***What is Abstraction in Python?***

**Abstraction in python** is defined as a process of handling complexity by hiding unnecessary information from the user. This is one of the coreconcepts of object-oriented programming (OOP) languages. That enables the user to implement even more complex logic on top of the provided abstraction without understanding or even thinking about all the hidden background/back-end complexity.

That’s a very generic core topic not only limited to object-oriented programming. You can observe it everywhere in the real world or in our surroundings.

Abstraction provides a programmer to hide all the irrelevant data/process of an application to reduce complexity and increase the efficiency of the program

***Achieving Abstraction in Python:***

In Python, abstraction can be achieved by having/using abstract classes and methods in our programs.

***Understanding Abstract Methods and Classes:***

An abstract method is a method that is declared, but does not contain implementation. An abstract method in a base class identifies the functionality that should be implemented by all its subclasses. However, since the implementation of an abstract method would differ from one subclass to another, often the method body comprises just a pass statement. Every subclass of the base class will ride this method with its implementation. A class containing abstract methods is called abstract class.Python provides the abc module to use the abstraction in the Python program, syntax as:

|  |  |
| --- | --- |
| 1  2 | **from** abc **import** ABC,  **class** ClassName(ABC): |

import math

from abc import ABC,abstractmethod

class Area\_calculator(ABC):

    @abstractmethod

    def print\_area(self):

        pass

class Circle(Area\_calculator):

    def \_\_init\_\_(self, \*\*kwargs):

        self.radius = kwargs.get('radius')

    def calculate\_area(self):

        area = math.pi \* (self.radius \*\* 2)

        return area

    def print\_area(self):

        area = self.calculate\_area()

        print(f"The area of the circle with radius {self.radius} is {area:.2f}")

class Square(Area\_calculator):

    def \_\_init\_\_(self, \*\*kwargs):

        self.side = kwargs.get('side')

    def calculate\_area(self):

        area = (self.side \*\* 2)

        return area

    def print\_area(self):

        area = self.calculate\_area()

        print(f"The area of the circle with radius {self.side} is {area:.2f}")

class Rectangle(Area\_calculator):

    def \_\_init\_\_(self, \*\*kwargs):

        self.length = kwargs.get('length')

        self.width = kwargs.get('width')

    def calculate\_area(self):

        area = (self.length\*self.width)

        return area

    def print\_area(self):

        area = self.calculate\_area()

        print(f"The area of the Rectanngle is {area:.2f}")

# Usage example

# circle = Circle(radius = 15)

# circle.calculate\_area()

my\_rectangle\_object = Rectangle(length=20,width = 40)

#print(my\_rectangle\_object.print\_area())