obj.m3()
Child3obj.m4()
```

7. SUPER() FUNCTION
   - ⇨ super() function is the built in function
   - ⇨ It is used to access the super class constructor, methods, variables

Syntax:

```
super().__init__()
super().method_name()
```

Ex:

```
class Parent:
    def __init__(self):
        print("iam parent constructor")
    def m(self):
        print("iam method of parent")
class Child(Parent):
    def __init__(self):
        super().__init__()
        print("iam child  constructor")
    def m(self):
        super().m()
        print("iam method of child method")
obj=Child()
obj.m()
```

8. CONSTRUCTOR
   a. Constructor is a special kind of method which is used to initialize the values to the variables
   b. In python, the constructor name is __init__(self)
   c. self holds the memory address of instance/object
   d. Constructor will be called automatically when the instance/object is created
   e. If you don’t define any constructor, the python will create the default constructor when the instance/object is created

Ex:

```
class Person:
    def __init__(self,name,age):
        self.name=name
        self.age=age
    def display(self):
        print(f"name={self.name} age={self.age}")
obj=Person("raju",30)
obj.display()
```

9. TYPES OF VARIABLES

There are mainly 3 types of variables:

1. Local variable/method level variable
2. Static variable/class level variable
3. Instance variable/object level variable

INSTANCE VARIABLE

⇨ Instance variables are defined
- inside the constructor by using self
- inside the method by using self
- outside the class by using object reference variable

⇨ For every instance a copy of the instance variable will be created

⇨ If we modify the value of one object it will not reflect in another object

Ex:

```
class Person:
    def __init__(self,name,age):
        self.name=name
        self.age=age
    def display(self):
        print(f"name={self.name} age={self.age}")
person1=Person("raju",30)
person2=Person("ramu",20)
person1.display()
person2.display()
print("modifying the age of person1")
person1.age=40
print("after modifying")
person1.display()
person2.display()
```

STATIC VARIABLE

⇨ Static variables are defined

- Inside the constructor by using class name
- Inside the instance method by using class name
- Inside the class method by using class name or “cls” variable
- Inside the static method by using class name
- Outside the class by using class name or object reference variable
- For all the instances only one copy of variable will be available
- if we modify the value of one object it will reflect in all the objects

Ex:

```
class Student:
    college="svs"
    def __init__(self,name):
        self.name=name
    def display(self):
        print(f"name={self.name} college={self.college}")
std1=Student("raju")
std2=Student("ramu")
std1.display()
std2.display()
print("changing the college name")
Student.college="IARE"
std1.display()
std2.display()
```

10. TYPES OF METHODS

There are mainly 3 types of methods:

1. Instance method
2. Class method
3. Static method

INSTANCE METHOD

- ⇨ Inside the method implementation if we are using instance variables then such type of method is called instance method
- ⇨ In the instance method we have to pass atleast one parameter i.e., self
- ⇨ Instance method is called with the help of object
- ⇨ With in the class we can call the instance method by using “self” variable
- ⇨ Outside the class by using object reference variable

Ex:

```
class Person:
    def __init__(self,name,age):
        self.name=name
        self.age=age
    def display(self):
        print(f"name={self.name} age={self.age}")
person1=Person("raju",30)
person1.display()
```

CLASS METHOD

- ⇨ Inside the method implementation if we are using “class” variables then such type of method is called class method
- ⇨ It is declared with @classmethod decorator
- ⇨ Inside the class method we can call the variables by using “cls” variable or object reference variable

Ex:

```
class Student:
    college="svs"
    @classmethod
    def modify(cls):
        cls.college="IARE"
        print(cls.college)
obj=Student()
obj.modify()
```