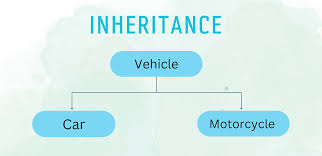
**Inheritance**: Inheritance is passing down behaviours and properties from parent class to child class; classes can inherit characteristics and behaviors from other classes. This makes it easier to create new things that are similar to existing ones.



class Animal:

def \_\_init\_\_(self, name):

self.name = name

def sound(self):

return "Some sound"

class Dog(Animal):

def sound(self):

return "Bark"

def cry(self):

return "Woooooooooo"

class TrainedDog(Dog):

def sound(self,command):

if (command == "Speak"):

return "Bark"

else:

return "..."

def cry(self):

return "…"

# Using Inheritance

dog = Dog("Buddy")

print(dog.name)

print(dog.sound()) # Dog inherits the properties and methods of Animal

trainedDog = TrainedDog("Rocky")

print(trainedDog.name)

print(trainedDog.sound("No command"))

print(trainedDog.sound(""))

print(trainedDog.sound("Speak")) # Dog inherits the properties and methods of Animal

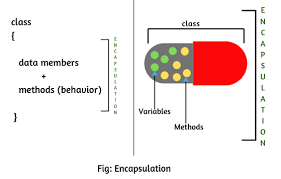
**Polymorphism –** Changing behaviours based on context. A derived class object will behave differently in case the behaviour is altered.

list = [dog,trainedDog]

for dogObj in list:

print("Polymorphism - "+ type(dogObj).\_\_name\_\_ + " cries " + dogObj.cry())

**Encapsulation -** It is about keeping the important stuff hidden and share what is necessary. E.g. Access modifiers – public, private, internal etc.



class BankAccount:

def \_\_init\_\_(self, owner, balance=0):

self.owner = owner

self.\_\_balance = balance # Private variable, hidden from outside

def deposit(self, amount):

self.\_\_balance += amount

def withdraw(self, amount):

if amount <= self.\_\_balance:

self.\_\_balance -= amount

else:

print("Insufficient balance.")

def get\_balance(self):

return self.\_\_balance

# Creating a bank account

account = BankAccount("Alice")

account.deposit(100)

print(account.get\_balance()) # Can only access balance through method

# account.\_\_balance = 500 # This will give an error because \_\_balance is private

**Abstraction** : Hiding complexity – Identifying common things. We use cellphones to talk, play, use apps. We do not care about how it is working. It helps make the code simpler and easier to work with by only exposing the necessary functionality while hiding the complex parts.

class Animal:

def \_\_init\_\_(self, name):

self.name = name

def sound(self):

return "Some sound"