**Abstraction:**

Abstraction is used to hide the internal functionality of the function from the users. The users only interact with the basic implementation of the function, but inner working is hidden. User is familiar with that ****"what function does"**** but they don't know **"**how it does**."**

abstract classes and abstract methods are implemented using the abc module, which stands for Abstract Base Classes. The @abstractmethod decorator is used to declare abstract methods within an abstract class.

1. **Abstract Class:**
   * An abstract class is a class that cannot be instantiated and typically serves as a base class for other classes.
   * Abstract classes are defined using the ABC (Abstract Base Class) metaclass from the abc module.
2. **Abstract Method:**
   * An abstract method is a method declared in an abstract class but has no implementation in the abstract class itself.
   * Subclasses inheriting from the abstract class are required to provide an implementation for abstract methods.

**Example:1**

from abc import ABC, abstractmethod

class Shape(ABC):

@abstractmethod

def area(self):

pass

@abstractmethod

def perimeter(self):

pass

class Circle(Shape):

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

self.radius = radius

def area(self):

return 3.14 \* self.radius\*\*2

def perimeter(self):

return 2 \* 3.14 \* self.radius

class Square(Shape):

def \_\_init\_\_(self, side\_length):

self.side\_length = side\_length

def area(self):

return self.side\_length\*\*2

def perimeter(self):

return 4 \* self.side\_length

# Attempting to create an instance of the abstract class will raise an error

# shape = Shape() # Uncommenting this line would raise a TypeError

# Creating instances of concrete subclasses

circle = Circle(radius=5)

square = Square(side\_length=4)

print("Circle - Area:", circle.area(), "Perimeter:", circle.perimeter())

print("Square - Area:", square.area(), "Perimeter:", square.perimeter())

**@staticmethod in python:**

A static method is a method which is bound to the class and not the object of the class. It can’t access or modify class state. It is present in a class because it makes sense for the method to be present in class. A static method does not receive an implicit first argument.

the **@staticmethod** decorator is used to define a static method within a class. A static method is a method that belongs to a class rather than an instance of the class. It is not bound to an instance and does not have access to the instance itself (via **self**).

**Example:**

class MathOperations:

@staticmethod

def add(x, y):

return x + y

@staticmethod

def subtract(x, y):

return x - y

# Using static methods without creating an instance of the class

result\_add = MathOperations.add(5, 3)

result\_subtract = MathOperations.subtract(8, 4)

print("Result of addition:", result\_add)

print("Result of subtraction:", result\_subtract)

# @classmethod in Python:

the @classmethod decorator is used to define a class method within a class. A class method is a method that is bound to the class and not the instance of the class. It takes the class itself as its first parameter, conventionally named cls.

**Example:1**

class MyClass:

class\_variable = "I am a class variable"

@classmethod

def print\_class\_variable(cls):

print(cls.class\_variable)

# Using the class method without creating an instance

MyClass.print\_class\_variable()

**Example:2**

class MathOperations:

@classmethod

def add(cls, x, y):

print(f"Performing addition using class method ({cls.\_\_name\_\_})")

return x + y

@classmethod

def multiply(cls, x, y):

print(f"Performing multiplication using class method ({cls.\_\_name\_\_})")

return x \* y

# Using class methods without creating an instance of the class

result\_add = MathOperations.add(5, 3)

result\_multiply = MathOperations.multiply(4, 5)

print("Result of addition:", result\_add)

print("Result of multiplication:", result\_multiply)

**Class Variable:**

A class variable in Python is a variable that is shared by all instances of a class. It is defined within the class but outside any class methods or instance methods. Class variables are accessed using the class name rather than an instance of the class. They are often used to store data that is common to all instances of the class.

**Example:**

class Counter:

# Class variable

count = 0

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

# Incrementing the class variable in the constructor

Counter.count += 1

# Creating instances of the class

instance1 = Counter()

instance2 = Counter()

instance3 = Counter()

# Accessing the class variable

print("Number of instances created:", Counter.count)