Python

SARAVANA

Object-Oriented Programming (OOP)

Introduction to OOP

- **Definition**: Object-Oriented Programming is a programming paradigm based on the concept of "objects," which can contain data in the form of fields (attributes) and code in the form of procedures (methods).
- **Purpose**: To increase the modularity, reusability, and maintainability of code.

OOPS Concepts

Class

Objects

Polymorphism

Encapsulation

Inheritance

Data Abstraction

Classes and Objects

Class

A class is a blueprint for creating objects. It defines a set of attributes and methods that the created objects (instances) will have.

Class: A blueprint for creating objects with a set of methods and attributes.

Objects

An object is an instance of a class. It is created using the class name followed by parentheses.

Object: An instance of a class, containing data and methods defined in the class.

Inheritance & Polymorphism

Inheritance allows a class to inherit attributes and methods from another class. The class that inherits is called a derived or child class, and the class being inherited from is called a base or parent class.

Polymorphism allows methods to do different things based on the object it is acting upon. It means "many shapes" and can be implemented through method overriding and operator overloading.

Combining inheritance and polymorphism allows a base class to define a method that can be overridden by derived classes. This lets the base class method call the appropriate derived class method at runtime.

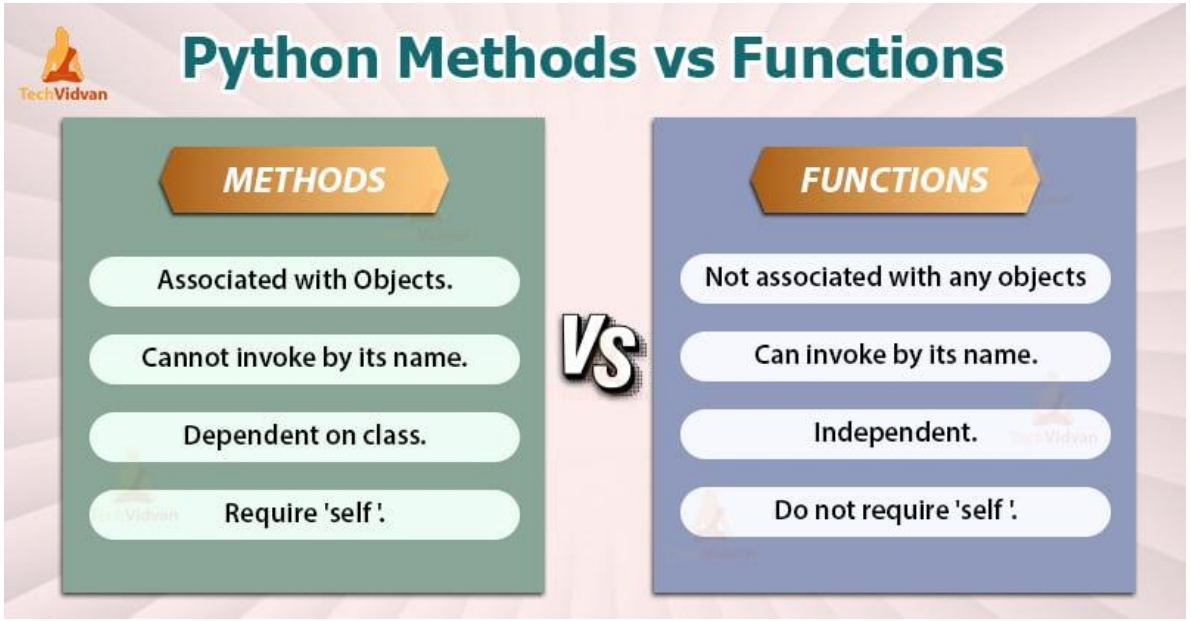
Encapsulation

Encapsulation is the concept of wrapping data (attributes) and methods into a single unit (a class) and restricting access to some of the object's components. This is typically achieved through private and public attributes and methods.

Method vs Function

Function: A block of code that performs a specific task and can be defined outside of a class.

Method: A function defined within a class and used to interact with the object's attributes.



Methods & Attributes

- **Methods**: Functions defined within a class that operate on the class's attributes.
- **Attributes**: Variables that belong to an object.

- **Types of Attributes**:
  - **Instance Attributes**: Unique to each object.
  - **Class Attributes**: Shared among all instances of the class.

Decorators

Decorators in Python are a powerful feature that allows you to modify the behavior of functions or methods dynamically.

They provide a way to wrap additional functionality around existing functions or methods without modifying their actual code.

Decorator is a function that takes another function and extends its behavior without explicitly modifying it.

Decorators are often used for logging, access control, instrumentation, and caching

A decorator is typically applied to a function using the @ symbol above the function definition

Built-in Decorators

`@staticmethod`

Purpose: Defines a method that doesn’t operate on an instance or the class itself. It behaves like a regular function but is associated with the class's namespace.

Static Methods: Do not modify class or instance state.

`@classmethod`

Purpose: Defines a method that is bound to the class and not the instance. The method receives the class itself (`cls`) as its first argument.

Class Methods: Access and modify class state.

`@property`

Purpose: Converts a method into a read-only property. It allows you to define methods that can be accessed like attributes.

Private & Static methods

Private attributes and methods are those that are intended to be accessed only within the class. They are not meant to be accessible from outside the class.

Syntax: Prefix the attribute or method name with double underscores (__).

static methods are methods that do not modify or access the state of the class or instance. They are utility functions that belong to the class but don't require access to class or instance-specific data.

Syntax: Use the `@staticmethod` decorator

Summary

Classes: Blueprints for creating objects.

Objects: Instances of classes.

Methods: Functions defined in classes.

Attributes: Variables within classes.

Multithreading and Multiprocessing in Python

Understanding Concurrent and Parallel Execution

Multithreading: Concurrent execution of threads within the same process. Best for I/O-bound tasks.

Multiprocessing: Parallel execution of processes, each with its own Python interpreter and memory space. Best for CPU-bound tasks.

Modules & Packages

What is a Module?

A module is a file containing Python code. Modules are used to encapsulate code into reusable components. They can include functions, classes, and variables, and can be imported into other modules or scripts to use their functionality.

To define a module, simply create a `.py` file containing Python code

Modules

Create a Module

Import a Module

Run the Module

`dir()` with Modules

The `dir()` function returns a list of attributes and methods of an object, including modules.

Python looks for modules in specific locations which are listed in `sys.path`

Module Namespace - Each module in Python has its own namespace. This means that variables and functions defined in one module are not directly visible to other modules unless explicitly imported. This helps in avoiding naming conflicts.

Packages

Defining a package

Importing a package

Installing third party package

A module is a single file containing Python code. It can define functions, classes, and variables, and it can also include runnable code.

A package is a collection of modules organized in a directory hierarchy.

Introduction to Database Operations

Introduction to MySQL

MySQL: An open-source relational database management system (RDBMS) that uses Structured Query Language (SQL) for database access. It is widely used for web applications and offers a robust, scalable solution for managing data.

Introduction to Oracle

Oracle Database: A powerful and widely-used RDBMS developed by Oracle Corporation. It supports complex queries, transactions, and high levels of concurrency and scalability.

Thanks