In [1]:
```
Q1: What is Abstraction in OOPs? Explain with an example.
```

```
  Cell In[1], line 1
    Q1: What is Abstraction in OOPs? Explain with an example.
                                ^
SyntaxError: invalid syntax
```

Abstraction in object-oriented programming is the process of simplifying complex systems by modeling classes based on the essential properties and behaviors they possess. It involves hiding the complex reality while exposing only what is necessary.

Example: Consider a Car class. In an abstract sense, a car has properties like color, model, and behaviors like starting, stopping, and accelerating. In abstraction, you would focus on these essential aspects and hide the intricate details of how the engine, transmission, or braking systems work.

In [6]:
```python
class Car:
    def __init__(self, color, model):
        self.color = color
        self.model = model

    def start(self):
        print(f"The {self.color} {self.model} car is starting.")

    def stop(self):
        print(f"The {self.color} {self.model} car is stopping.")

    def accelerate(self):
        print(f"The {self.color} {self.model} car is accelerating.")
```

```
 q.2 Differentiate between Abstraction and Encapsulation. Explain with an example.
```

Abstraction focuses on exposing only the necessary features of an object and hiding the complex details. It involves using abstract classes and methods.

Encapsulation involves bundling the data (attributes) and the methods that operate on the data into a single unit, known as a class. It helps in data hiding and controlling access to the data.

In [8]:
```python
# Abstraction
from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def area(self):
        pass

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return 3.14 * self.radius * self.radius

# Encapsulation
class Student:
    def __init__(self, name, age):
        self._name = name  # encapsulated attribute
        self._age = age    # encapsulated attribute

    def get_name(self):
        return self._name

    def get_age(self):
        return self._age

# Creating instances
circle = Circle(5)
student = Student("John", 20)
```

In [ ]:
```
Q3: What is abc module in Python? Why is it used?
```

```
The abc (Abstract Base Classes) module in Python provides the infrastructure for defining abstract base classes. Abstract
base classes are classes that cannot be instantiated and are meant to be subclassed by other classes.

Purpose:

To define a common interface for a set of related classes.
To enforce that a set of methods must be implemented by all subclasses.
```

In [ ]:
```
Q4: How can we achieve data abstraction?
```

```
Data abstraction in Python can be achieved through abstract classes and abstract methods. Abstract classes cannot be
instantiated and can have abstract methods that must be implemented by their subclasses.
```

In [7]:
```
from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def area(self):
        pass

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return 3.14 * self.radius * self.radius

# Creating an instance
circle = Circle(5)
print(circle.area())  # Accessing the abstracted method
```

```
78.5
```

In [ ]:
```
Q5: Can we create an instance of an abstract class? Explain your answer.
```

```
No, we cannot create an instance of an abstract class in Python. Abstract classes are meant to be subclassed, and their
abstract methods must be implemented by the subclasses. Attempting to instantiate an abstract class directly would result
in a TypeError.
```