Object-Oriented Programming (OOP)
Object-Oriented Programming (OOP) is a programming paradigm that uses objects and classes to structure software. It emphasizes principles like encapsulation, inheritance, and polymorphism to create reusable and modular code.

Class in OOP
A class is a blueprint for creating objects. It defines attributes (variables) and behaviors (methods) that objects of the class will have.

Object in OOP
An object is an instance of a class. It represents a real-world entity with attributes (data) and methods (functions) that define its behavior.

Difference Between Abstraction and Encapsulation

- **Abstraction** hides the implementation details and only exposes the necessary functionality to the user.
- **Encapsulation** restricts direct access to an object's data and allows modification through controlled methods.

Dunder Methods in Python
Dunder (double underscore) methods, also called magic methods, are special methods in Python that start and end with double underscores (e.g., `__init__`, `__str__`, `__repr__`). They define object behaviors and allow customization of built-in operations.

Concept of Inheritance in OOP
Inheritance is a mechanism in OOP that allows a class (child class) to acquire the properties and methods of another class (parent class). This promotes code reuse and hierarchical relationships.

Polymorphism in OOP
Polymorphism allows objects of different classes to be treated as objects of a common superclass. It enables methods to have the same name but different implementations in different classes.

How Encapsulation is Achieved in Python
Encapsulation in Python is achieved using private (`__variable`), protected (`_variable`), and public attributes and methods. Private members restrict access outside the class, ensuring data security.

Constructor in Python
A constructor is a special method (`__init__`) in a class that gets called automatically when an object is created. It is used to initialize object attributes.

Class and Static Methods in Python

- **Class Methods (`@classmethod`)**: Methods that operate on the class level and have access to class attributes.
- **Static Methods (`@staticmethod`)**: Methods that do not access instance or class-specific data and work independently.

Method Overloading in Python
Method overloading refers to defining multiple methods with the same name but different parameters. Python does not support method overloading directly but allows default arguments to achieve similar functionality.

Method Overriding in OOP
Method overriding occurs when a child class provides a specific implementation of a method that is already defined in its parent class. The child class method replaces the parent class method when called through the child class object.

Property Decorator in Python
The `@property` decorator allows a method to be accessed like an attribute. It is used to implement getter and setter methods in an elegant way.

Importance of Polymorphism in OOP
Polymorphism allows code to be more flexible and reusable by enabling a single interface for different data types. It simplifies code maintenance and enhances extensibility.

Abstract Class in Python
An abstract class is a class that cannot be instantiated and contains one or more abstract methods. Abstract methods are defined in the parent class but must be implemented in derived classes. It is created using the `ABC` module.

Advantages of OOP

- Code Reusability (via inheritance)
- Scalability and maintainability
- Data security through encapsulation
- Easier debugging and organization

Multiple Inheritance in Python
Multiple inheritance allows a class to inherit from more than one parent class. It enables the child class to acquire attributes and methods from multiple sources.

Difference Between Class Variable and Instance Variable

- **Class Variable**: Shared across all instances of a class and defined at the class level.

- **Instance Variable**: Unique to each object and defined within the constructor (`__init__`).

Purpose of `__str__` and `__repr__` Methods in Python

- `__str__`: Defines a human-readable string representation of an object.
- `__repr__`: Returns an official string representation of an object, mainly for debugging.

Significance of `super()` Function in Python

The `super()` function allows a child class to access methods from its parent class, facilitating method overriding and avoiding code duplication.

Significance of `__del__` Method in Python

The `__del__` method is the destructor method in Python, which gets called when an object is deleted or goes out of scope. It is used for cleanup operations.

Difference Between `@staticmethod` and `@classmethod` in Python

- `@staticmethod`: Does not require `self` or `cls` and operates independently.
- `@classmethod`: Takes `cls` as a parameter and can modify class attributes.

How Polymorphism Works in Python with Inheritance

Inheritance enables polymorphism by allowing subclasses to override methods from the parent class, ensuring that different objects respond to the same function call in different ways.

Method Chaining in Python OOP

Method chaining allows multiple method calls on the same object in a single statement by returning `self` at the end of each method.

Purpose of `__call__` Method in Python

The `__call__` method allows an instance of a class to be called as if it were a function, enabling objects to behave like functions.