Create Student Class that takes name & marks of 3 subjects as list as argument in contructor.
then create a method to print the average

```
class Student:
    def __init__(self, name, marks):
        self.name = name
        self.marks = marks

    def avg():                    [1,2,3]
        sum = 0
        for ele in self.marks:
            sum = sum + ele
        print("Hi", self.name, "your avg
              is", sum/3)

s1 = Student("ABC", [99,98,97])
s1.avg()
```

Static method

```
class student:
    # Constructor
    @staticmethod  # decorator
    def welcome():
        ≡
```

OOPs

Abstraction Encapsulation Inheritence Polymorphism

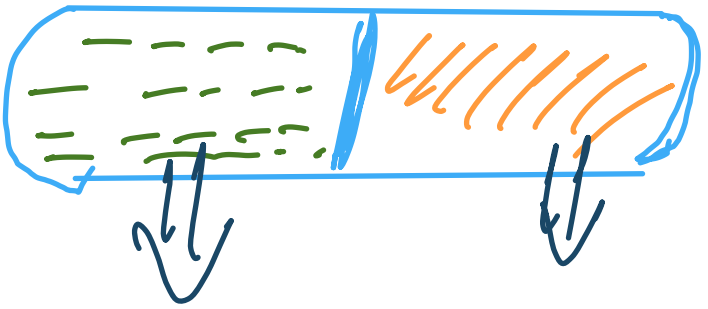
① Abstraction

↳ Detail Hiding

class student :

method : # only method signature

② Encapsulation



data function

Binding the data & function into a single unit

⇒ Data Hiding

```
class Bank:
    def __init__(self, acc, pass):
        self.acc = acc
        self.__pass = pass

    def reset_pass():
        print(self.__pass)
```

```
b1 = Bank("abc", 1234)
print(b1.acc)
print(b1.reset_pass())
```

```
class car:
    def __init__(self):
        self.acc = false
        self.brk = false
        self.clutch = false

    def start(self):
        self.acc = true
        self.clutch = true
        print("car started")

c = car()
c.start()  # car started
```

Q: Create a Bank account class & take account no & balance as argument in constructor. Create 3 methods for credit, debit & get balance

```
class BA:
    def init(self, Acc No, balance):
        ⟹
        →

    def credit(self, amount):
        balance = balance + amt

    def debit(self, amount)
        balance = balance - amt
```

def getbalance(self):
return balance

ankit = BA(123, 10000)
ankit.debit(1000)
ankit.credit(500)

Inheritence

Inheriting the prop. from parent class to the child class/sub class

```
class car:
    color = "Black"
    @staticmethod
    def start():
        print("car started")
    @staticmethod
    def stop():
        print("car stop")

class Maruti(car):
    def __init__(self, name):
        self.name = name
```

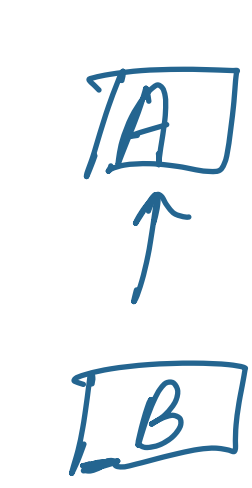
m1 = Marvel("Script")

m1.start()

m1.stop()

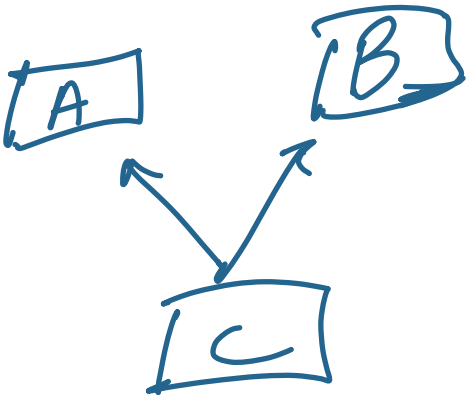
Types of Inheritence

1. Single Inheritence
2. multi-level Inheritence
3. multiple Inheritence
4. Hybrid / Heirarchical

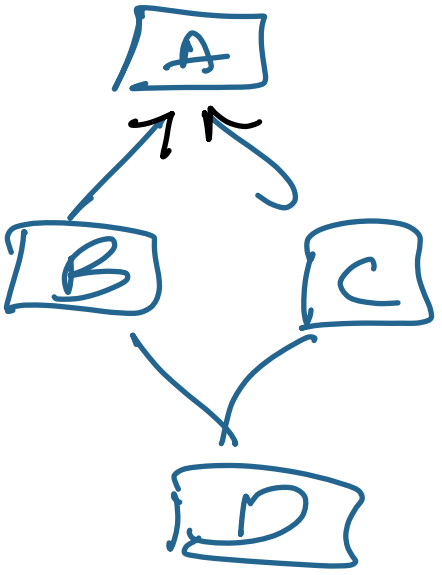


Single

multi level



multiple




```
class car :
    __init__(self, type)
        ==
    start():
        ==

class maruti(car):

    engineStop():
        ==

class swift(maruti) :

    End():
        ==
```

s = Swiggy()

s1.Encd()

s1.Engine()

s1.Software()

multiple

class A:

    varA = "in A"

class B:

    varB = "in B"

class C(A, B):

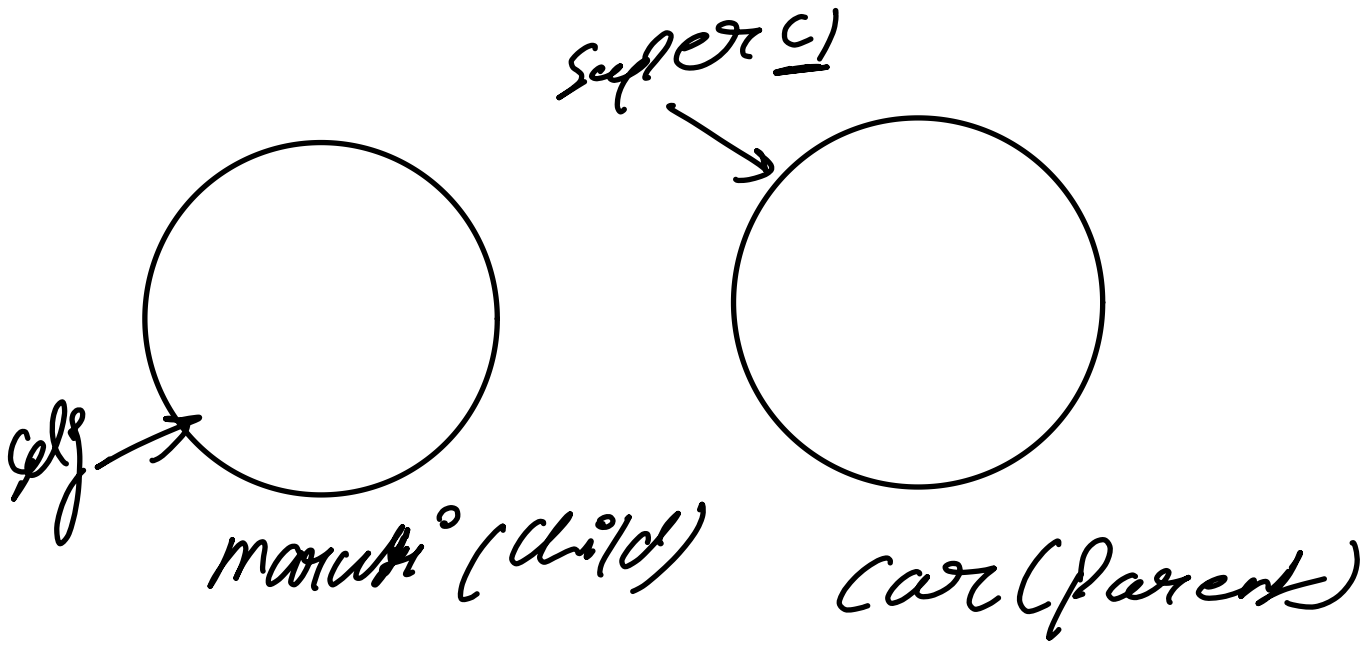
    varC = "in C"

C1 = (C)

p(C1.varA)

p(C1.varB)

p(C1.varC)



super

super()

Super keyword:

```
class Car:
    def __init__(self, type):
        self.type = type
    @staticmethod
    def start():
        print("car Started")

    @staticmethod
    def stop():
        print("Car Stopped")

class Maruti(Car):
    def __init__(self,name, type):
        self.name = name
        super().__init__(type)


s1 = Maruti("Swift", "Diesel")
print(s1.name)
print(s1.type)
s1.start()
```

@classmethod

```
class Car:  2 usages
    name="Arun"

    def __init__(self, name):
        self.name=name

c1 = Car("Rahul")
print(c1.name)  #rahul
print(Car.name) #arun
```

way1

```
class Car:  3 usages
    name="Arun"

    def __init__(self, name):
        Car.name=name

c1 = Car("Rahul")
print(c1.name)  #rahul
print(Car.name) #Rahul
```

way 2

```
class Car:  2 usages
    name="Arun"

    def __init__(self, name):
        self.__class__.name=name

c1 = Car("Rahul")
print(c1.name)  #rahul
print(Car.name) #Rahul
```

Polymorphism

Poly + morphism

Many forms/structure

Compile time Polymorphism/
Early Binding

↓

overloading

Runtime Polymorphism/
late Binding

↓

overriding

Overloading

① def show(self): ⓪

② def show(self, age, name) ②

③ def show(self, name, age) ②

Condⁿ

① Within same class
② Same function name
③ diff
- no. of parameter
- Type of parameter
- Sequence of parameter

s1.show(20, "Arun")

Overriding

```
class maruti:
    start():
        print("start with key")

class swift(maruti):
    start():
        print("start with button")

s1=swift():
    s1.start()
```

① within different class

② Same function name

③ Same

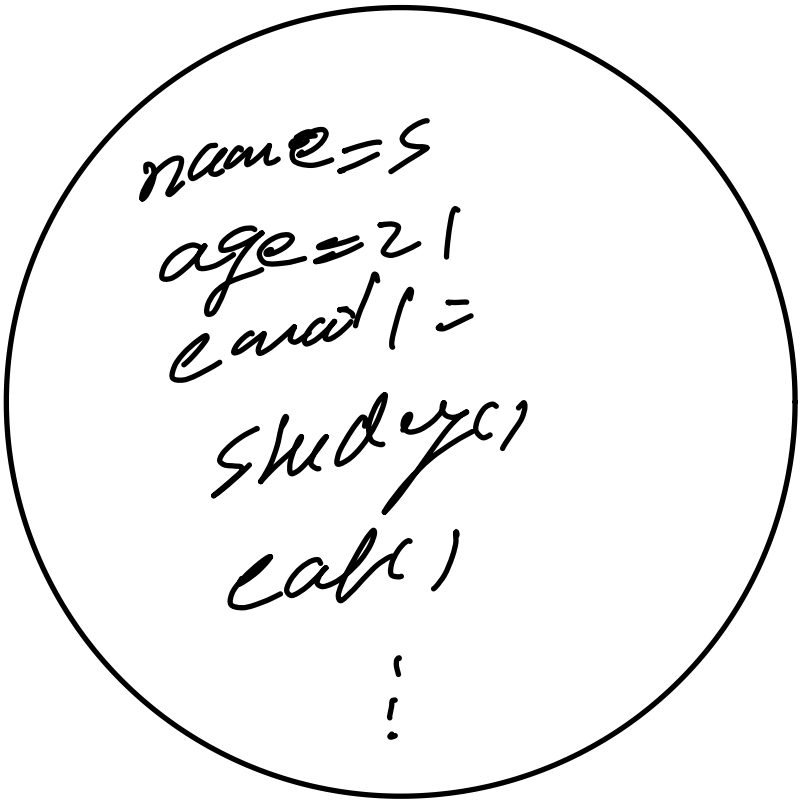
- no. of parameter
- Type of parameter
- sequence of parameter

Oops

```
class Student:
    name = "sumit"
    age = 21
    email = "  "

def study():
    ____
    ____
    ____
    ____
    ____

def eat():
    ____
    ____
    ____
    ____
```



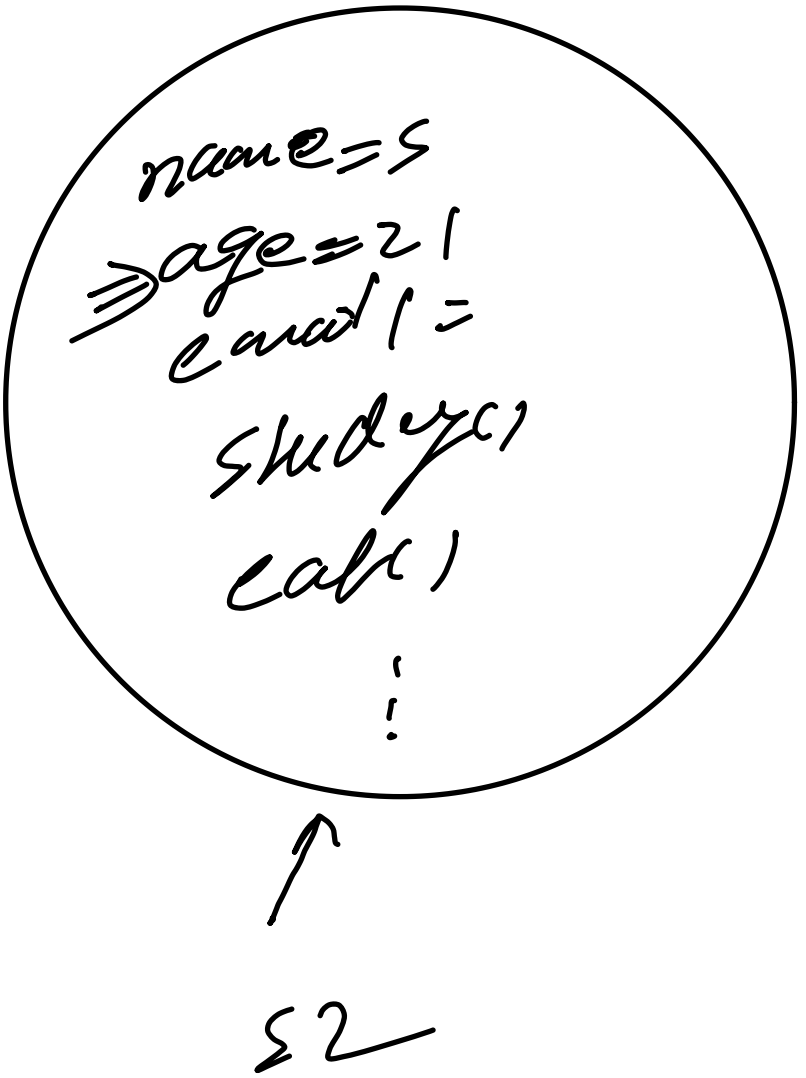
s1

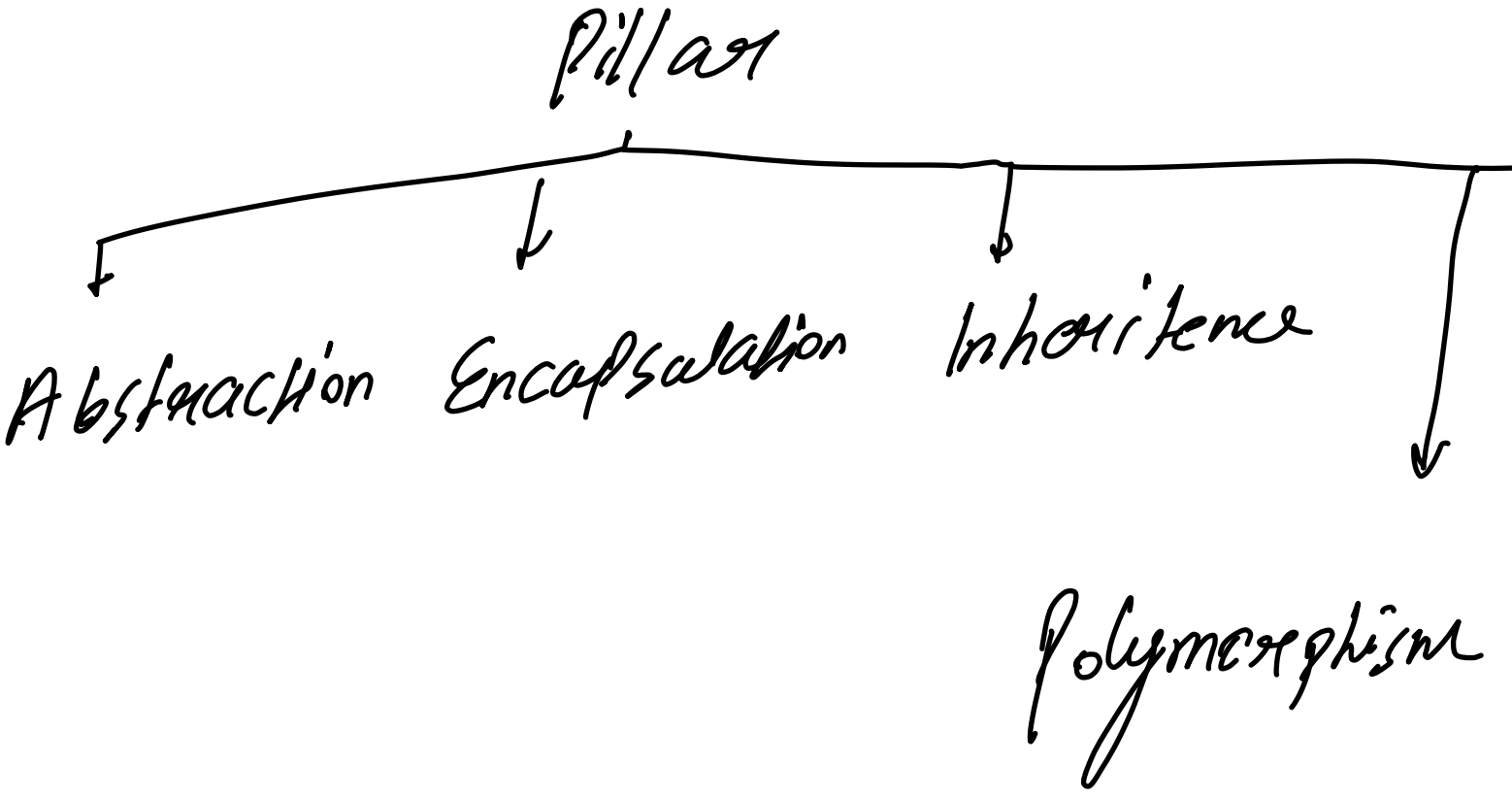
```
s1 = Student()
print(s1.name)
```

⇒ s1.study() ⇒

s1.eat()

```
s2 = Student()
print(s2.age)
```





Abstraction → Detail

start():

stop():

Encapsulation ⇒ Data hiding

Data junction

Inheritance

Parent()

↑

Child()

class parent :

data

function

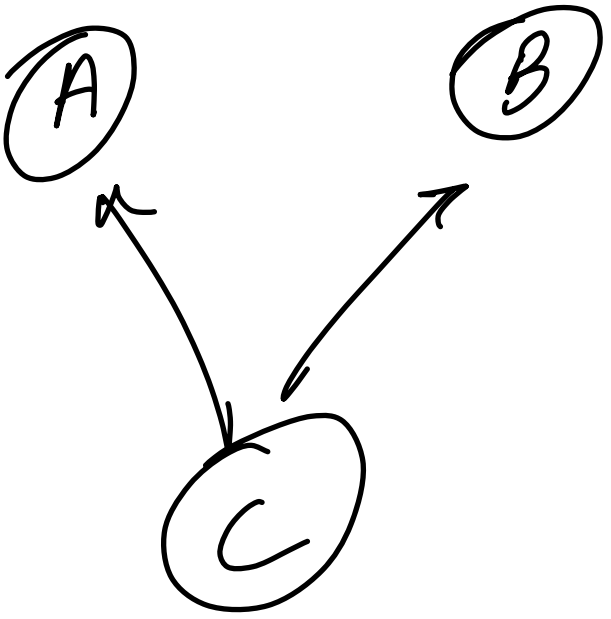
class child(parent):

Types

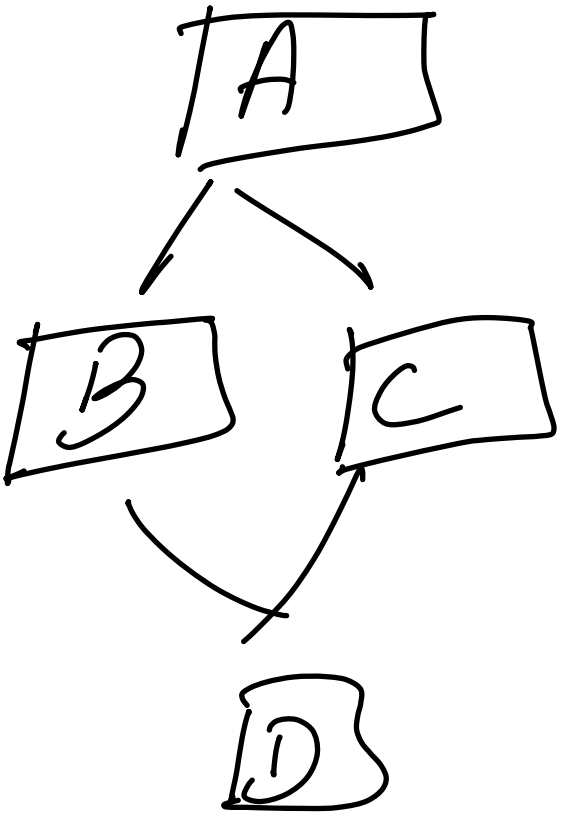
① Single

② multilevel

③ multiple



④



Polymorphism

CTP

↓

Overloading

1. Within Same Class
2. Same function name
3. diff
   - → no. of parameter
   - → Type of parameter
   - → Sequence of parameter

RTP

↓

Overriding

1. within diff class
2. Same function name
3. Same

```
sound(String name)

sound(String email)

sl.sound("Arun")

add(num1, num2)
    return num1 + num2

add(num1, num2, num3)
    return 1+2+3
```