Lecture 2.7

Instance method, Class method, static method & Encapsulation

Agenda

- Introduction to Python Methods
- Instance Methods
- Class Methods and @classmethod
- Static Methods and @staticmethod
- Comparison of Method Types
- Data Encapsulation and Access Modifiers

Recap: Classes and Objects

- A Class is a blueprint for creating objects.
- An Object is an instance of a class.
- Constructors (__init__) initialize the object's state.
- Self: A reference to the current instance of the class.

Introduction to Methods in Python

- Methods are functions defined inside a class body.
- They represent the behaviors of an object.
- Python provides three main types of methods:
- 1. Instance Methods
- 2. Class Methods
- 3. Static Methods
- Each serves a different purpose based on what data they need to access.

Instance Methods

- Most common type of method in Python classes.
- Used to access or modify the object's state (instance variables).
- Must have 'self' as the first parameter.
- Can access both instance attributes and class attributes (via self.__class__).

Instance Methods: Example

- class Student:
- def __init__(self, name, marks):
- self.name = name
- self.marks = marks
- def display_info(self): # Instance Method
- print(f'Student: {self.name}, Marks: {self.marks}')
- s = Student('Alice', 85)
- s.display_info()

Class Methods

- Used to access or modify class state (class variables).
- Bound to the class, not the object of the class.
- Decorated with '@classmethod'.
- Takes 'cls' as the first parameter instead of 'self'.
- Often used as alternative constructors (factory methods).

Class Methods: Example

- class Student:
- school_name = 'ABC High School'
- @classmethod
- def change_school(cls, new_name):
- cls.school_name = new_name
- Student.change_school('XYZ High School')
- print(Student.school_name) # Output: XYZ High School

Static Methods

- Independent of class and instance states.
- Bound to the class, but cannot modify class or instance attributes.
- Decorated with '@staticmethod'.
- Does not take default parameters like 'self' or 'cls'.
- Used as utility or helper functions that logically belong to the class.

Static Methods: Example

- class MathOperations:
- @staticmethod
- def add_numbers(x, y):
- return x + y
- result = MathOperations.add_numbers(5, 10)
- print(result) # Output: 15

Comparing the Method Types

- Instance Method -¿ Takes 'self' -¿ Accesses Instance & Class State
- Class Method -¿ Takes 'cls' (@classmethod) -¿ Accesses Class State Only
- Static Method -¿ No default params (@staticmethod) -¿ No State Access

Data Encapsulation: Concept

- Encapsulation is bundling data (variables) and methods that operate on the data into a single unit (class).
- It restricts direct access to some of an object's components.
- Prevents accidental modification of data.
- In Python, encapsulation is achieved through access modifiers.

Access Modifiers in Python

- Python uses naming conventions instead of strict keywords.
- Public Members: Accessible from anywhere (default). e.g., 'name'
- Protected Members: Accessible within the class and its subclasses. Prefix with one underscore: '_name'
- Private Members: Accessible only within the class. Prefix with two underscores: '__name'

Encapsulation Example

- class BankAccount:
- def __init__(self, balance):
- self.__balance = balance # Private attribute
- def get_balance(self): # Getter
- return self.__balance
- def set_balance(self, amount): # Setter
- if amount ¿ 0: self.__balance = amount
- account = BankAccount(1000)
- print(account.get_balance()) # Output: 1000

Summary

- Instance methods handle object-level state ('self').
- Class methods handle class-level state ('cls', @classmethod).
- Static methods provide utility functions (@staticmethod).
- Encapsulation bundles data and methods, protecting data integrity.
- Access modifiers (public, protected, private) control visibility.