29/Dec/2022

Object-Oriented Programming

Coffee-making Machine.

Everything is object in python, and object will of a class.
Generality to Specificity

* you can create your own datatype.
* different way of doing coding than traditional programming.

Class Kya hai? — blueprint.
— a = 2
a is object of 'int' class.
— datatype is a class, variable is object

class
- Property / Data
- function / behaviour.

* Class name should be in pascal case

↓

where every character is join and their first letter is capital e.g ThisIsCool.

Cameal case = pascal Not camel

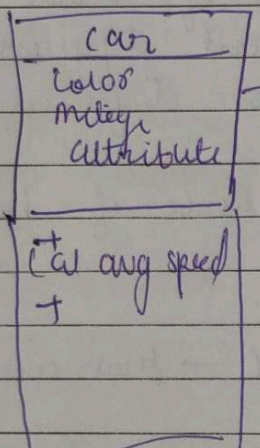
Snake case = we put underscore ranivathore

Syntax

class classname :

~~def~~ # data

function.



→ class representation

① Method are special function return inside class

② calling of method and function are different

eg function ⇒ len(List)

method ⇒ list.append( )

③ function can by access by anone but only class object can access method.

Special method → __init__(self): constructor
→ we declare our variable inside this method in python, unlike java

* constructor is a special method whose inside code will execute when you create object, without even calling.

* In java constructor name and class name same but in python constructor name always __init__( )

Object is a instance of class. means we can store different different value in same variable on different property
eg
balance is keeping balance of hdfc and sbi object born

* user doesn't have control → that code written in constructor.

* Self → object.

print (id(self)) → printing address of self

understand class X:
def __init__(self):
print (id (self))

obj = X()
print (id(obj))

Both are same that means Humhara Object hi self hai.

Very Imp

~~Part kharta~~ → 9

Object Storiya ☺

* A fundamental rule of object ~~oriented~~ oriented programming says that class contains only two things data and method and only object of that class have right to access them means only class object can access its attributes and method.

* Even two method of same class cannot call each other or access each other.

* when we create our object

Object = class ()

unintentially

→ this means here we are ~~unitentelly~~ passing our object to ~~oby~~ our class through a special method which we called constructor which evoked even without being called by ~~constructo~~r object.

→ now this object (current object of class) is our self, with the help of whom we access method inside class.

12/Sept/2022

Another special method is __str__(), this method will automatically calls when you pass your object into print function.

```
class X:
    def __init__(self):
        self.x = 5

x = X()
print(x) → Hash code
```

```
def __str__(self):
    return Hii

print(x) → Hii
```

* 3 magic method __add__() — whenever you put + btw two object, automatically this method invoked

instance variable → object ke liye variable ka value har alag hota hai.

Encapsulation

lets take an example

```
class X:
    def __init__(self):
        self.a = 0
        self.b = 0
```

if I create object of this class X.

obj = X()

means I can access this class X, data variables a and b. of course

10/September

Task 1 → Create ~~doc~~ own datatype.

Maths

Obj.a

→ this means I can modify this a and b value

Obj.a = "xyz"

Obj.b = "virus"

Boom !! this mean this can be Security thread in java we can use access modifier private to make them secure in python we do this using double underscore

~~now~~ ~~you~~

```
def __init__(self):
    self.__a=0
    self.__a=0
```

now you can't access this data members and you can also hide your methods.

Internal Working

we have class name Atm having data member __pin at the time of execution, python internally convert it into _Atm__pin.

now if some is even modifying pin it will create a new object.

eg.

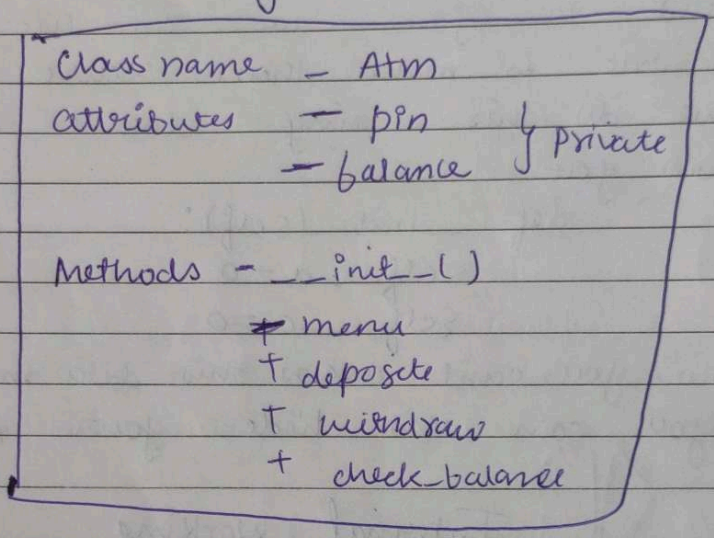
obj.__pin = "Virus" → ~~code unaffected~~ your code is unaffected

obj._Atm__pin = "Virus" → error.

this means nothing in python is truly private - because python is for adults.

if you want to access the hidden variable you can create method for that getter and setter method and in setter method you can write logic according to you.

Class Diagram



Reference variable

from xyz import Atm
- creating object of this class

>> Atm()
→ we have not stored its address (memory location)
→ because object is lost.

technically sbi = Atm()
sbi is a variable name, pointing to that address where actual address of object is stored

and this variable is known as reference variable as argument.

→ You can pass object into function, and function can return object as well.

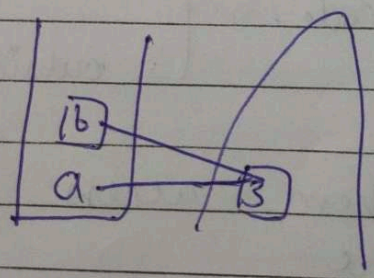
Intresting

```
class Customer:

    def __init__(self, name):
        self.name = name

    def greet(customer):
        print(id(customer))

cust = Customer("Rani")
print(id(cust))
```

Question - will they both return same or different value ?

answer :) they will return same value because we are passing reference means

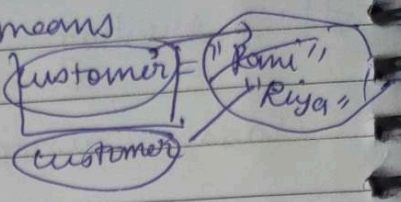
```
a = 3
and b = a
print(b)
same
```



class ke objects are mutable. prove ?

```
def greet(customer):
    print(id(customer))
    customer.name = "Nitish"
    print(customer.name)
    print(id(customer))
```

same means




```
cust = Customer("Rani")
```

means object are mutable.

Pass by reference — mutable object pass — Permanent changes. (at same location)

~~Eclipse~~ Static :) Need ?

Two type of variable
→ Instance variable → value diff. for diff. object
→ class variable | Static variable :) value same of all object.

eg. IFC code.
↳ outside constructor

to access them access them using class name
Self.sno = Atm.counter

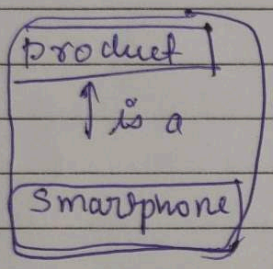
Static method are method who doesn't require object. to access member of class.

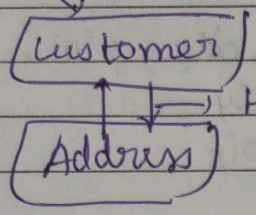
@staticmethod
def get_counter():
    return Atm_Counter

Relationship between Two class.

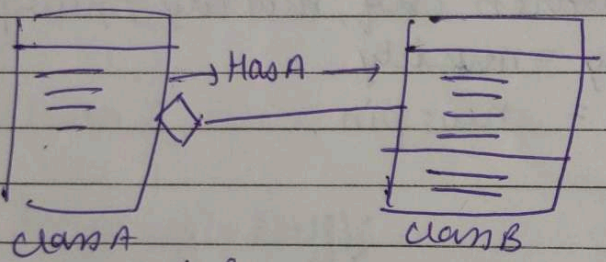
Agregation (Has-A)

Inheritance (Is-A)





Class Diagram Agression



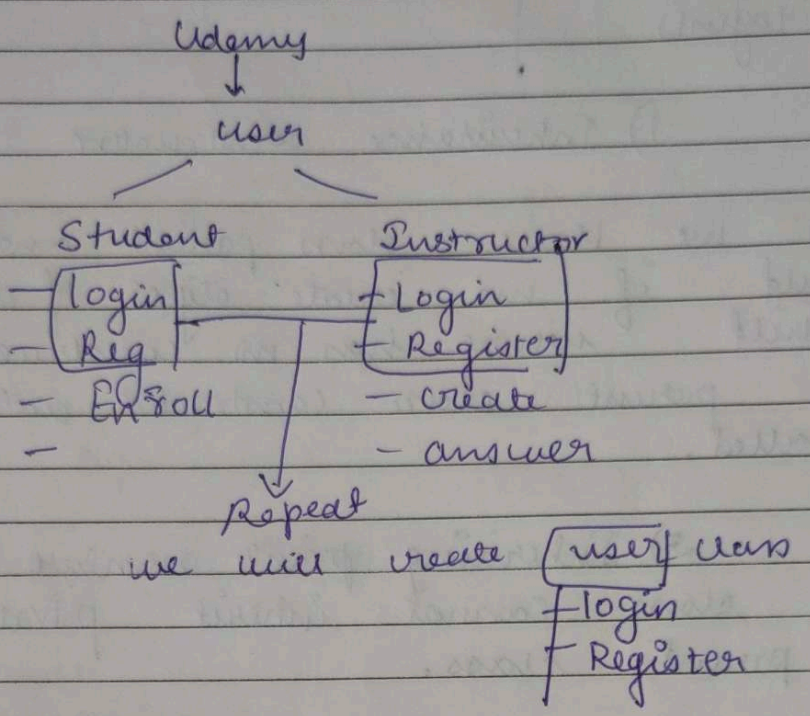
this shows Agregation

```
class Customer:

    def __init__(self, name, gender, add):
        self.name = name
        self.gender = gender
        self.addres = addres

    def editprofile(self, new, name, newcity, newpin):
        self.name = name
        self.addres.change_addres(newcity, newpin)

class Addres:

    def __init__(self, city, pin):
        self.city = city
        self.pin = pin

    def change_addres(self, new_city, new_pin):
        self.city = new_city
        self.pin = new_pin


add = Addres("Kolkata", 700)
cust = Customer('Nitish', 'male', add)

cust.editprofile(          )
```

DRY → Don't Repeat Yourself.

Inheritance → Code Reusability

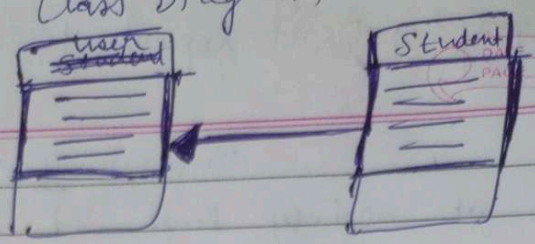
born at society level and biological concept too.



* private members are not inherited.

```
class user:

    def login(self):
        print('login')

    def Register(self):
        print('Register')

class Student:

    def enroll(self):
        print('Enroll')
```

Class Diagram



Stu1 = Student()
Stu1.enroll()
Stu1.verify()
Stu.login()

1) Inheritance constructor

Suppose we Have class parent and class child if we create object of class child whose has no constructor then parent class constructor will be called.

2) Inheriting private member
child class cannot inherit private member of parent class.

3) Method overriding → Polymorphism
if we Have 2 class parent and child and child class doesn't Has constructor then if we create object of child class then parent class constructor would be ~~invoked~~ Evoke but with this object if they call method whose name in both class is same then child class method will be called.

4) super should be first statement inside method | class.

```
class Phone:
    def __init__(self, price, brand, camera):
        {{{{}}}}

    def buy(self):
        print("Buying a phone")

class Smartphone(phone):

    def buy(self):
        super.buy()

S = Smartphone(2000, "Apple", 13)
S.buy()
S.super().buy()
```

Super Keyword

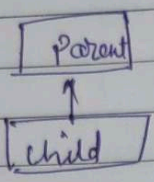
5) With ~~super~~ ~~method~~ you can access parents method and constructor, you can't access its attributes and you can't use it outside class.

```
6) class Parent:
    def __init__(self): self.num = 100
   class child(parent):
    def __init__(self): super().__init()
        self.var = 200

    def show(self):
        print ~~~
```

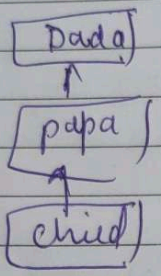
10/Sept/2022

Types of Inheritance

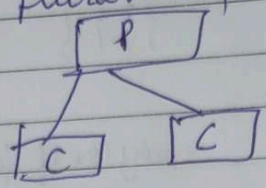
1. Single level



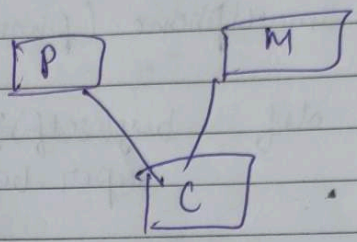
2) Multi-level



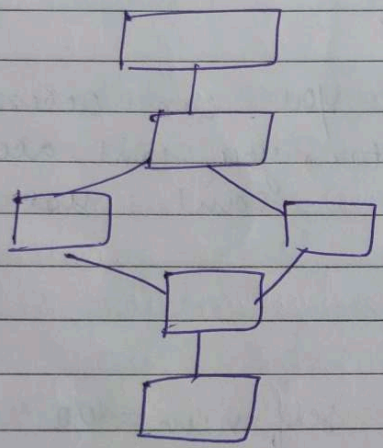
3) Hierarchical



3) Multiple Inheritance



4) Hybrid Inheritance



```
class A:
  ~~
  ~~
class B(A):
  ~~
  ~~
```

Single

```
class A:
  ~~
class B(A):
  ~~
  ~~
class C(B):
  ~~
  ~~
```

Multilevel

Multiple Inheritance

```
class A:
    ~ ~
class B:
    ~ ~
    ~ ~
class C(A,B):
    ~ ~
    ~ ~
```

if class C doesn't have constructor ~~then~~ it will call class A & whoever comes first ~~before~~ ~~visited~~ ~~sir~~ in order C(A,B)

~~Multi~~ Multiclass

```
class A
    def m1(self):
        return 20

class B(A):

    def m1(self):
        val = super.m1() + 30

class C(B):

    def m1(self):
        val = self.m1() + 20
        return val

obj = C()
print(obj.m1()) → return error
```

recursion occurs n ~~times~~ and no breaking condition.

Polymorphism

multiple cases

1. Method overriding - different class, same name function
2. Method overloading - same class, same function name

```
class Geometry:

    def area(self, radius):
        return 3.14 * radius ** 2

    def area(self, l, b)
        return l * b.
```

```
obj = Geometry()
obj.area(4,5)   → valid in java
obj.area(4)     → not valid in python
```

technically method overloading doesn't work in python.
but we can do method overloading by

```
def area(self, a, b=0):
    if (b==0):
        return 3.14 * a * a
    else
        return a * b.
```

3 Operator overloading :) behaving differently then usual behaviour.

eg 3+4=7
'3'+'4'='34'
done using magic operator.

Indian Railway account

What is abstract class?
* which has has abstract Method.
↓
which are declared but not defined
* you can't create its instance.

pure virtual function
→ Base class → declared
→ subclass → redefined