Understanding Class and Instance Attributes in Python.

Understanding Class and Instance Attributes in Python

Introduction

In Python, attributes are variables that belong to a class or an object. There are two main types of attributes:

1. **Class Attributes**
2. **Instance Attributes**

Understanding the difference between them is very important in Object-Oriented Programming (OOP).

---

1. Class Attributes

Definition

A **class attribute** is shared by all objects (instances) of a class.

It is defined inside the class but outside methods.

Syntax

```
class ClassName:
    class_attribute = value
```

Example

```
class Student:
    school = "Ghazi University"   # Class Attribute

# Creating objects
s1 = Student()
s2 = Student()

print(s1.school)
print(s2.school)
```

Output

```
Ghazi University
Ghazi University
```

Explanation

- `school` is a class attribute.
- All objects share the same value.
- Memory is allocated only once for class attributes.

Accessing Class Attributes

You can access class attributes using:

1. Object Name

```
print(s1.school)
```

2. Class Name

```
print(Student.school)
```

Modifying Class Attributes

```
class Student:
    school = "Ghazi University"

Student.school = "Punjab University"

s1 = Student()
print(s1.school)
```

Output

```
Punjab University
```

---

2. Instance Attributes

Definition

An **instance attribute** belongs to a specific object.

Each object has its own copy of instance attributes.

They are usually created inside the `__init__()` method using `self`.

Example

```
class Student:
    def __init__(self, name, roll_no):
        self.name = name
        self.roll_no = roll_no

# Creating objects
s1 = Student("Ali", 101)
s2 = Student("Sara", 102)

print(s1.name)
print(s2.name)
```

Output

```
Ali
Sara
```

Explanation

- `name` and `roll_no` are instance attributes.
- Every object stores its own data.
- Changing one object's attributes does not affect others.

Modifying Instance Attributes

```
class Student:
    def __init__(self, name):
        self.name = name

s1 = Student("Ali")
s2 = Student("Sara")

s1.name = "Ahmed"

print(s1.name)
print(s2.name)
```

Output

```
Ahmed
Sara
```

Difference Between Class and Instance Attributes

Feature	Class Attribute	Instance Attribute
Shared or Separate	Shared by all objects	Separate for each object
Defined	Inside class	Inside methods using `self`
Memory Usage	One copy	Separate copy for each object
Accessed By	Class name or object	Object only
Purpose	Common data	Object-specific data

Example Combining Both

```python
class Student:
    school = "Ghazi University"   # Class Attribute

    def __init__(self, name):
        self.name = name          # Instance Attribute

s1 = Student("Ali")
s2 = Student("Sara")

print(s1.school)
print(s2.school)

print(s1.name)
print(s2.name)
```

Output

```
Ghazi University
Ghazi University
Ali
Sara
```

Important Concept: Attribute Lookup

Python first checks:

1. Instance attributes
2. Then class attributes

Example

```
class Student:
    school = "Ghazi University"

s1 = Student()

print(s1.school)

s1.school = "UET"

print(s1.school)
print(Student.school)
```

Output

```
Ghazi University
UET
Ghazi University
```

Explanation

- Initially, `s1.school` uses the class attribute.
- After assignment, Python creates an instance attribute for `s1`.
- The class attribute remains unchanged.

Real-Life Example

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

    def __init__(self, color):
        self.color = color   # Instance Attribute

c1 = Car("Red")
c2 = Car("Black")

print(c1.wheels)
print(c2.wheels)
```

```
print(c1.color)
print(c2.color)
```

Advantages of Class Attributes

- Saves memory
- Stores common data
- Easy to update shared information

Advantages of Instance Attributes

- Stores unique object data
- Allows flexibility
- Each object behaves independently

Common Mistake

```
class Test:
    value = []

t1 = Test()
t2 = Test()

t1.value.append(10)

print(t2.value)
```

Output

```
[10]
```

Why?

Because `value` is a class attribute shared by all objects.

Best Practice

Use:

- **Class attributes** for shared/common values
- **Instance attributes** for object-specific data

Summary

Concept	Description
Class Attribute	Shared by all objects
Instance Attribute	Unique for each object
`self.attribute`	Creates instance attributes
`ClassName.attribute`	Accesses class attributes

Tasks

Task 1

Create a `Teacher` class with:

- Class attribute: `school`
- Instance attributes: `name`, `subject`

Create two objects and display values.

Task 2

Create a `Mobile` class with:

- Class attribute: `brand`
- Instance attribute: `model`

Change the class attribute and observe the result.

Conclusion

Class and instance attributes are fundamental concepts in Python OOP.

- Use **class attributes** for shared data.
- Use **instance attributes** for unique object information.