**1)What do you mean by decorators in python?**

**Ans:**In Python, decorators are a powerful and flexible way to modify or extend the behavior of functions or methods without directly altering their source code. Decorators are higher-order functions, which means they take another function as input and return a new function with some added functionality.

Example:

class student:  
 school = "myschool" # class/static variable  
 def \_\_init\_\_(self,m1,m2,m3): # instance method  
 self.m1=m1  
 self.m2=m2  
 self.m3=m3 # instance variable  
 def avg(self):  
 return (self.m1+self.m2+self.m3)/3  
 @classmethod # decoder  
 def getSchool(cls): # class method  
 return cls.school  
 @staticmethod # decoder  
 def info(): # static method  
 print("THIS IS STUDENT CLASS")  
s1=student(23,54,87)  
print(s1.avg())  
print(student.getSchool())  
student.info()

Decorators are commonly used for various purposes, such as logging, timing functions, authentication, and more. They provide a clean and modular way to extend the functionality of functions without modifying their core logic.

**2)What is the difference between @classmethod and @staticmethod?**

**Ans: @classmethod @staticmethod**

|  |  |
| --- | --- |
| ****Definition**** | Bound to the class | Bound to the class (like a regular function within the class) |
| ****First Argument**** | **cls** (class) | No specific first argument |
| ****Usage**** | Typically used to modify or access class-level attributes or methods | Used for utility functions that don't depend on class-specific attributes or methods |
| ****Access**** | Can access and modify class-level attributes and call other class methods using **cls** | Cannot access or modify class-level attributes directly |
| ****Instance Aware**** | Yes | No |

**3)compare and contrast:**

**Types of variable**

In programming, variables are used to store data in memory for later use. They are an essential concept in any programming language, including Python. Variables can be classified into different types based on their scope, lifetime, and accessibility. Let's compare and contrast the main types of variables:

**Global variable:**

Global variables are declared outside of any function or class and can be accessed from anywhere in the code, both inside functions and classes.

Global variables can be accessed and modified from any part of the code, making them widely available.

**Local Variable:**

Local variables are declared within a specific block, such as inside a function or a loop, and can only be accessed within that block.

Local variables are limited in scope and can't be accessed from outside their block.

**Instance Variables(Non static attributes):**

Instance variables are specific to objects instantiated from a class. Each instance of the class has its own copy of instance variables.

Instance variables are accessible from methods within the class, using the *self* keyword. Each instance has its unique values for instance variables.

**Class variables(static attributes):**

Class variables are shared among all instances of a class. They are declared within the class but outside of any method.

Class variables are accessible both from the class itself and its instances. However, if modified through an instance, it creates a new instance variable that shadows the class variable.

**4)What is the difference between Compile time polymorphism and Run time polymorphism?**

| **Compile Time Polymorphism** | **Run time Polymorphism** |
| --- | --- |
| In Compile time Polymorphism, the call is resolved by the compiler. | In Run time Polymorphism, the call is not resolved by the compiler. |
| It is also known as Static binding, Early binding and overloading as well. | It is also known as Dynamic binding, Late binding and overriding as well. |
| It is achieved by function overloading and operator overloading | It is achieved by virtual functions and pointers. |
| Compile time polymorphism is less flexible as all things execute at compile time. | Run time polymorphism is more flexible as all things execute at run time. |
| Inheritance is not involved. | Inheritance is involved. |

**5)What is the difference between abstraction and encapsulation? explain with program example.**

**Ans:**

**Abstraction Encapsulation**

|  |  |
| --- | --- |
| It is the process of gaining information. | It is a method that helps wrap up data into a single module. |
| The problems in this technique are solved at the interface level. | Problems in encapsulation are solved at the implementation level. |
| It helps hide the unwanted details/information. | It helps hide data using a single entity, or using a unit with the help of method that helps protect the information. |
| It can be implemented using abstract classes and interfaces. | It can be implemented using access modifiers like public, private and protected. |
| The complexities of the implementation are hidden using interface and abstract class. | The data is hidden using methods such as getters and setters. |
| Abstraction can be performed using objects that are encapsulated within a single module. | Objects in encapsulation don't need to be in abstraction. |