Class Reusability
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) → Special type of composition
3. Inheritance (IS-A)

Composition
Composition is defined by the PART-OF relationship which means that one object IS PART-OF ANOTHER OBJECT, but Aggregation is defined by the HAS-A relationship which means that one object HAS-A RELATION with another object.

Example:
```
class Engine: # Contained class
    def start(self):
        print("Engine Start")
    def stop(self):
        print("Engine Stop")

class Car:  # Container class
    def __init__(self):
        self.e=Engine()
    def start(self):
        self.e.start()
    def stop(self):
        self.e.stop()


car1=Car()
car1.start()
car1.stop()
```

Output:
Engine Start
Engine Stop
Example:

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

class Person:
    def __init__(self):
        self.__name=None
        self.__add=Address()
    def read_person(self):
        self.__name=input("Name ")
        self.__add.read()
    def print_person(self):
        print(f'Name {self.__name}')
        self.__add.print_address()


p1=Person()
p1.read_person()
p1.print_person()
```

Output:

```
Name kishore
Street ameerpet
City hyd
Name kishore
Street ameerpet
City hyd
```

What’s the difference between Aggregation and Composition?

There are two sub-types of Association relationships — Aggregation and Composition. What's the difference between these two?

Composition

Composition implies that the contained class *cannot* exist independently of the container. If the container is destroyed, the child is also destroyed.

Take for example a Page and a Book. The Page cannot exist without the Book, because the book is *composed of* Pages. If the Book is destroyed, the Page is also destroyed.

In code, this usually refers to the child instance being created inside the container class:

```
class Book:
   def __init__(self):
      page1 = Page('This is content for page 1')
      page2 = Page('This is content for page 2')
      self.pages = [page1, page2]

class Page:
   def __init__(self, content):
      self.content = contentbook = Book() # If I destroy this Book instance,
           # the Page instances are also destroyed
```

Aggregation

With an aggregation, the child *can* exist independently of the parent. So thinking of a Car and an Engine, the Engine doesn't need to be destroyed when the Car is destroyed.

```
class Car:
   def __init__(self, engine):
      self.engine = engine

class Engine:
   def __init__(self):
      passengine = Engine()
```

```
car = Car(engine) # If I destroy this Car instance,
                  # the Engine instance still exists
```

Example:

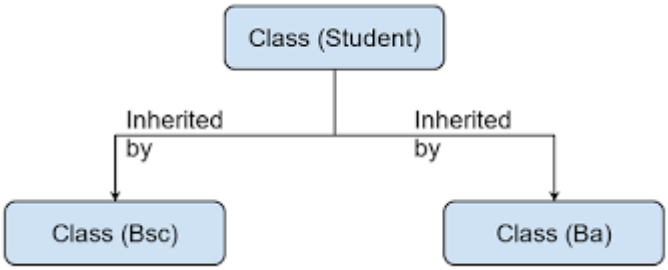
```
class Sim:
    def connect(self):
        print("Connect to Network")

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


jio1=Sim()
airtel1=Sim()
apple1=Mobile(jio1)
apple2=Mobile(airtel1)
```

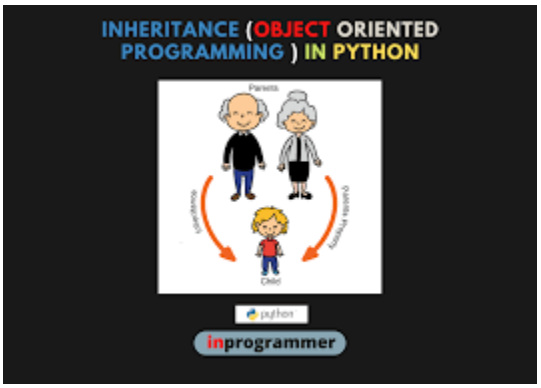
Output:

```
Connect to Network
Connect to Network
```

Inheritance

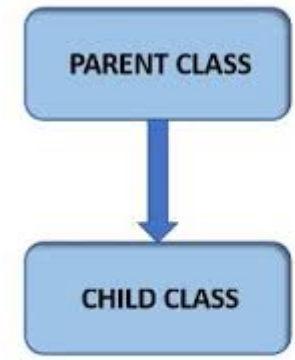
Inheritance is a process of acquiring the properties and behavior of one class inside another class.
Inheritance is a process of grouping all the classes which share common properties and behavior.
Inheritance allows creating new class or data type based on existing class or data type.





Advantage of inheritance,

1. Reusability: The attributes and methods of one class can be used inside another class.
2. Easy to understand
3. Extensibility



Based on the reusability of classes

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