OOPS IN Python

1. Class
2. Objects/Instances
3. Constructors (Default,Parameterised)
4. Attributes (Object Attributes > Class Attributes)
5. Creating Methods (....)
6. Decorators (Staticmethod, Classmethod, Property,GetterSetter)

[ ] ↳ 15 cells hidden

** IMPORTANT Concepts in OOPs**

1. **Delete Keyword**

- Del keyword :- used to delete object properties or object itself.

2. **Private(Like) attributes and methods**

- Private function can be accessed by only internal function
- We can use it externally through internal access.

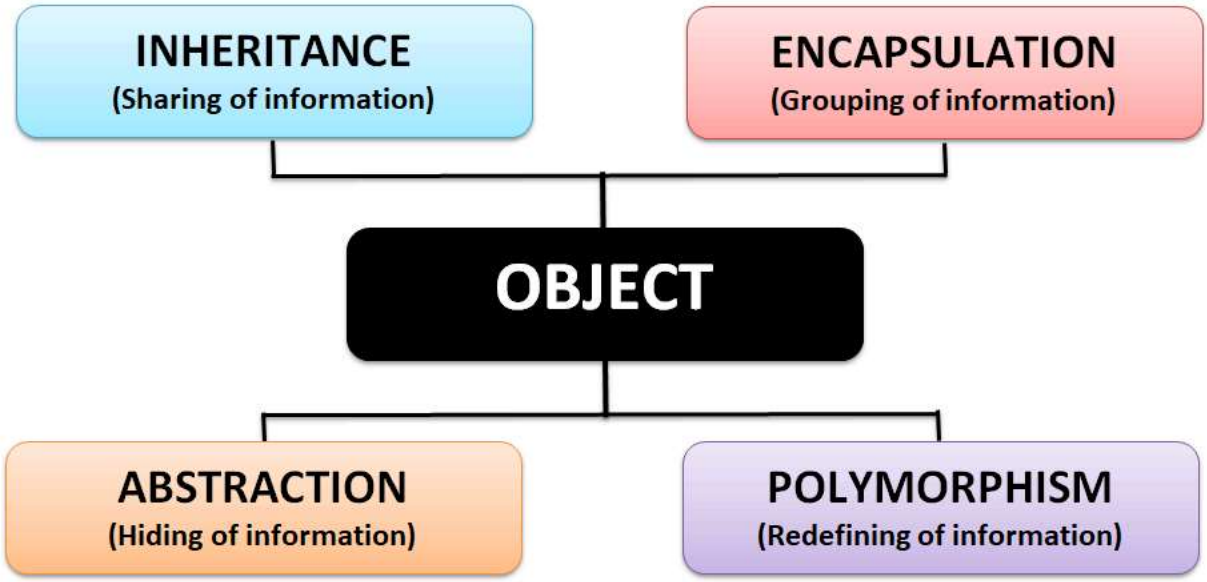
3. **Class Method Decorator**

- Static method (no self arg)
- Class Method (cls key arg is used)
- Instance Method(self arg is used)

4. **Property Decorator**

[ ] ↳ 15 cells hidden

Four Pillars of OOPS



ABSTRACTION

1. Abstraction is one of the fundamental principles of object-oriented programming (OOP).
2. It involves simplifying complex systems by hiding unnecessary details and exposing only essential information to the user.
3. In the context of classes, abstraction is achieved through abstract classes and interfaces.
4. An abstract class is a class that cannot be instantiated directly. Instead, it serves as a blueprint for other classes (concrete classes) to inherit from.
5. It may contain both abstract methods (methods without implementation) and concrete methods (methods with implementation).
6. Subclasses are required to provide implementations for the abstract methods defined in the abstract class.
7. This ensures that all subclasses adhere to a specific contract, while allowing them to define their own specific behaviors.

[ ] ↳ 2 cells hidden

Encapsulation

1. Encapsulation is one of the fundamental concepts in object-oriented programming (OOP).
2. It describes the idea of bundling data and the methods that operate on that data within a single unit, or class.

3. This bundling provides several benefits, including data hiding, code reusability, and increased maintainability.

4. Data Hiding:

- Encapsulation helps protect the internal state of an object from direct external access.
- It enforces controlled access to the data through methods (getters and setters).
- This prevents accidental modification of data and improves the overall integrity of the system.

5. Code Reusability:

- By encapsulating data and methods, you can reuse the same class in different parts of your program.
- You can create multiple objects from the same class, and each object will have its own data and methods.

6. Increased Maintainability:

- When data and methods are encapsulated within a class, changes to the internal implementation of the class
- do not necessarily affect other parts of the program, as long as the interface (public methods) remains the same.
- This makes it easier to maintain and update your code over time.

[ ] ↳ 1 cell hidden

Inheritence

Inheritance is a mechanism in object-oriented programming that allows you to create new classes (derived classes)based on existing classes (base classes). The derived class inherits the attributes and methods of the base class, and can also add its own attributes and methods or override existing ones.

1. Types of Inheritance:

- Single Inheritance: A class inherits from a single base class. (e.g., Dog inherits from Animal)
- Multiple Inheritance: A class inherits from multiple base classes. (Less common in Python due to potential ambiguity)
- Multilevel Inheritance: A class inherits from a derived class, creating a chain of inheritance.
- Hierarchical Inheritance: Multiple derived classes inherit from a single base class. (e.g., Dog and Cat both inherit from Animal)
- Hybrid Inheritance: A combination of multiple inheritance types.

2. Benefits of Inheritance:

- Code Reusability: Avoids redundant code by inheriting attributes and methods from base classes.
- Extensibility: Easily extend the functionality of existing classes without modifying them.
- Polymorphism: Objects of different classes can be treated as objects of a common base class.
- Maintainability: Changes to the base class automatically reflect in derived classes.

[ ] ↳ **4 cells hidden**

Polymorphism

1. Polymorphism (meaning "many forms") is a key concept in object-oriented programming.
2. It allows objects of different classes to be treated as objects of a common type.
3. This means that the same method call can behave differently depending on the specific object it is called on.
4. There are two main types of polymorphism:

4.1. Compile-time Polymorphism (Method Overloading):

- Not directly supported in Python.
- In languages like Java or C++, you can have multiple methods with the same name but different parameters within the same class.
- The compiler determines which method to call based on the arguments passed at compile time.

4.2. Runtime Polymorphism (Method Overriding):

- Achieved through inheritance and method overriding.
- A subclass provides a specific implementation for a method that is already defined in its superclass.
- The decision of which method to call is made at runtime based on the actual object type.

[ ] ↳ **2 cells hidden**

Dunder Functions

[ ] ↳ **3 cells hidden**

Practice Questions

Q1. Create student class that takes name & marks of three subjects as argument in constructors. Create method to print average.

Q2. Create account class with 2 attributes - balance & account no. Create methods for debit, credit & printing the balance.

Q3. Define a circleclass to create a circle with radius r using constructor. Define an area Method of cxlass which calc area of the circle. Define perimeter method of the class which calc perimeter of the circle.

Q4. Define a Employeeclass with attributes role, dept & salary. This class also has a showDetails() method

Q5. Create an engineer class that inherits properties from Employee and has additional attributes : name & age

Q6.Create a class called orer which stores item & its price. Use dunder function gt() to convey that: order1>order2 if price of order1>price of order2

↳ 14 cells hidden