Drawback → PoP

① Global data (not bind easily)

# OOPS #

Day 1 to Day 5
05, 06, 07, 08, 09, 10

01.

Date: 04/06/20 (2.)

**Object →:** Objects are real world entites.

→ Anything you can describe in this world is an object.

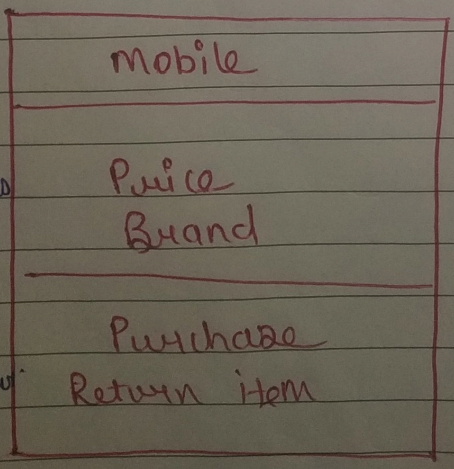
→ Classes on the other hand are not real.

→ They are just a concept.

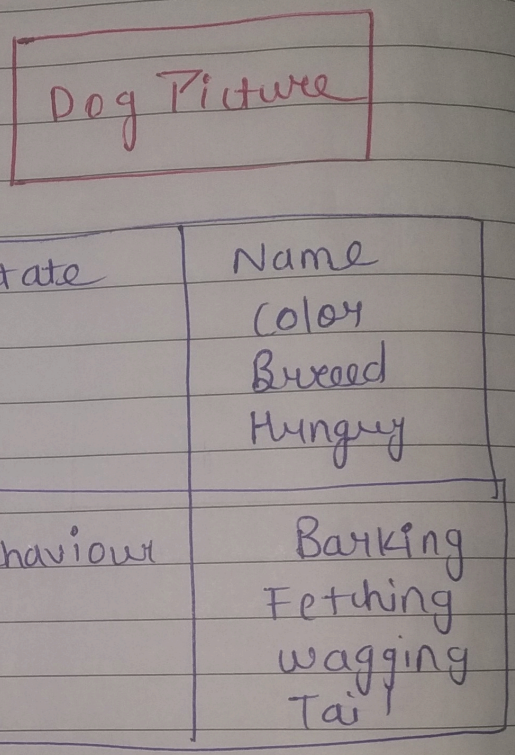
→ Class is a short form of classification.

→ A class is a classification of certain objects & its just a description of the properties and behaviour all objects of that classification should possess.

Ex →:




Real world object share two characteristics →:




→ the object Dog has both state & behaviour

→ Class is a blueprint.

→ A class contains the properties / attributes of the object & the operations / behaviour that can be performed on an object.

# How to define a class:→

→ class is defined using class keyword in Python.

Ex→

```
class Mobile:
    Pass
```

Python built-in classes:→

→ List → Tuple
→ Date → Set

# How to create object:→

→ To create object we need a class.

Syntax is →

<classname> ()

where classname is name of class

→ Ex→ Mobile()
→ Mobile()

object

# Object literals:→

→ Object literal is a special syntax through which objects are created & intialized

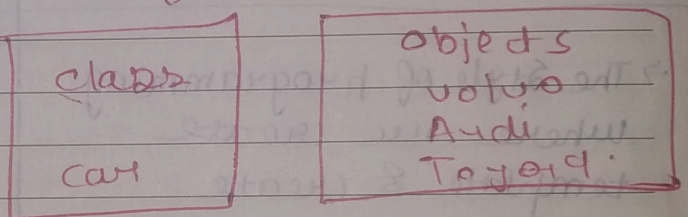
Ex→

list1 = [1, 2, 3]

↓

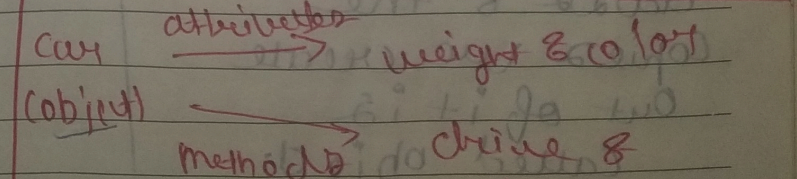
object literal for list.

Ex→ of object & class



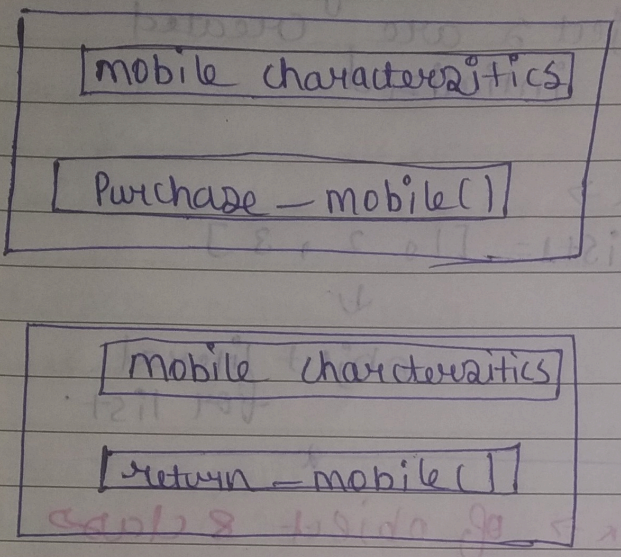

→ class is a template for objects & object is an instance of a class.

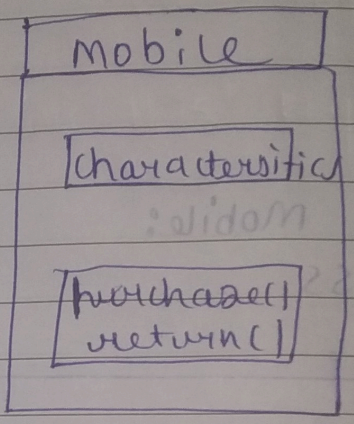
⇒ When individual objects are created, they inherit all the variables & methods from the class.



# Remaining Topic → object oriented Programming #

# clubbing data & Behaviours →:




→ The style of programming where we create template & create copies from that template is called object-oriented Programming

→ This style allows us to code for scenarios closely linked with real life:

→ The template we create is called a class & the copies we create out of it is called object.

write on P.NO

(1)

Previous Part

# Accessing objects :→

→ Just like we need variables to access & use values

→ we need variables to access & reuse the objects we create

→ Such variables that are used to access objects are called reference variables

Ex
mob1 = Mobile()
mob2 = Mobile()

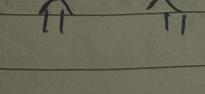
# Attributes of an object :→

→ can be created using . dot operator

Syntax →

reference_variable.attribute_name = value

Ex →



0 1 2 3

```
class Mobile:
    pass

mob1 = Mobile()
mob2 = Mobile()

mob1.price = 2000
mob1.brand = "Apple"

mob2.price = 3000
mob2.brand = "Samsung"
```

| Regular variable | Attribute |
|---|---|
| x = 5 | m1.color = "Green" |
| Update<br>x = 5<br>x = 6 | m1.color = "Red" |
| A variable can assigned to another variable<br>y = x | c1 = m1.color |

1 → 1 0 3 4
2 → 3
3 → 2
4 → 0
5 → 1
6 → 8
7 → Graph
8 → 3

9 → 2
10 → 3 9 4

Java (# , 8 other const. allow us to create different set of attributes for different object like Python does

# How do we create Attributes in a class:->

→ Attributes can be added to a class through special function called __init__().

Ex →:

```
class Mobile:
    def __init__(self, brand, price):
        self.brand = brand
        self.price = price

mob1 = Mobile("Apple", 2000)
mob2 = Mobile("Samsung", 3000)
```

Parameters

who allocate size to object.

↓ (constructor)

self → This tells the interpreter to deal with the current object.

↳ like 'this' keyword in Java

# Constructor →:

→ When we create object, the special __init__() function inside the class of that object is invoked automatically.

→ The special function is called Constructor.

→ Constructor is called when Object is created

OOA

↓

OOD →

↓

OOP (Working Program)

↳ how things should be done (implementation specification)

→ In Python everything is an object. Thus every thing would have either attributes or behaviour or both. Ex → list, String etc. all are objects

# How do we create behaviour in a class?

→ We can create behaviour in a class by adding functions in a class.

→ However such functions should have special Parameter called self as the first Parameter

Ex →

```
class mobile:
    def __init__(self):
        print("Inside constructor")

    def purchase(self):
        print("Purchasing a mobile")

mob1 = mobile()
mob1.purchase()
```

O/P →

Inside constructor
Purchasing a mobile

# How do we Access An attribute in a method?

→

Ex →:

We can access an attribute in a method by using self.

```
class mobile:
    def __init__(self, brand, price):
        self.brand = brand
        self.price = price

    def purchase(self):
        print("Purch. a mobile")
        print("This mobile has brand", self.brand, "and price", self.price)

print("mobile-1")
mob1 = mobile("Apple", 2000)
mob1.purchase()

print("mobile-2")
mob2 = mobile("Samsung", 3000)
mob2.purchase()
```

O/P →

07.

# Methods :→

→ In Python everything is an object.

→ Thus, everything would have either attributes or behaviour or both.

→ That means, even no, string, list, set, dictionary etc all are treated as objects in Python.

Ex →
(12.5).is_integer()

↳ Here we invoking a method is_integer() on numerical value.

→ That means numerical values are all objects.

→ A Python method is like a Python function, but it must be called on an obj. And to create it, you must Pu

# Difference →

| Function | method |
|---|---|
| → It is a block of code with a name | → It is Part of object & represent the behaviour of object |
| Invoked process | Invoked process |
| Ex → len([1,2,3]) It can be invoked using name of func & Passing Parameter | → without an object we cannot invoke a method Ex → [1,2,3].reverse() |
| Parameters are optional in function | A method have at least one Parameter : Self |

Remaining Topics
→ Types of constructors.

# Read details about constructor & all.

056

08.

# Abstraction:→

→ The ability to use something without having to know the details of how it's working is called abstraction.

Date:→ 05/06/20

# Printing an object:→

```
class Shoe:
    {
        → content
    }

s1 = Shoe(1000, "canvas")
print(s1).
```

→ It will display the internal hex representation of it.

⇒

⇒

⇒

# For more readable output:→

→ For more readable output when printing an object we can use inbuilt special — str — method.

→ This method must return STRING & this string will be used when the object is printed.

# Invoking methods:→

→ We can also invoke one method from another using self.

```
class mobile:
    def display(self):
        print("Details")

    def purchase(self):
        self.display()
        print("Calc. details")

Mobile().purchase()
```

09.

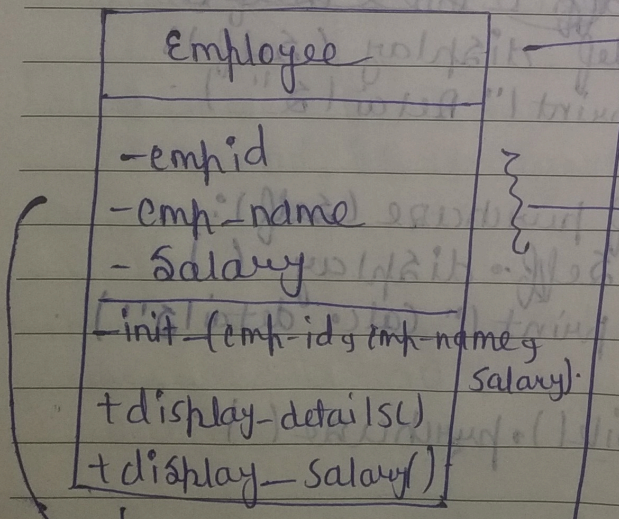
# Class Diagram:→

→ To ensure that programmers all over understand each other properly, we need a common way of representing a class

→ This is called as class diagram

→ It has 4 Parts:→

→ the name of the class,
→ the list of attributes,
→ the list of methods & access specifiers




(+) or (-) indicates access specifiers

✓** In class Name first character is captialized.

10.

* want to know Python Philosophy → DS7

import this

# Quick Summary →:

→ Reference variables hold the object's

→ An object can have multiple reference variable

→ Assigning a new reference variable to an existing object doesn't create a new object.

→ class diagram represent the class with its attributes & behaviours

! Encapsulation:→

Ex →

```
class Customer:
    def __init__(self, cust_id
        , name,
        age, wallet
        _balance):
        self.cust_id = cust_id
        self_name = name
        self.age = age
        self.wallet_balance
            = wallet_balance

    def update_balance(self,
                       amount):
        if amount < 1000 and
                amount > 0:
            self.wallet_balance += 
                amount
    def show_balance(self):

        print("The balance is",
              self.wallet_balance)


c1 = Customer(100, "GOPAL", 24, 1000)
c1 = update_balance(500)
```

Date:⇒ 07/06/20.

# Encapsulation:⇒

→ Adding a double underscore makes the attribute to a private attribute

→ Private attributes are those which are accessible only inside the class

→ This method of restricting access to our data is Encapsulation.

# Accessing private variables:⇒

→ Since we know the name of the variable changes when we make it private, we can access it by using modified name.

→ So, if a private variable can be accessed outside the class & can be modified.

✱✱ Languages like Java, C# don't allow access of private variable to the class.

# Getter & Setter methods:⇒

→ To have a error free way of accessing & updating private variables, we create specific method for this

→ Those methods which are meant to set a value to a private variable are called Setter methods

&

→ methods meant to access private variable values are called getter methods.

→ Setter methods are also as mutator methods

⇒ & getter methods are called accessor methods (as they access the values).

methods:>

free,
g &
e
create
d by

which
set a
rivate
called

t to
variable
called
.

are called
nethods

nods

thods
cess
).

Ex>
of Getter Setter &
private variables

```
class Example:

    def __init__(self):
        self.__num(5)

    def set_num(self,num):
        self.__num=num

    def get_num(self):
        return self.__num

obj=Example()
obj.set_num(6)
print(obj.get_num())
```

private variable

setter

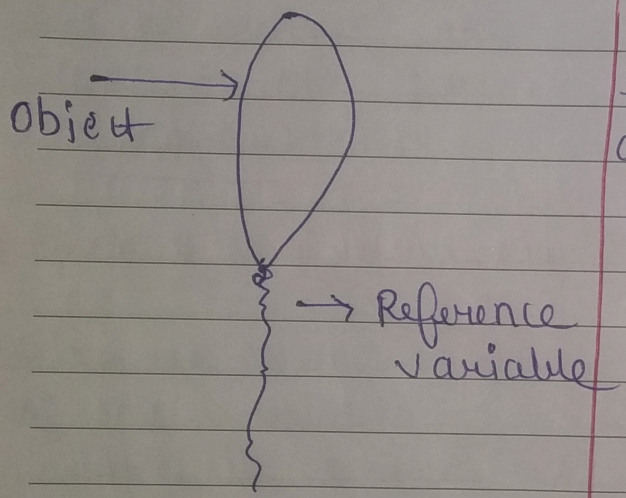
getter

O/P → 6

# Day → 2:

## # Helium balloon & reference variable: →

→ An object is like a ballon & the reference variable is like the ribbon connecting it to the ground.




Ex →

```
class Mobile :

    def __init__(self, price, brand):
        self.price = price
        self.brand = brand

mob1 = Mobile(1000, "Apple")
print(mob1.price)
```

## # Can one ballon have multiple ribbons →:

→ Just like a ballon can have multiple ribbons, an object can have multiple reference variable.

→ Both the references are refering to the same object.

→ when you assign an already created object to a variable, a new object is not created.

Ex →

```
mob1 = Mobile(1000, "Apple")
mob2 = mob1
```

← same as

same — " —

An object may be referred by more than one reference variables.

059

A.

→ Just like the balloon with multiple ribbons, if we change the attribute of an object through reference variable it immediately affects in other reference variable as there is only one balloon ultimately.

Ex →

```
mob2 = mob1
mob2.price = 3000
```

⇓

To solve this problem make another reference variable refer to another object

Ex →

```
class Table:
    def __init__(self):
        self.no_of_legs = 4
        self.glass_top = None
        self.wooden_top = None

dining_table = Table()
back_table = Table()
Front_table = back_table()
back_table = dining_table()
```

How many objects & reference variables will be there at the end of line 9.

→ Reference variable = dining_table
→ back_table
→ front_table

→ object →

Table()

back_table() & dining_table()

↓

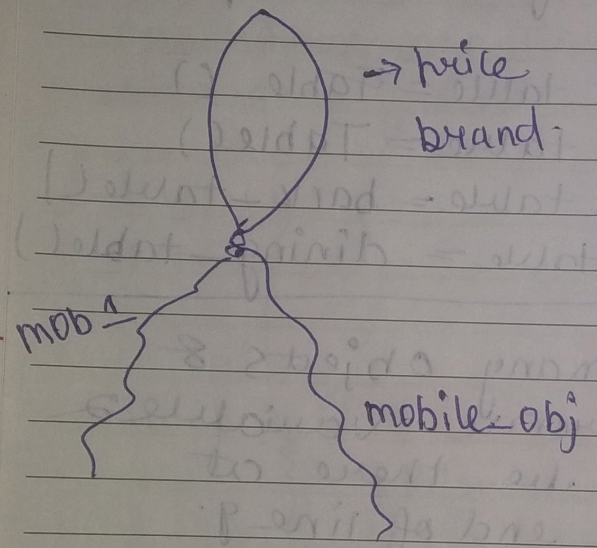
(1).

** Detail meal Padhna

Imp → ** Adding 9 —— in front of attribute makes it ~~private~~

# Pass by reference: →

→ When we pass an object to a Parameter, the Parameter name becomes reference variable.



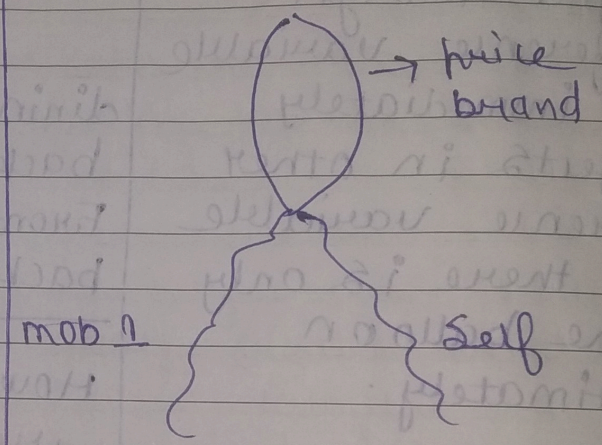

→ Thus there is one object with the two reference variable, one the formal Parameter & the actual Parameter

→ Thus any, changes made through one reference variable will affect the other as well.

# Self: →

→ Self is not a keyword

→ Self refers to the current object being executed.




→ without Self we are only creating a local variable & not an attribute.

give ex of shopping mall

# Static →:

# Shared attributes →:

→ what we need is a way to make an attribute shared across objects.

→ The data is shared by all objects, not owned by each object.

→ Thus by making a single change, it should reflect in all objects to do.

Ex →:

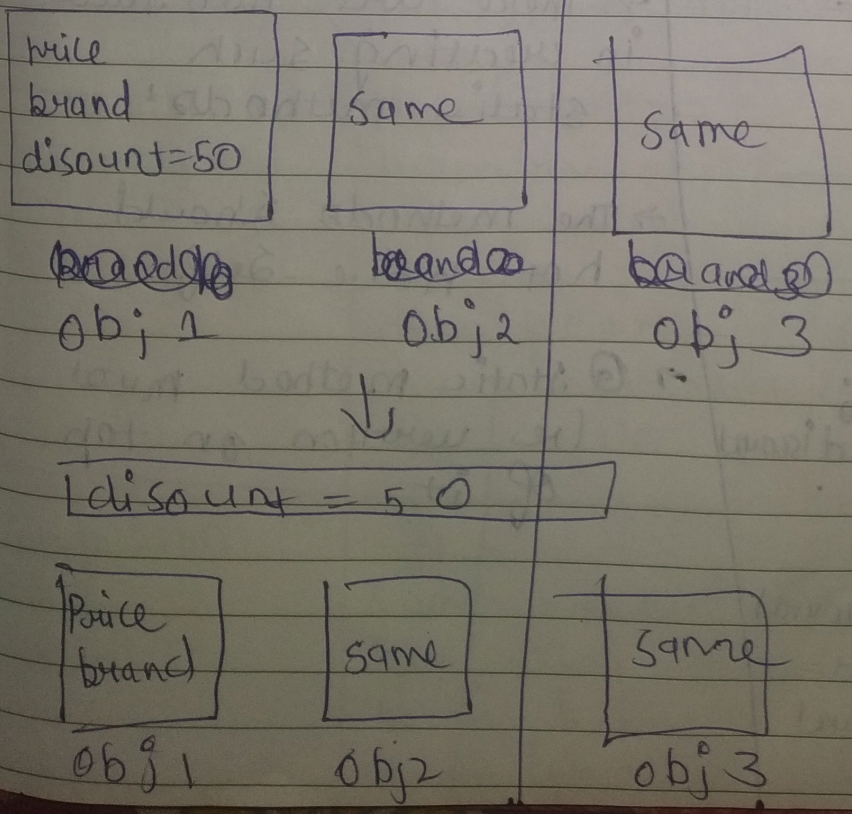
→ we can create shared attributes by placing them directly inside the class & not inside the constructor

→ And since the attribute is not owned by any one object, we don't need the self to create this attribute.

→ Such variables which are created at a class level is called static variables

→ Here discount is static variable

Ex → class Mobile:
    discount = 50
    def __init__(self, price, brand):
        self.price = price
        self.brand = brand



# Accessing static variables:

→ we can access static variables using the class name itself.

→ Static variable belongs to the class & not an object.

→ Hence we don't need self to access static variables.

# Static variables and Encapsulation:

→ we can make our static variables as a private variable by adding a double underscore in front of it

Ex →

__discount = 50

ex → @ S.M
def get_discount():
return Mobile.__discount

@ S.M
def set_discount(discount):
mobile.__discount =

# Static methods:

→ Static variable are object independent, we need a way to access the getter setter methods without an object.

→ This is Possible by creating static methods.

→ Static methods are those methods which can be accessed without an object.

→ They are accessed using class name.

→ There are two rules in creating such static methods:

→ The methods should not have self.

→ @ Static method must be written on top of it

Day → 3

18.

# Read about Instance variable

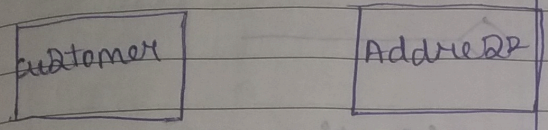
061

Date → 09/06/207:

# Using static variable as a counter :→

# Realtionships :→

in OOP →:




Customer has an address.

# Common type of OOP Relationships :→

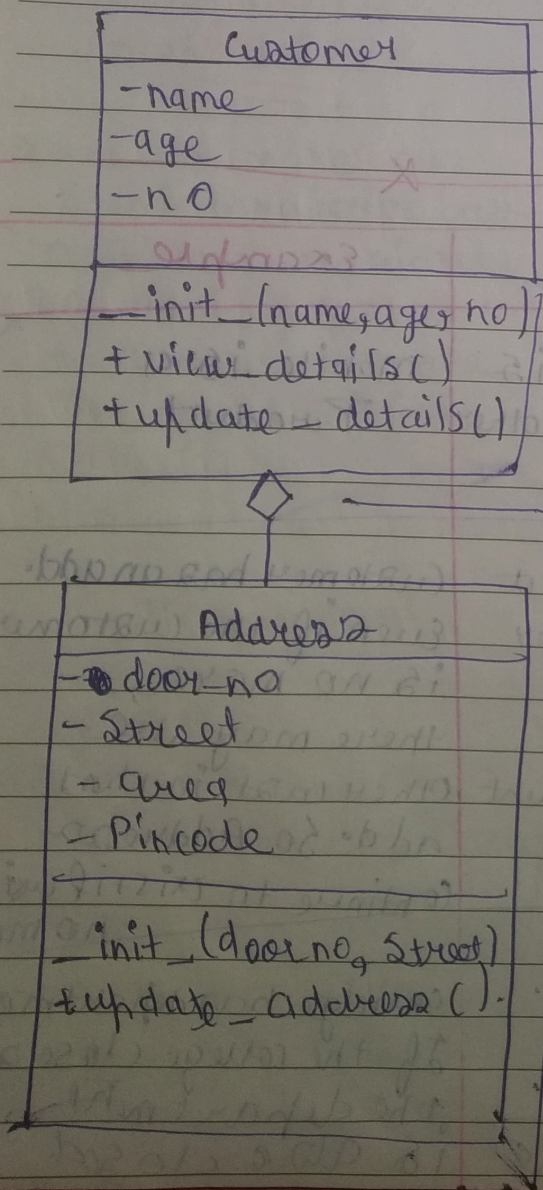
| Relationship | Description | Example |
|---|---|---|
| Inheritance | when one obj is a type of another obj | Mobile is a Product |
| Aggregation | when one object owns another object, but they both have independent life cycle | Customer has an add. Even if the customer is no more, there may be other cust in that add. So Address continue to exist if cus is no more |
| Composition | — | College has department. If the college closes the department is also closed |

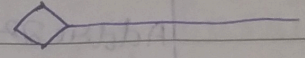
# Aggregation →:

→ If class A owns class B, then class is said to aggregate class B.

→ This is known as "has-A" relationship

For ex → In our shopping aph, a customer has an address.



→ In this diagram aggregation is represented by a line connecting the classes & a diamond symbol in the class which aggregates the other class.

→ For giving full access

# Need of Inheritance:->

-> Understanding with an egs ->

| SmartPhone |
|---|
| -brand |
| -price |
| -camera |
| -OS |
| -RAM |
| -battery |
| __init__(all ) |
| +buy() |
| +return() |
| +insure() |

Common Attributes

Common behaviour

| Feature Phone |
|---|
| -brand |
| -price |
| -camera |
| -type |
| __init__(brand, price, camera, type) |
| +buy() |
| +return() |

# Inheritance ->: Since both the classes are of type phone, we can create a Phone class with common attributes & methods & making these two classes inherit those attri. & behav

| Phone |
|---|
| -brand |
| -price |
| -camera |
| __init__( ) |
| +buy() |
| +return() |

| Feature Phone |
|---|
| -type |
| __init__(all ) |
| +exchange() |

| SmartPhone |
|---|
| -OS |
| -RAM |
| -battery |
| __init__( ) |
| +insure() |

→ When a class inherits from another class, & those classes are said to have an inheritance relationship.

→ The class which is inheriting is called child / sub / derived class.

→ & the class which is getting inherited is called Parent / Super / base class.

→ Inheritance is also called "is-a" relationship.

# Advantages of inheritance →:

→ We can keep common properties in a single place. Then any changes needs to be made need not repeated.

→ Inheritance encourages code reuses thus saving us time.

→ If we want to add a new type of phone later on, we can simply inherit the Phone class instead of writing from scratch.

** From a code perspective, a child class inherits:

→ constructor
→ private Attributes
→ Non private methods

→ Unlike other languages, private variable gets inherited in Python.

AK

22.

→ Sometimes a child may not want to use what it has inherited from the Parent. The same holds true for OOP as well.

→ If the child class doesn't want to use a method inherited from the Parent class then it may create its own method with the same name

→ When the child has a method with the same name of that Parent, it is said to override the Parent method

→ This is called method overriding

→ Also called as Polymorphism.