Abstraction in oops

Abstraction: -

It is the process of hiding the implementation and providing the functionalities to the user.

Key Characteristics of Abstraction:

1. Simplifies complex reality by modeling classes appropriate to the problem
2. Hides unnecessary details from the user
3. Exposes only relevant interfaces for interaction
4. Reduces programming complexity and effort

Functionalities of Abstraction in Python:

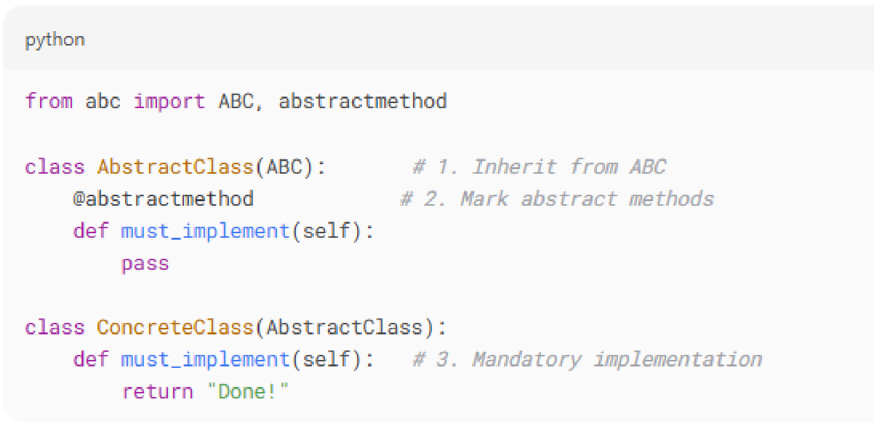
- Hides internal code
- Reduces complexity
- Makes code reusable
- Secures important data
- Supports scalable designs

How is Abstraction Implemented in Python?

In Python, abstraction is mainly implemented using:

- Abstract Methods
- Abstract Base Classes (ABC)

Example:



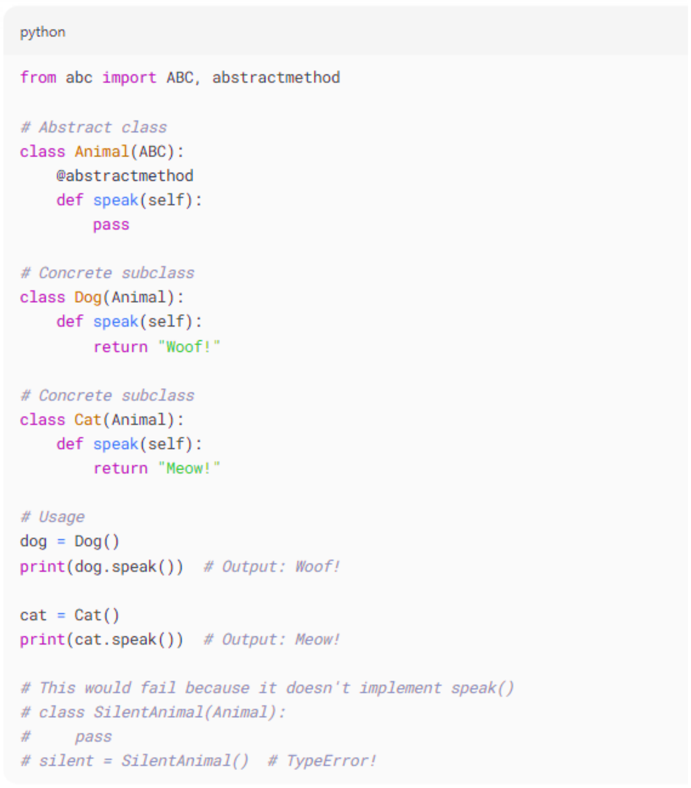
```python
from abc import ABC, abstractmethod

class AbstractClass(ABC):       # 1. Inherit from ABC
    @abstractmethod            # 2. Mark abstract methods
    def must_implement(self):
        pass

class ConcreteClass(AbstractClass):
    def must_implement(self):   # 3. Mandatory implementation
        return "Done!"
```

Abstract Class: -

An abstract class is a class that cannot be instantiated on its own and is designed to be a blueprint for other classes.

Abstract classes allow us to define methods that must be implemented by subclasses, ensuring a consistent interface while still allowing the subclasses to provide specific implementations.

Example:-

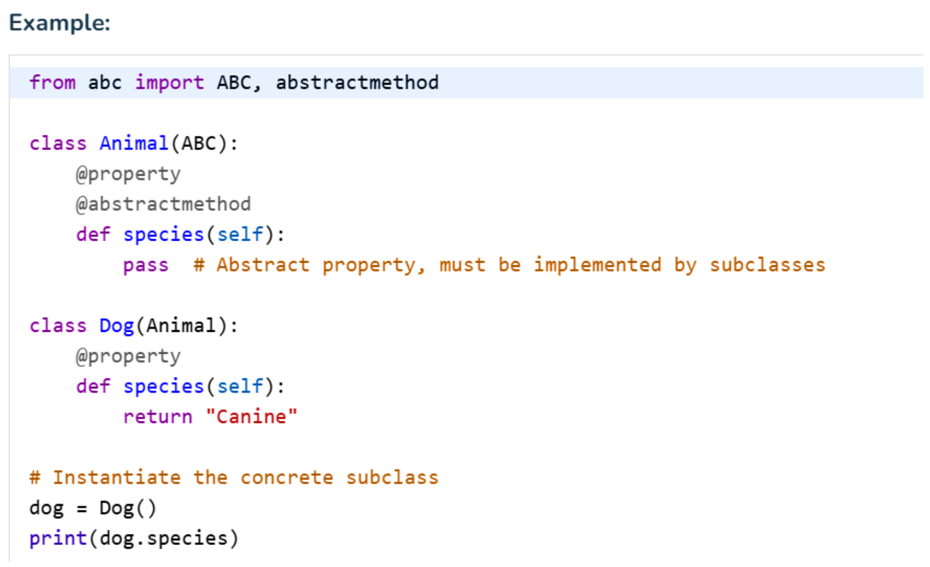


```python
from abc import ABC, abstractmethod

# Abstract class
class Animal(ABC):
    @abstractmethod
    def speak(self):
        pass

# Concrete subclass
class Dog(Animal):
    def speak(self):
        return "Woof!"

# Concrete subclass
class Cat(Animal):
    def speak(self):
        return "Meow!"

# Usage
dog = Dog()
print(dog.speak())  # Output: Woof!

cat = Cat()
print(cat.speak())  # Output: Meow!

# This would fail because it doesn't implement speak()
# class SilentAnimal(Animal):
#     pass
# silent = SilentAnimal()  # TypeError!
```

Abstract Properties

Abstract properties work like abstract methods but are used for properties.

These properties are declared with the @property decorator and marked as abstract using @abstractmethod. Subclasses must implement these properties.



Example:

```python
from abc import ABC, abstractmethod

class Animal(ABC):
    @property
    @abstractmethod
    def species(self):
        pass  # Abstract property, must be implemented by subclasses

class Dog(Animal):
    @property
    def species(self):
        return "Canine"

# Instantiate the concrete subclass
dog = Dog()
print(dog.species)
```

Explanation:

- species is an abstract property in the Animal class and it is marked as @abstractmethod.
- The Dog class implements the species property, making it a concrete subclass that can be instantiated.
- Abstract properties enforce that a subclass provides the property's implementation.

Interface: - It is the Intermediate between the service and the user or consumer.

Difference between abstract class and interface in Python?

	Abstract Class	Interface
1.	An abstract features developer's class can consist of abstract as well as concrete methods	All methods of an interface are abstract
2.	It is used when there are some common feature shared by all objects	It is used when all the feature need to be implemented differently for different objects
3.	Its developer responsibility to create a child class for the features of an abstract class	Any 3rd person will responsible for creating a child class
4.	It is comparatively fast	It is comparatively slow

Abstract Base Class:- It defines methods that must be implemented by its subclasses, ensuring that the subclasses follow a consistent structure.

ABCs allow you to define common interfaces that various subclasses can implement while enforcing a level of abstraction.

Python provides the abc module to define ABCs and enforce the implementation of abstract methods in subclasses.

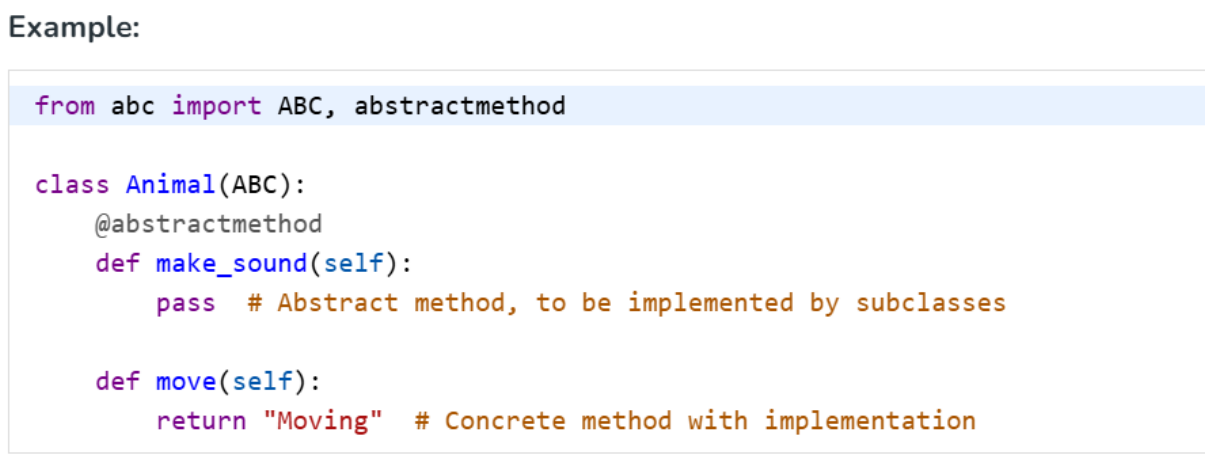
- Syntax:

```python
from abc import ABC, abstractmethod

class Shape(ABC):  # Abstract Class
    @abstractmethod
    def area(self):  # Abstract Method (no implementation)
        pass
```

Concrete Methods:-

Concrete methods are methods that have full implementations in an abstract class.

These methods can be inherited by subclasses and used directly without needing to be redefined.



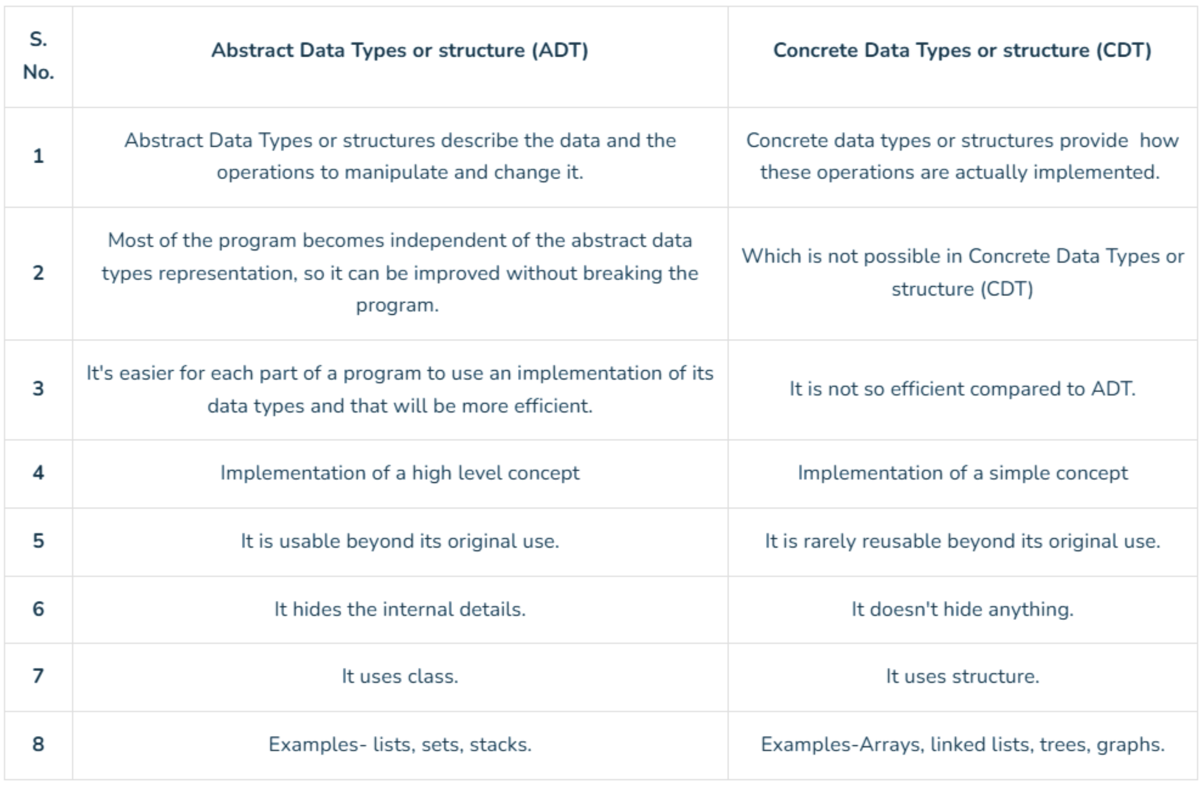
Example:

```
from abc import ABC, abstractmethod

class Animal(ABC):
    @abstractmethod
    def make_sound(self):
        pass  # Abstract method, to be implemented by subclasses

    def move(self):
        return "Moving"  # Concrete method with implementation
```

Explanation:

- The move() method is a concrete method in the Animal class. It is implemented and does not need to be overridden by the Dog class.

Difference Between Class and Object :
There are many differences between object and class. Some differences between object and class are given below:

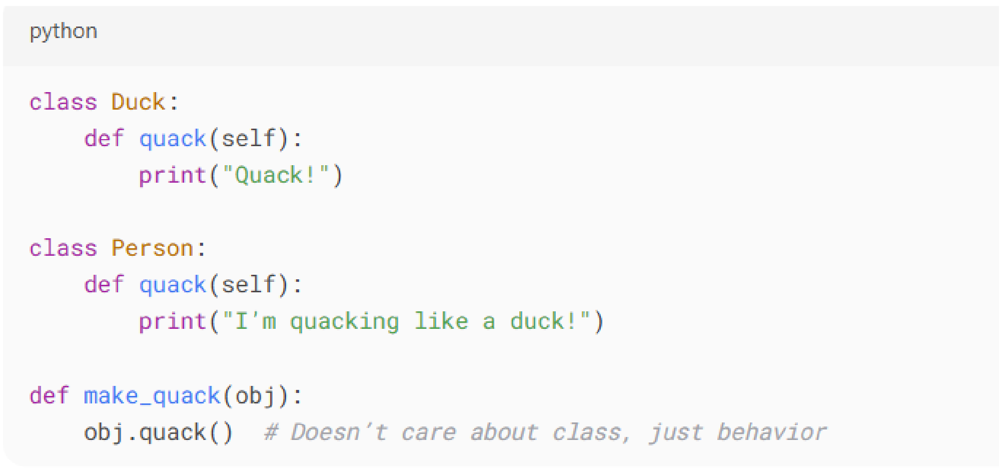


S. No.	Abstract Data Types or structure (ADT)	Concrete Data Types or structure (CDT)
1	Abstract Data Types or structures describe the data and the operations to manipulate and change it.	Concrete data types or structures provide how these operations are actually implemented.
2	Most of the program becomes independent of the abstract data types representation, so it can be improved without breaking the program.	Which is not possible in Concrete Data Types or structure (CDT)
3	It's easier for each part of a program to use an implementation of its data types and that will be more efficient.	It is not so efficient compared to ADT.
4	Implementation of a high level concept	Implementation of a simple concept
5	It is usable beyond its original use.	It is rarely reusable beyond its original use.
6	It hides the internal details.	It doesn't hide anything.
7	It uses class.	It uses structure.
8	Examples- lists, sets, stacks.	Examples-Arrays, linked lists, trees, graphs.

Partial Abstraction: Abstract class contains both abstract and concrete methods.
Full Abstraction: Abstract class contains only abstract methods (like interfaces).

Duck Typing (Python's Approach to Abstraction)

- **Concept**: "If it walks like a duck and quacks like a duck, it's a duck."
- **Key Takeaway**: Python focuses on **behaviour** rather than strict types.



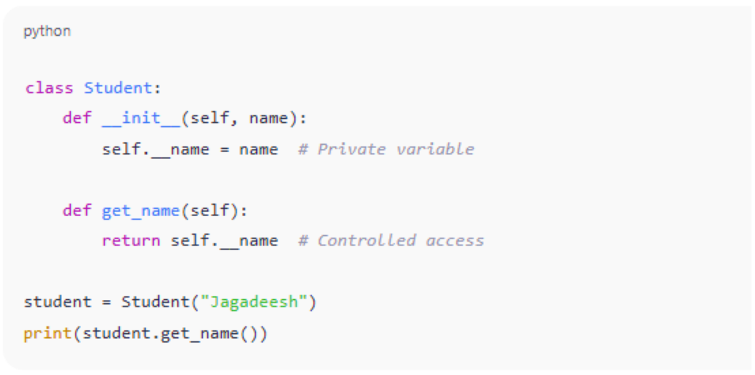
python

```python
class Duck:
    def quack(self):
        print("Quack!")

class Person:
    def quack(self):
        print("I'm quacking like a duck!")

def make_quack(obj):
    obj.quack()  # Doesn't care about class, just behavior
```

Types of Abstraction:-

1. Data Abstraction

- Data abstraction in Python is a programming concept that hides complex implementation details while exposing only essential information and functionalities to users.
- It Hides internal data and showing only necessary information.
- Achieved using access specifiers (public, private, protected).
- In Python, we can achieve data abstraction by using abstract classes and abstract classes can be created using abc (abstract base class) module and abstractmethod of abc module.
- Example



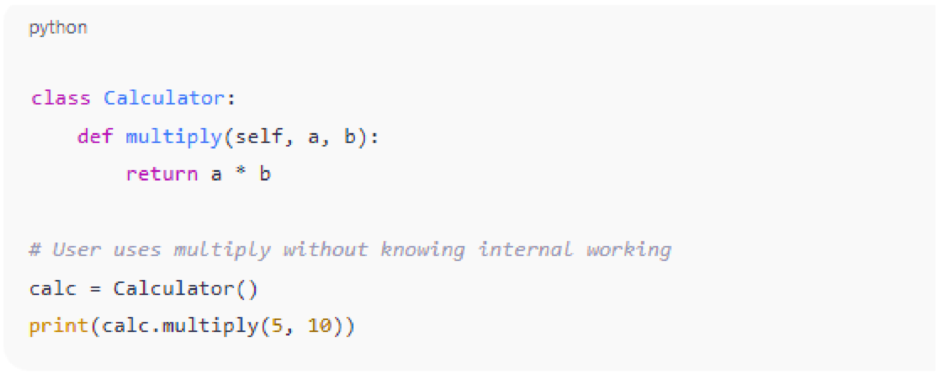
python

```python
class Student:
    def __init__(self, name):
        self.__name = name  # Private variable

    def get_name(self):
        return self.__name  # Controlled access

student = Student("Jagadeesh")
print(student.get_name())
```

2. Process Abstraction

- Also known as control abstraction, this type of abstraction focuses on hiding the implementation details of a process or a sequence of operations, exposing only the high-level functionalities and hiding the internal working of methods and functions.
- This allows users to invoke a process without needing to understand the intricate steps involved.
- You can call the method without knowing the step-by-step process inside it.
- **Abstract Methods in ABCs:** As mentioned above, abstract methods define a contract for behaviour without specifying the implementation. This forces subclasses to implement the process, thus abstracting the "how" of the process.
- **Well-defined functions and methods:** By creating functions and methods that perform specific tasks and expose a clear interface, you can abstract away the underlying logic and complexity. Users only need to know how to call the function/method and what it achieves, not the detailed steps it takes.

- Example



python

```python
class Calculator:
    def multiply(self, a, b):
        return a * b

# User uses multiply without knowing internal working
calc = Calculator()
print(calc.multiply(5, 10))
```

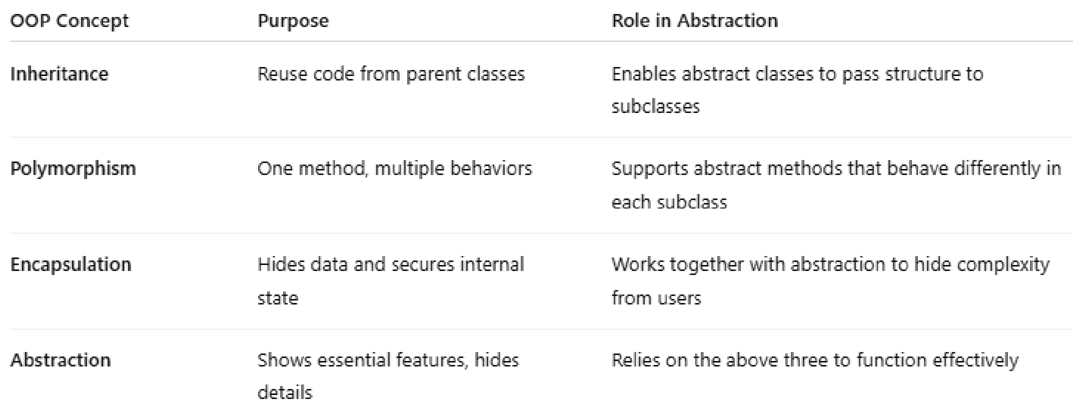
Advantages of Abstraction in Python:

- Simplifies Code Usage
- Improves Security
- Supports Reusability
- Allows Flexible Code Design

Disadvantages of Abstraction in Python:

- Complex design
- Tough for beginners
- Debugging is harder
- Slightly affects performance
- Risk of overuse (over-engineering)

- So, Inheritance, Polymorphism, and Encapsulation are the foundational pillars that support and make Abstraction powerful in Object-Oriented Programming (OOP).



OOP Concept	Purpose	Role in Abstraction
Inheritance	Reuse code from parent classes	Enables abstract classes to pass structure to subclasses
Polymorphism	One method, multiple behaviors	Supports abstract methods that behave differently in each subclass
Encapsulation	Hides data and secures internal state	Works together with abstraction to hide complexity from users
Abstraction	Shows essential features, hides details	Relies on the above three to function effectively

So, Abstraction in Object-Oriented Programming (OOP) is the process of hiding internal implementation details and exposing only the essential functionalities to the user. It focuses on what an object does rather than how it does it. Abstraction is supported by the core OOP principles: inheritance, which passes structure and properties from parent to child classes; polymorphism, which allows the same method to behave differently in different classes; and encapsulation, which hides internal data and protects it from unauthorized access. Together, these concepts simplify complex systems, enhance code security, improve reusability, and make the system more manageable, although they may introduce design complexity and a slight learning curve