

## 1) What is class?

In object-oriented programming (OOP), a class is a blueprint or template for creating objects. Objects are instances of a class.

## 2) What is a constructor?

In object-oriented programming (OOP), a constructor is a special method or function that is automatically called when an object is created. Its primary purpose is to initialize the object's attributes or properties and set up the initial state of the object.

Key characteristics of constructors:

**Initialization:** The main task of a constructor is to initialise the object's attributes or properties. It sets the initial values for the object's state.

**No Return Type:** Unlike regular methods, constructors don't have an explicit return type. They implicitly return an instance of the class.

## 3) What is oop ?

কম্পিউটার প্রোগ্রামিং এর একটি ধরন হচ্ছে অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং। প্রোগ্রামিং এর জন্য এটি একটি শক্তিশালী হাতিয়ার। খুব সাধারণ অর্থে বলতে গেলে, অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং কাজ করে কোড এবং তার সাথে সংশ্লিষ্ট ডাটা নিয়ে।

অন্য কথায় বলা যায় এখানে আমাদের কাছে বেশ কিছু ডাটা বা অবজেক্ট থাকবে এবং আমরা সেই ডাটা বা অবজেক্টগুলোকে নিয়ে আর সুন্দরভাবে কাজ করার জন্য বেশ কিছু ফাংশন বা প্রোগ্রাম লিখব।

[আপনি চাইলে উত্তরটিকে আরো সুন্দরভাবে অর্গানাইজ করতে পারেন ]

## 4) What is inheritance ?

In object-oriented programming (OOP), inheritance is a mechanism that allows one class to inherit the properties and behaviours of another class. The class that is being inherited from is called the "parent" or "base" class, and the class that inherits from it is called the "child" or "derived" class.

The main idea behind inheritance is to promote code reuse and create a relationship between classes that reflects an "is-a" relationship. This means that a derived class is a specialised version of the base class, and it inherits the attributes and methods of the base class while being able to add or override them as needed.

```
class Animal:

    def __init__(self, name):

        self.name = name

    def make_sound(self):

        pass # Placeholder method


class Dog(Animal):

    def make_sound(self):

        return "Woof!"


class Cat(Animal):

    def make_sound(self):

        return "Meow!"
```

```
# Creating instances of derived classes

dog = Dog("Buddy")

cat = Cat("Whiskers")


# Using inherited method

print(dog.make_sound())    # Output: Woof!

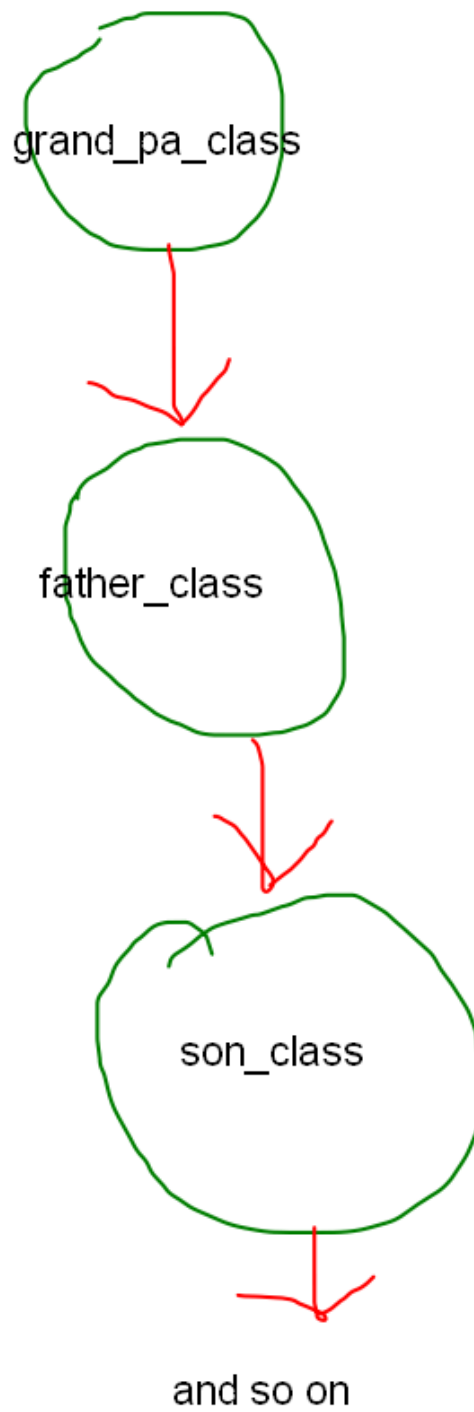
print(cat.make_sound())    # Output: Meow!
```

## 5) What is Multilevel inheritance ?

multi level inheritance বলতে সাধারণত আমরা বুঝে থাকি এখানে একটি লেভেল জিরো এর প্যারেন ক্লাস থাকবে, তারপর এর থেকে আরেকটি চাইল্ড ক্লাস বের হবে ধরি এটি হলো লেভেল ১ এর। তারপর এই লেভেল এক এর ক্লাস কে ইনহেরিট করে আরেকটি child ক্লাস বের হবে এর নাম দেওয়া যাক লেভেল টু এর চাইল্ড ক্লাস।

কিরকম একেও ইনহেরিট করে আরেক নতুন ক্লাস বের হবে।

এইভাবে যে পেরেন্ট চাইল্ডের একটা সিকোয়েন্স তৈরি হবে একে সাধারণত বলা যেতে পারে মাল্টিলেভেল ইনহেরিটেন্স।



I am going to write a example code bellow

```

class Dada:

    gramer_bari_jomi = "20 Biga"

class Father(Dada):

    dhake_te_sompotti = "10 tata building"

class my_fried_morir(Father):#Vagabond person হওয়া
সঙ্গেও ও বাপ দাদার সম্পত্তি ভোগ করতে পারবে। এটাই মাল্টিলেভেল
ইনহেরিটেন্স

    def monir_er_smopotti(self):return f"land =
{self.gramer_bari_jomi} house =
{self.dhake_te_sompotti}"

Dihan_Monir = my_fried_morir()

print(Dihan_Monir.monir_er_smopotti())

```

## 6) What is Multiple inheritance ?

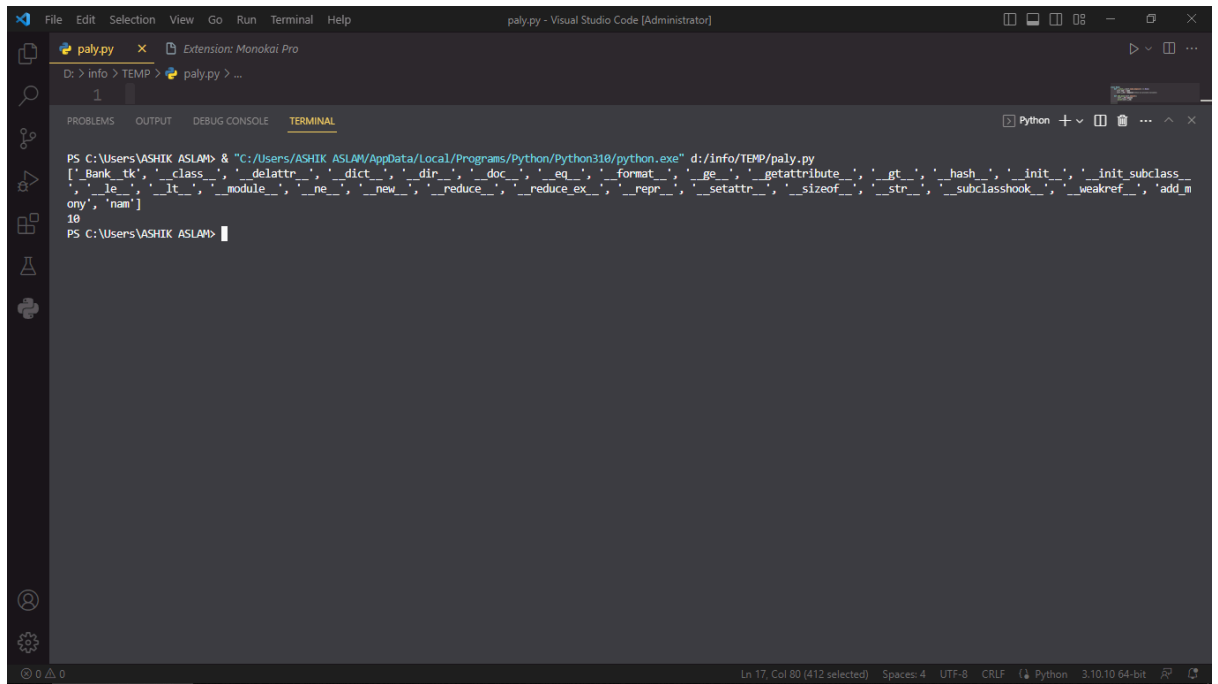
এখানে চাইল্ড ক্লাস টি একের অধিক parent ক্লাস কে ইনহেরিট করে থাকে

## 7) How to print private variables outside of the class?

```
class Bank:
    def __init__(self, name, deposit) -> None:
        self.nam = name
        self.__tk = deposit #this is private variable

    def add_mony(self, amount):
        self.__tk+=amount
        print(self.__tk)

aslam = Bank('aslam',10)
# aslam.add_mony(10)
print(dir(aslam))
print(aslam._Bank__tk) # here we print the private variable outside of the class
```



```
File Edit Selection View Go Run Terminal Help
paly.py - Visual Studio Code [Administrator]
paly.py x Extension: Monokai Pro
D: > info > TEMP > paly.py > ...
1
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Python + -
PS C:\Users\ASHIK ASLAM> & "C:/Users/ASHIK ASLAM/AppData/Local/Programs/Python/Python310/python.exe" d:/info/TEMP/paly.py
['Bank', 'tk', 'class', 'delattr', 'dict', 'dir', 'doc', 'eq', 'format', 'ge', 'getattr', 'getattribute', 'gt', 'hash', 'init', 'init_subclass',
'_le', '_lt', '_module', '_ne', '_new', '_reduce', '_reduce_ex', '_repr', '_setattr', '_sizeof', '_str', '_subclasshook', '_weakref', 'add_m
ony', 'nam']
10
PS C:\Users\ASHIK ASLAM>
```

## "abstract class" or an "abstract method."

এক্ষেত্রে আমরা প্যারেন্ট ক্লাস তৈরি করার সময়ই বলে দেই কোন চাইল্ড ক্লাস যদি একে ইনহেরিট করতে চায় তাহলে প্যারেন্ট ক্লাসের মধ্যে অবস্থিত কিছু নির্দিষ্ট ফাংশন কে চাইল্ড ক্লাসের ভিতরে আবার নতুন করে ওই child ক্লাসের উপর ভিত্তি করে তৈরি করতে হবে  
অন্যথায় আমরা প্যারেন্ট ক্লাস কে ইনহেরিট করতে পারবো না  
অর্থাৎ এখানে চাইল্ড ক্লাস কে একরকম বাধ্য করা হয় ফাংশনগুলোকে নতুন করে ইমপ্লিমেন্ট করার জন্য

```
from abc import ABC, abstractmethod

# abs = abstract base class

class animal(ABC):
    @abstractmethod # it will enforces all
    child class to have eat() function
    def eat(self):
```

```

        pass

    def move(self):
        pass

class monkey(animal):
    def __init__(self, name) -> None:
        self.Name = name
        super().__init__()
    def eat(self):
        print(f"ola im eating bannana!!")

monkey1 = monkey('ben')
monkey1.eat()

```

## Abstract Classes Vs Interfaces

abstract class এর ক্ষেত্রে python এ মালটি লেভেল ইনহেরিটেন্স সাপোর্ট করলেও অন্যান্য প্রোগ্রামিং ল্যাঙ্গুয়েজ গুলোতে মাটি লেভেল ইনহেরিটেন্স কাজ করে না  
কিন্তু ইন্টারফেসের ক্ষেত্রে সব ক্ষেত্রে multi level inheritance কাজ করে

ইন্টারফেস এবং abstract দুটোর ক্ষেত্রে মূলনীতি মূলত একি তা হল ফাংশন বা মেথড তৈরিতে বাধ্য করা।

php language অনুযায়ীকোনো ইন্টারফেস ক্লাস কে যদি ইনহেরিট করতে হয় তাহলে চাইল্ড ক্লাসকে প্যারেন্ট ক্লাসের সবগুলো ফাংশন কে পুনরায় নিজের মধ্যে ইমপ্লিমেন্ট করতে হবে।

কিন্তু abstract ক্লাসের ক্ষেত্রে শুধুমাত্র ওই ফাংশন গুলো কি ইমপ্লিমেন্ট করতে হবে যেগুলোকে আমরা abstract মেথড হিসেবে রাখবো



# Polymorphism

There are two main types of polymorphism:

- 1) compile-time (static) polymorphism or method overloading
- 2) runtime (dynamic) polymorphism. Or method overriding

[Video link](#)

## operator overloading

operator overloading হল এমন একটি প্রক্রিয়া যেখানে আমরা কোন অপারেটরের behaviour কে আমাদের ক্লাসের মধ্যে এমন ভাবে কাস্টমার করতে পারি যাতে করে সেই অপারেটরটি আমাদের ক্লাসের এড প্রতিটি ইনস্ট্যান্সের জন্য আমরা যেভাবে চাই ওই ভাবে behave করে

### Dunder or magic methods in Python

```
class person:
    def __init__(self, name, age) -> None:
        self.name = name
        self.age = age

    def __add__(self, other): #   ekhane + kaj ke
amar modifi koresi
```

```
        return(self.age + other.age)

Ashik_Aslam = person('ashikaslam',19)
mr_robot = person('mr_robot',26)

print(Ashik_Aslam + mr_robot)
```

যেমন আমরা এখানে দুটো পার্সন কে সরাসরি যোগ করতে পারি না কিন্তু + এর behaviour কে আমরা ক্লাসের ভিতর পরিবর্তন করে তারপর একে ব্যবহার করেছি।

## Getter and setter

```
class person:
    def __init__(self,name,age) -> None:
        self.name = name
        self.age = age
        self.__tk = 100
    @property
```

```
def tk(self):# this only getter
    return self.__tk
@tk.setter
def set_tk(self,amount):# and this is setter
    self.__tk+=amount

Ashik_Aslam = person('ashikaslam',19)
print(Ashik_Aslam.tk)
Ashik_Aslam.set_tk = 10

print(Ashik_Aslam.tk)
```

## Inheritance vs Composition

Inheritance means is a relationship whereas composition means has a relationship

## C++ code for oop

```
#include <bits/stdc++.h>
using namespace std;

class father__class
{
public:
    int age = 37;
    father__class()
    {

        age = 39;
    }
    void say_some()
    {
        cout<<"hello world" <<endl;
    }

};

int main()
{
    father__class theobj;
    theobj.say_some();
}
```

```
    return 0;  
}
```

## Friend class in c++

>>