**Abstract Base Class and Abstract Method in Python**

**Code Summary**

You defined an abstract base class Shape with an abstract method area. Two derived classes, Circle and Square, implement the area method.

python

CopyEdit

from abc import ABC, abstractmethod

import math

class Shape(ABC):

@abstractmethod

def area(self):

pass # Abstract method with no implementation

class Circle(Shape):

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

self.\_radius = radius

def area(self):

result = math.pi \* self.\_radius \*\* 2

print(f'Circle Area: {result}')

return result

class Square(Shape):

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

self.\_side = side

def area(self):

result = self.\_side \*\* 2

print(f'Square Area: {result}')

return result

# Usage

sq = Square(4)

sq.area()

cir = Circle(2)

cir.area()

**Explanation**

**Abstract Class**

* An **abstract class** cannot be instantiated directly.
* It serves as a **template** for other classes.
* Declared by inheriting from ABC (Abstract Base Class) in the abc module.
* Defines one or more **abstract methods** — methods without implementation.

**Abstract Method**

* Decorated with @abstractmethod.
* Acts as a placeholder, forcing subclasses to provide their own implementation.
* Subclasses must implement **all** abstract methods to be instantiable.

**Why Use Abstract Classes?**

* To **enforce a common interface** among related classes.
* To **prevent incomplete classes** from being instantiated.
* To improve **code organization** and **maintainability** in complex systems.

**Important Notes**

* Instantiating an abstract class like Shape() raises a TypeError.
* Abstract classes support **multiple inheritance** in Python.

**Summary Table**

| **Concept** | **Description** |
| --- | --- |
| Abstract Class | Class that cannot be instantiated, designed as a base/template. |
| Abstract Method | Method without implementation, must be overridden by subclasses. |
| Usage | Enforces interface, prevents incomplete classes, improves design. |
| Multiple Inheritance | Supported in Python’s abstract classes. |

**🔷 What is an Abstract Method?**

An **abstract method** is a **placeholder** for a method that must be defined in any child class.

* It tells the child classes:

"You **must** write this method!"

**🛠️ How to Create an Abstract Class & Abstract Method**

python

CopyEdit

from abc import ABC, abstractmethod

class Animal(ABC): # Abstract class

@abstractmethod

def make\_sound(self): # Abstract method

pass

**✅ Subclass Example**

python

CopyEdit

class Dog(Animal):

def make\_sound(self):

print("Woof!")

**❌ What Raises an Error**

python

CopyEdit

a = Animal() # ❌ Cannot instantiate an abstract class

**✅ This Works**

python

CopyEdit

d = Dog()

d.make\_sound() # Output: Woof!

**💡 Simple Analogy**

* **Abstract class** = Recipe book (you can’t eat it)
* **Abstract method** = Recipe steps you must follow
* **Child class** = Real cook who follows the recipe and makes food