Unit:3; Class:2

Attributes and Methods

Instructor: Musfique Ahmed

Class Variables vs Instance Variables

Key Concepts:

- **Instance Variable:** Belongs to the object; unique to each instance.
- **Class Variable:** Shared across all instances of the class.

```
class Car:
    wheels = 4  # Class variable

    def __init__(self, brand, color):
        self.brand = brand  # Instance variable
        self.color = color  # Instance variable
```

Practice Problem:

Create a `Student` class with:

- Class variable: `school_name = "ABC High School"`
- Instance variables: `name` and `grade`.

Write a method to display all information.

Types of Methods

Instance Methods:

- Operate on instance variables.
- Require `self` as the first parameter.

Class Methods:

- Operate on class variables.
- Use `@classmethod` decorator and `cls` as the first parameter.

Static Methods:

- Independent of both instance and class variables.
- Use `@staticmethod` decorator.

Types of Methods


```python
class Math:
    @staticmethod
    def add(a, b):
        return a + b

    @classmethod
    def description(cls):
        return "This is a Math class."
```

Practice Problem:

Create a `Circle` class with:

- A static method to calculate the area.
- A class method to display a general description of the class.

Decorators in Python

What are Decorators?

- Functions that modify the behavior of other functions or methods.
- Allow reusability and cleaner code.

How They Work:

- Use the `@decorator_name` syntax above a function/method.

Built-in Decorators:

1. `@staticmethod`
2. `@classmethod`
3. `@property`

`@property` Decorator:

- Used to define getter methods.
- Allows accessing methods like attributes.

Decorators in Python


```python
class Temperature:
    def __init__(self, celsius):
        self._celsius = celsius

    @property
    def fahrenheit(self):
        return (self._celsius * 9/5) + 32

obj = Temperature(25)
print(obj.fahrenheit)  # Output: 77.0
```

Practice Problem:

Create a `Rectangle` class:

- Attributes: `length` and `width`.
- Define a `@property` method to calculate the area.

Special Methods

`__str__` vs `__repr__`:

- `__str__`: Defines a readable string representation of an object.
- `__repr__`: Defines a developer-friendly string representation of an object.

```
class Book:
    def __init__(self, title, author):
        self.title = title
        self.author = author

    def __str__(self):
        return f"{self.title} by {self.author}"

    def __repr__(self):
        return f"Book(title='{self.title}', author='{self.author}')"

book = Book("1984", "George Orwell")
print(book)       # Output: 1984 by George Orwell
print(repr(book)) # Output: Book(title='1984', author='George Orwell')
```

Practice Problem:

Create a `Movie` class with:

- Attributes: `title` and `director`.
- Define `__str__` and `__repr__` methods for the class.

Hands-On Activity

1. Create a class `Employee` with:
   - Attributes: `name`, `position`, `salary`.
   - Methods to:
     - Display details.
     - Increase salary.
2. Add methods to demonstrate the use of `@staticmethod` and `@classmethod`.
3. Write a custom decorator to log a message before displaying employee details.

Hands-On Activity

Task:

- Revise all topics covered.
- Create a class `Library`:
  - Attributes: `name`, `books` (list).
  - Methods to:
    - **Add a book**: Take the book's title as input and append it to the `books` list.
    - **Remove a book**: Check if the title exists in the list before removing it; handle errors gracefully.
    - **Display all books**: Print the list of books in a neat format.
  - Use `__str__` for a neat display.

Example Input and Outputs:

- Input: Add "The Great Gatsby" Output: `'The Great Gatsby' has been added to the library.`
- Input: Remove "The Great Gatsby" Output: `'The Great Gatsby' has been removed from the library.`
- Input: Display books Output: `Books in [Library Name]: - The Great Gatsby`

Thank You

Do the Quiz Please, you have 10 minutes to do that!