OOPS Concepts in Python

* → python is an object oriented programming languag

1. class
2. object
3. Abstraction
4. Inheritance
5. Encapsulation
6. polymorphism

1. class :-

→ It is blueprint or template for creating object

→ It defines the attributes and methods of that objects of the class.

→ Create class by using 'class' keyword

Eg:-

```
class Bank :
    print ("Anil")
```

2. object :-

→ It is instance of a class.

→ we can access the attribute and methods of class by using object.

Eg:- obj = Bank ( )

* __init__() Method

→ when class initiated, the '__init__()' method is automatically called.

Note:- The 'self' parameter is reference to the current instance of the class. and it is used to access variables that belongs to the class.

Example:-

```
class Bank:
    def __init__(self, ac_NO, Bank_name, bal):
        self.AC_NO = ac_NO
        self.Bank_Name = Bank_Name
        self.Bal = bal

    def Balance(self):
        print(self.Bal)

    def withdraw(self, amount)
        self.Bal = self.Bal - amount

    def deposite(self, amount)
        self.Bal = self.Bal + amount

obj1 = Bank(123, "SBI", 5000)        → object creation
obj1.withdraw(1000)
print(obj1.Balance) ⇒ 4000 → Result
```

3. Abstraction

→ Hiding complex implementation details and showing only essential features of the object.

Example:-

```
From abc import ABC, abstractmethod

Class shape(ABC):
    @abstractmethod
    def area(self):
        pass

class Rectangle(shape)
    def __init__(self, width, height)
        self.width = width
        self.height = height
    def Area(self)
        return self.width * self.height

rectangle = Rectangle(5,4)
print(rectangle.Area())
```

20 ⟶ Result

Note:-

→ If class contains one or more than one abstract method then it is called abstract class.

→ The method without implementation it is called abstract method

4. Inheritance

→ Create a new class from already created class.

→ In new class we can access attributes and methods from all created class.

→ Types of inheritances

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

1. Single inheritance:

→ Create a single child class from single parent class.

Example:-

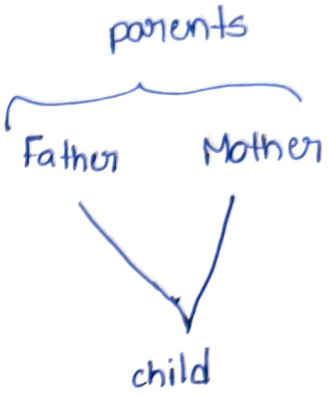
parent
↓
child

```
class parent:
    def method1(self)
        print("parent")

class child(parent):
    def method2(self)
        print("child")

child_obj = child()
child_obj.method1() → parent
child_obj.method2() → child
```

→ Results

2. Multiple inheritance

→ Create single child class from more than one parent class



Example:-

```
class Father:
    def method1(self):
        print("Father")

class Mother:
    def method2(self):
        print("Mother")

class child(Father, Mother):
    def method3(self):
        print("child")

obj-child = child()
obj-child.method3() ⟶ child
obj-child.method1() ⟶ Father
obj-child.method2() ⟶ Mother
```

} output

```
print(child.__mro__)
```

B.

output:-

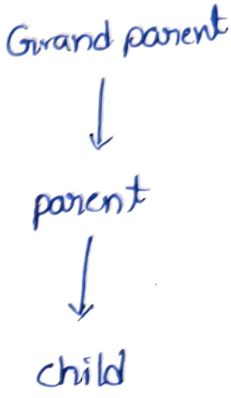
(<class '__main__.child'>, <class '__main__.Father'>, <class '__main__.Mother'>, <class 'object'>)

Note:- mro ⟶ method resolution order

3. Multilevel Inheritance

→ Create a parent class from grand parent class. and also & create a child class from parent class

Example:-



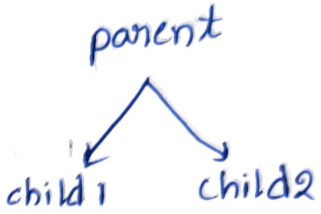

```
class grandparent:
    def method1(self):
        print("grand parent")

class parent(grandparent):
    def method2(self):
        print("parent")

class child(parent):
    def method3(self):
        print("child")

obj_child = child()
obj-child.method1()
obj_child.method2()
obj_child.method3()
```

output:-

```
grand parent
parent
child.
```

4. Hierarchical Inheritance

→ Create more than one child class from single parent class.



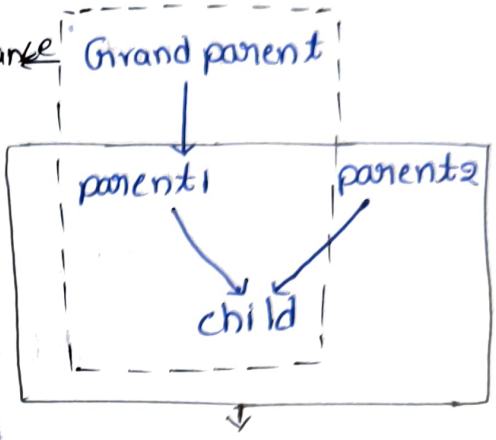
Example:-

```
class parent:
    def method1(self):
        print("parent")

class child1(parent):
    def method2(self):
        print("child1")

class child2(parent):
    def method3(self):
        print("child2")

obj_child1 = child1()
obj_child1.method1()  → parent
obj_child1.method2()  → child1
obj_child2 = child2()
obj_child2.method1()  → parent
obj_child2.method3()  → child2
```

output

5. Hybrid Inheritance:-

→ Combination of more than one inheritance.



Example:-

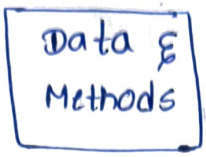
```
class grandparent:
    def method1(self)
        print ("grand parent")
class parent1 (grandparent):
    def method2(self):
        print ("parent 1")
class parent2:
    def method3 (self):
        print ("parent2")
class child (parent1, parent2)
    def method4 (self)
        print ("child")
```

```
obj_child = child()
obj_child.method1() → grand parent
obj_child.method2() → parent 1
obj_chid.method3() → parent2
obj_child.method4() → child
```

} output

5. Encapsulation

→ Combine the data & methods in single unit ग्रा class



class

Reasons for using encapsulation :-

1. Code will be organized and clean
2. prevent the data from accidental removal & deletion
3. Abstractio → Hides implementation logic & allows only use
4. Data hiding

* Access specifiers ग्रा Access modifiers

1. public access specifiers
2. protected access specifiers
3. private access specifiers

} using for security purpose.

1. public access specifier :-

→ if data declare in public, we can access the data

* In same class
* By object
* we can access subclass & we can access by subclass obj.

Example:-

```
class parent:
    publicdata = 20
    def method1(self):
        print(self.publicdata)

class child(parent):
    def method2(self):
        print(self.publicdata)

obj1 = parent()
obj1.method1()          → 20
print(obj1.publicdata)  → 20
obj2 = child()
obj2.method2()          → 20
print(obj2.publicdata)  → 20
```

} outputs

Note:- we can access by class, class object, subclass and subclass object.

2. protected access specifier:-

→ if data is declare in protected, it can accas only in same class and subclass.

→ Just it is signal, we can eaccess like public only

→ For protected we need to [single underscore] befor data & methods.

Example :-

```
class parent:
    _protecteddata = 20
    def method1(self):
        print(self._protecteddata)

class child(parent):
    def method2(self):
        print(self._protecteddata)

obj1 = parent()
obj1.method1()  ⟶ 20
obj2 = child()
obj2.method2()  ⟶ 20
print(obj._protecteddata) ⟶ 20
```

output

3. private access specifier :

→ If data and method declare as private, we can access only in the same class.

→ It is just way of signal for developers.

→ For private we need to '__' [double underscore] before data & methods.

Example:-

```
Class parent:
    --privatedata =20
    def method1(self):
        print(self.--privatedata)

Class child (parent):
    def method2(self):
        print(self.--privatedata)

obj1 = parent()
obj1. method1()  ---> 20
obj2 = child()
obj2. method2()  ---> it's through error
```

Note:- we can access in the same class. By using name manling we can access by subclass and objects.

Example:- (Name manling)

```
print(obj1._parent__privatedata) -> 20
```

↓ output

_<class name>
we Need to add
before private data variable

Note:- TO avoid name collisions we use {public, private, protected} in python

6 polymorphism

→ The word polymorphism means "Many forms".

→ In programming it refers to methods / operators with same name that can executed many objects or many tasks.

→ An ability to do more than one task.

Example:- 1. Example in operator (+)

```
print (3+4)  → '+' act as sum
'7'

print ("Hi" + "Anil") → '+' act as concanterate
Hi Anil
```

2. Example in function

```
def sum(*args):
    if args:
        → start=0
        for i in args:
            start = start + i
        return start
```

output:

```
print (sum (1,2,3))  ——→ 6
print (sum ([1,2,3],[4,5,6])) → [1,2,3,4,5,6]
print (sum ('Hellow','world')) → Hellow world
```

Note:- In above sum function work as sum and concatenate

1. Method overloading

→ If the class contain more than one method has same name &

→ The method contains different data types of parameters (or) different number of parameters Both is called method overloading.

Examples:-

1. Same method name with different type of parameters

```
Class A:
    def sum(int,int):
        pass
    def sum(str,str):
        pass
```

2. Same method with different number of parameters

```
Class B:
    def sum(int,int):
        pass
    def sum(int,int,int):
        pass
```

2. Method overriding:-

→ super class (parent class) & subclass (child class) has same method name. if we access the method in subclass then only sub class method accessed without accessing super class method.

Rules for Method overriding:-

1. Super class and sub class must be present (Inheritance)
2. Declare two classes with with same method and same parameter
3. Logic must be different in methods
4. Method override will be done when we access the same method in sub class. if we access in super class the method will not override.

Example:-

```
class parent:
    def method (self):
        print ("parent")

Class child (parent):
    def method (self):
        print ("child")
obj = child()
obj.method() ⟶ 'child' → output.
```