

ANSWER TO THE QUESTION NUMBER 1

>

a)

static method হলো এমন একটি মেথড এটি ক্লাসের ভিতরেই থাকে কিন্তু এখানে argument হিসাবে self দিতে হয় না। কারণ ক্লাস ইনস্ট্যান্সের সাথে এর কোন সম্পর্ক থাকে না।

Example:-

```
class Person:

    country = "bangladesh"

    def __init__(self,name,age) -> None:

        self.name = name

        self.age = age

    @staticmethod

    def addtwoNum(a,b):return a+b # do not to
take argument for class refarance
```

অপরদিকে

class method হলো এমন একটি মেথড এটি ক্লাসের ভিতরেই থাকে কিন্তু এখানে argument হিসাবে self বা অন্য কিছু দিতে হয় ।

```
class Person:

    country = "bangladesh"
```

```
def __init__(self,name,age) -> None:

    self.name = name

    self.age = age

    @classmethod

    def

change_country(cls,newcountry):cls.country=newcou
untry
```

b)

static মেথডকে আমরা সাধারণত ইউটিলিটি মেথর হিসাবে ব্যবহার করি।

অপরদিকে

Class method আমরা তখনই ব্যবহার করব যখন আমাদের ক্লাস state নিয়ে কাজ করার দরকার পড়বে

c)

কোন মেথড কে ইন্সট্যান্স মেথড হিসেবে ব্যবহার করতে হলে তার উপরে

```
@staticmethod
```

দিতে হয়।

অপরদিকে

কোন মেথড কে classmethod মেথড হিসেবে ব্যবহার করতে হলে তার উপরে

```
@classmethod
```

দিতে হয়।

d)

static method ক্লাসের মধ্যকার কোন কিছুকে access করতে পারেনা এবং সেগুলোকে মডিফাই ও করতে পারে না

অপরদিকে

static method ক্লাসের মধ্যকার কোন কিছুকে access করতে পারে এবং সেগুলোকে মডিফাই ও করতে পারে

ANSWER TO THE QUESTION NUMBER 2

>

একই ফাংশনের ভিন্ন ভিন্ন সময় বা ক্ষেত্রে ভিন্ন ভিন্ন রূপ ধারণ বা কাজ করার প্রক্রিয়াকে বলা হয় polymorphism।

1)method Over Writing

2)method overloading

এইগুলো পলিমরফিজমের এক একটি ধরন।

নিচে পলিমরফিজম এর উদাহরণ দেওয়া হল

```
class Person:#parent class

    def info(self):print("im a human")

class Student(Person):#child class

    def info(self):print("im a student")

class Cowboy(Person):#child class

    def info(self):print("im a cowboy")

Personx = Person()

Persony = Student()

Personz = Cowboy()

Personx.info()
```

```
Persony.info()  
  
Personz.info()
```

Output

im a human

im a student

im a cowboy

এখানে আমরা প্যারেন্ট ক্লাসের info

ফাংশন টিকে নিজেদের প্রয়োজন অনুযায়ী চাইল্ড ক্লাসে পরিবর্তন করেছি। যার কারণে ভিন্ন ভিন্ন ক্লাস থেকে একে কল করলে এখন ভিন্ন ভিন্ন আউটপুট দেখাচ্ছে।

ANSWER TO THE QUESTION NUMBER 3

>

```
import math
```

```
class Poor_math:

    def __init__(self,a,b,c) -> None:

        self.A = a

        self.B=b

        self.C = c

    def sUm(self):return self.A + self.B+self.C

    def facto(self):return math.factorial(self.B)

Objectx = Poor_math(1,5,3)

print(Objectx.sUm())

print(Objectx.facto())
```

ANSWER TO THE QUESTION NUMBER 4

>

multi level inheritance বলতে সাধারণত আমরা বুঝে থাকি এখানে একটি লেভেল জিরো এর প্যারেন্স 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।

grand_pa_class

```
graph TD; A((grand_pa_class)) --> B((father_class)); B --> C((son_class)); C --> D[and so on];
```

father_class

son_class

and so on

I am going to write a example code bellow

```
class Dada:

    gramer_bari_jomi = "20 Biga"

class Father(Dada):

    dhake_te_sompotti = "10 tata buliding"

class my_fried_morir(Father):#Vagabond person হওয়া
সত্বেও 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())
```

ANSWER TO THE QUESTION NUMBER 5

>

যখন কোন outer function ভিতরে inner function তৈরি করা হয়। তখন ওই ইনার ফাংশনটি আউটার ফাংশনের সকল ভেরিয়েবলের এক্সেস পায়।

এবং এর সাহায্যে অনেক কাজকে খুব সহজে করা যায় যার ফলে প্রোগ্রামের readability and way of organisation অনেক সহজ হয়ে যায়।

ডিনার ফাংশন ব্যবহার করে ডেকারেটরও তৈরি করা যায়।

নিচে এর উদাহরণস্বরূপ একটি প্রোগ্রাম লেখা হলো।

```
import math

def timer(fun):

    def inner(n):

        print("time started>>")

        fun(n)

        print("time ended>>")

    return inner

@timer

def get_facto(n):

    print("facto is >> ",math.factorial(n))
```

```
get_facto(5)
```

ANSWER TO THE QUESTION NUMBER 6

>

```
n,m = map(int,input().split())

my_list = [0]*n

my_list = list(map(int, input().split()))

ans = [0]*(m+1)

for i in my_list:ans[i]+=1

for i in range(1,m+1):print(ans[i])
```

ANSWER TO THE QUESTION NUMBER 7

>

```
class Person:

    def __init__(self, name, age, height, weight)
-> None:

        self.name = name
```

```
        self.age = age

        self.height = height

        self.weight = weight


class Cricketer(Person):

    def __init__(self, name, age, height, weight)
-> None:

        super().__init__(name, age, height,
weight)

    def __lt__(self,others):

        return self.age<=others.age

    def __repr__(self) -> str:

        return self.name


Sakib = Cricketer('Sakib', 38, 68, 91)

Mushfiq = Cricketer('Mushfiq', 36, 55, 82)

Mustafiz = Cricketer('Mustafiz', 27, 69, 86)

Riyad = Cricketer('Riyad', 39, 72, 92)
```

```
mylist = [Sakib,Mushfiq ,Mustafiz,Riyad]
```

```
print(mylist)
```

```
mylist.sort()
```

```
print(mylist)
```

```
print("the youngest player is ",mylist[0]) # may  
be it is working like custom comparator like i do  
in c++
```