Class Reusability
An object oriented application is not developed by defining everything in one class.
Object oriented application is a collection of classes. The content of one class can be used inside another class in different ways

1. Composition (Has-A)
2. Aggregation (Use-A)
3. Inheritance (IS-A)

Composition
Creating an object of one class inside another class is called composition.

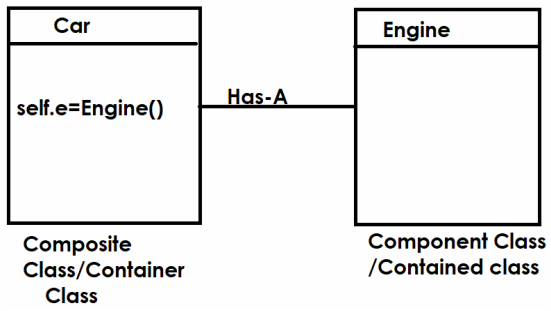
Composition in Python is a design principle where classes are built by combining objects of other classes, creating a "has-a" relationship. This approach promotes code reusability and flexibility

Composition is having two classes
1. Composite class
2. Component class

A class contains objects of other classes as members.

The composite class is responsible for creating and managing the lifecycle of its component objects.

Components can be reused in different composite classes, reducing redundancy



Example:
```
class Engine:
    def start(self):
```

```
        print("Engine Start...")
    def stop(self):
        print("Engine Stop...")

class Car:
    def __init__(self):
        self.e=Engine()
    def car_start(self):
        self.e.start()
    def car_stop(self):
        self.e.stop()

car1=Car()
car1.car_start()
car1.car_stop()
```

Output

```
Engine Start...
Engine Stop...
```

Component objects are destroyed automatically once composite class object is deleted/destroyed

Example:

```
class Address: # Component Class
    def __init__(self):
        self.__street=None
        self.__city=None
    def read_address(self):
        self.__street=input("Street :")
        self.__city=input("City :")
    def print_address(self):
        print(f"'Street {self.__street}
City {self.__city}'")

class Person: # Composite Class
    def __init__(self):
        self.__name=None
        self.__add=Address()
    def read_person(self):
```

```
        self.__name=input("Name :")
        self.__add.read_address()
    def print_person(self):
        print(f'Name {self.__name}')
        self.__add.print_address()

person1=Person()
person1.read_person()
person1.print_person()
```

Output

```
Name :Naresh
Street :Ameerpet
City :Hyderabad
Name Naresh
Street Ameerpet
City Hyderabad
```

Aggregation

Aggregation is a special type of composition.
In aggregation component class object is created outside the composite class and send/inject to composite class object using methods.
In aggregation component class exists independent of composite class. If composite class object is destroyed still component class object is exists.

Example:

```
class AirtelSim:
    def connect(self):
        print("connected to Airtel network")

class Mobile:
    def __init__(self,s):
        self.__sim=s
        self.__sim.connect()



sim1=AirtelSim()
iphone1=Mobile(sim1)
```

Output
connected to Airtel network

Example:

```
class Dept:
    def __init__(self,d,dn):
        self.__deptno=d
        self.__dname=dn
    def get_deptno(self):
        return self.__deptno
    def get_dname(self):
        return self.__dname

class Employee:
    def __init__(self,n,s,d):
        self.__name=n
        self.__salary=s
        self.__dept=d
    def print_employee(self):
        print(f"'EmployeeName {self.__name}
EmployeeSalary {self.__salary}
EmployeeDeptno {self.__dept.get_deptno()}
EmployeeDeptName {self.__dept.get_dname()}"')


dept1=Dept(10,"HR")
dept2=Dept(20,"SALES")
emp1=Employee("naresh",50000,dept1)
emp2=Employee("suresh",65000,dept1)
emp3=Employee("kishore",54000,dept2)
emp1.print_employee()
emp2.print_employee()
emp3.print_employee()
```

Output

```
EmployeeName naresh
EmployeeSalary 50000
EmployeeDeptno 10
EmployeeDeptName HR
EmployeeName suresh
```

EmployeeSalary 65000
EmployeeDeptno 10
EmployeeDeptName HR
EmployeeName kishore
EmployeeSalary 54000
EmployeeDeptno 20
EmployeeDeptName SALES

Inheritance (IS-A)

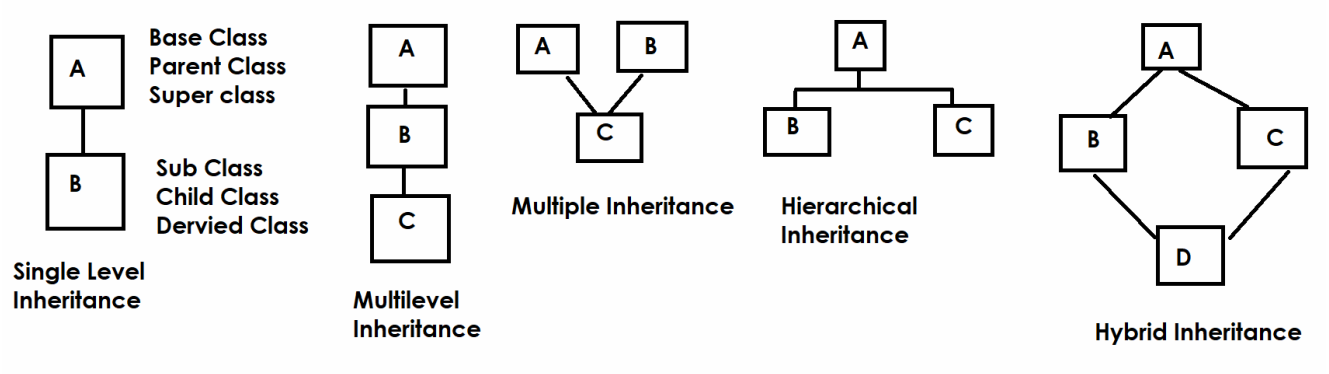
Inheritance is process of acquiring the properties and behavior of one class inside another class.
(OR) a class acquires the properties and behavior of another class is called inheritance.
Inheritance in Python is a mechanism that allows a new class (child class or subclass) to inherit attributes and methods from an existing class (parent class or superclass). This promotes code reusability, reduces redundancy, and enhances flexibility in code development
Inheritance is process of grouping all the class which share common properties and behavior

Advantage

1. **Reusability :** It allows to use the variables and methods of one class inside another class
2. **Extensibility :** it allows extending functionality of existing class without modifying

Types of inheritance

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



Syntax of creating subclass or child class or derived class

class derived-class-name(base-class-name,base-class-name,..):

variables

methods