Python Programming
Unit 05 – Lecture 02 Notes
Members, Class Attributes, GC, Abstract Classes

Tofik Ali

February 14, 2026

Contents

1 Lecture Overview

This lecture covers three practical topics used in real code:

- how Python represents "public/private" members (mostly conventions),
- how to inspect classes using built-in attributes,
- and how to design abstract classes using `abc`.

2 Core Concepts

2.1 Public and Private Members (Python Style)

Python does not enforce strict access modifiers like some languages. Instead it uses naming conventions:

- `value`: public

- `_value`: internal use ("protected" by convention)
- `__value`: name mangling (reduces accidental access)

Name Mangling

If a class defines `__balance`, Python internally renames it to include the class name. This helps prevent accidental access from subclasses.

```
class Account:
    def __init__(self):
        self.__balance = 0
```

Access is still possible (not truly private), but it is discouraged:

```
a = Account()
print(a._Account__balance)
```

2.2 Built-in Class Attributes

Some useful built-in attributes:

- `__dict__`: attribute dictionary (what the object/class contains)
- `__doc__`: documentation string (docstring)
- `__class__`: class of an object
- `__module__`: module name

```
class A:
    """Example class"""
    x = 10

print(A.__doc__)
print(A.__dict__)
```

2.3 Garbage Collection (High-Level)

Python manages memory automatically:

- reference counting frees objects when reference count becomes 0,
- cycle detector handles circular references.

In most cases, you do not need to manually free memory. Avoid relying on `__del__` for critical logic, because its timing is not guaranteed.

2.4 Abstract Classes (ABC)

An abstract class defines an interface. Subclasses must implement abstract methods.

```python
from abc import ABC, abstractmethod

class Shape(ABC):
    @abstractmethod
    def area(self) -> float:
        pass
```

3 Demo Walkthrough

File: `demo/abstract_shape_demo.py`

The demo shows:

- `Shape` cannot be instantiated (because `area` is abstract),
- `Circle` and `Rectangle` implement `area()`.

4 Interactive Checkpoints (with Solutions)

Checkpoint 1 Solution

Question: what does name mangling do?

Answer: It renames `__attr` internally to include the class name, reducing accidental access and collisions in subclasses.

Checkpoint 2 Solution

Question: why use abstract classes?

Answer: To enforce that subclasses implement required methods and to define a clean interface.

5 Practice Exercises (with Solutions)

Exercise 1: Private Balance

Task: Create an `Account` class with `__balance` and methods `deposit` and `get_balance`.

Solution:

```python
class Account:
    def __init__(self):
        self.__balance = 0

    def deposit(self, amount):
        if amount > 0:
            self.__balance += amount

    def get_balance(self):
        return self.__balance
```

Exercise 2: Abstract Class

Task: Create abstract class `Vehicle` with abstract method `start()`.

Solution:

```python
from abc import ABC, abstractmethod

class Vehicle(ABC):
    @abstractmethod
    def start(self):
        pass
```

6 Exit Question (with Solution)

Question: which module is used for abstract base classes?

Answer: `abc`