Ch 11 - Inheritance in Python

Inheritance is a method to create a new class from an existing class. The derived class will have all the attributes/function of the parent class. New methods can be added here.

Syntax :-

```
class Watch:
    # attributes/functions

class Rolex(Watch):
    # new attributes/functions
```

- All the elements of Watch class will be present in Rolex class.

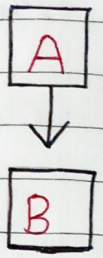
- Types of inheritance

i. Single inheritance
ii. Multiple inheritance
iii. Multilevel inheritance
iv. Hierarchical inheritance
v. Hybrid inheritance

* Inheritance plays a major role in reusability of code or for implementing DRY principle

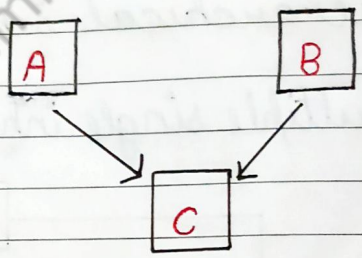
* Overwriting can also be done

- Single inheritance




```
class House: → Base class
    existence = True
    def enterHouse (self):
        print ("Entered house")

class Owner (House): → Derived class
    age = 45
```

- Multiple inheritance

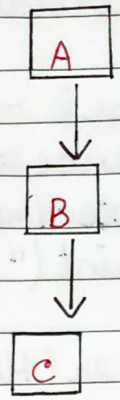



```
class Founder1:
    equity = "30%"

class Founder2:
    equity = "50%"

class Company (Founder1, Founder2): → Derived class
    valuation = "$ 100M"
```

Here, there is a variable called equity which is present in both classes.

In the Company class, value of equity variable will be set to "30%" because the class assigned before will get more priority

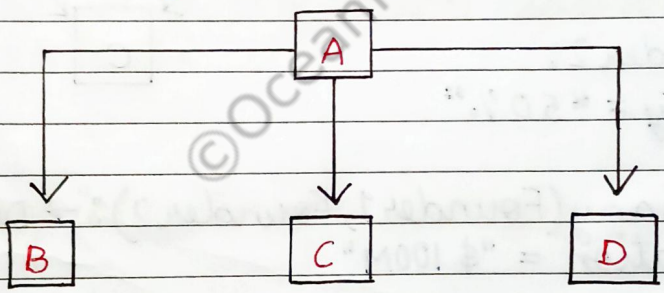
- Multilevel inheritance




```
class A :
    age = 27

class B(A):
    city = "Delhi"

class C(B)
    nationality = "Indian"
```

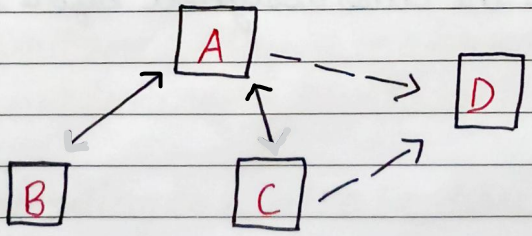
- Hierarchical inheritance

Multiple single inheritance = Hierarchical inheritance



- Hybrid inheritance

Multiple multiple inheritance = Hybrid instance



super() method

This method is used to access the methods of parent class into derived class.

```
class House:
    def enterHouse(self):
        print("Entered house")
    def exist(self):
        print("House Exists")

class Owner(House):
    def exist(self):
        super().exist()
        print("I exist")

A = Owner()
A.exist()
```

→ This first executes the exist() function of parent's class. Then it will execute it's own function

- @classmethod → Used to change class attribute
  - ↳ or create

```
class --:
    age = 32

    @classmethod
    def newAge(cls, new):
        cls.age = new
```