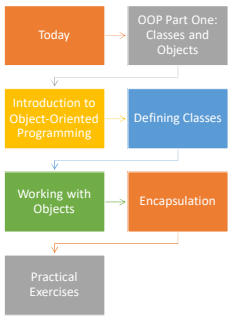
Object-Oriented Programming (OOP) Part Ⅰ

Computer Programming Ⅱ
Hamdi Abdurhman, PhD

1

Contents

- Last Time
  - File I/O



2

Introduction to Object-Oriented Programming

- **What is Object-Oriented Programming?**
  - Object-Oriented Programming (OOP) is a programming paradigm based on the concept of "objects."
  - Objects contain data, in the form of fields (attributes), and code, in the form of procedures (methods).
  - OOP languages, like Python, enable programmers to create classes that model real-world entities.
- **Key Concepts:**
  - **Classes**: Blueprints for creating objects. Define a set of attributes and methods.
  - **Objects**: Instances of classes. Represent specific examples with actual values.

3

Introduction to Object-Oriented Programming

- Example:
- Class Car might define attributes like make, model, year, and methods like `start()`, `accelerate()`.
- Object `my_car` is an instance of `Car` with specific values, like `make='Hyundai'`, model='Sonata', year=2020.

4

Benefits of Object-Oriented Programming

Encapsulation:

Bundles data and methods that operate on the data within one unit (class).

Protects data integrity by preventing external code from directly modifying internal states.

Modularity:

Classes can be developed and tested independently.

Facilitates teamwork and code maintenance.

Reusability:

Classes can be reused across different programs.

Inheritance allows new classes to be built upon existing ones.

Maintainability:

Clear structure makes it easier to manage and update code.

Enhances readability and reduces complexity.

5

Key Concepts

- **Classes:**
  - Templates that define the structure and behavior of objects.
  - Contain attributes (variables) and methods (functions).
- **Objects (Instances):**
  - Concrete occurrences of classes.
  - Have specific attribute values and can call methods.
- **Attributes:**
  - Variables that hold data specific to an object.
  - Also called properties or fields.
- **Methods:**
  - Functions defined within a class.
  - Describe behaviors and can manipulate object attributes.
- Example 1: **Basic Class and Object**

- Class: Dog

Dog
- name - age
+ bark() + sit()

- Object: my_dog (Instance of Dog)

Dog
- name - age
+ bark() + sit()

6

Key Concepts

- **Instantiation**: Creating an object from a class.
- **Attributes**: Variables that hold data; accessed using dot notation (e.g., `my_dog.name`).
- **Methods**: Functions that define behaviors; called using dot notation (e.g., `my_dog.sit()`).

Defining Classes

Basic Syntax of a Class

- Using the class Keyword:
  - To define a `class`, use the class keyword followed by the class name and a colon.
  - The class name should follow the `PascalCase` naming convention (first letter of each word capitalized).
- Example 2: Defining a Basic Class

```
# Defining a Basic Class
class Car:
    """A simple class to represent a car."""
    pass
```

- `class Car:` defines a new class named `Car`

The __init__() Method

- **Initialization of Object Attributes:**
  - The `__init__()` method is a special method called a constructor.
  - It is automatically called when an object is created from the class.
  - Used to initialize the object's attributes.
- **The self Parameter:**
  - `self` is a reference to the current instance of the class.
  - It is used to access the attributes and methods of the class.
  - The first parameter of every method in a class, including `__init__()`, should be `self`.
- Example 2: Defining a Basic Class

Creating Objects (Instances)

- **Instantiating Classes:**
  - Creating an object from a class is called instantiation.
  - Use the class name followed by parentheses and any required arguments.
- Example 3: Creating Objects

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

# Creating instances of the Dog class
my_dog = Dog('Buddy', 3)
your_dog = Dog('Max', 5)

# Accessing attributes and calling
methods
print(f"My dog's name is
{my_dog.name}.")
print(f"Your dog's name is
{your_dog.name}.")
```


Working with Objects

Accessing and Manipulating Object Data

Accessing and Modifying Attributes

- **Accessing Attributes:**
  - Use dot notation to access the attributes of an object.
  - Syntax: `object.attribute`
- **Modifying Attributes**:
  - Change the value of an attribute directly using dot notation.
  - Syntax: `object.attribute = new_value`
- **Example 4:** Accessing Attributes and Modifying Attributes

Instance Methods

- **Defining Methods within a Class:**
  - Methods are functions defined inside a class that operate on the object's attributes.
  - The first parameter of every method is `self`, referring to the instance.
  - **Example 5**: Defining instance Methods
- **The Role of `self`:**
  - `self` represents the instance of the class.
  - Used to access attributes and other methods within the class.
  - **Example 6**: Using `self` in Methods

Special Methods

- The `__str__()` Method:
  - Defines the human-readable string representation of an object.
  - Called by the str() function and print() statements.
  - Should return a string.
- The `__repr__()` Method:
  - Defines the **official** string representation of an object.
  - Called by the `repr()` function and when inspecting objects in the interpreter.
  - Should return a string that, if possible, can be used to recreate the object.
- Example 7: Implementing __str__() and __repr__()

Encapsulation

Protecting Object Data and Internal State

16

Introduction to Encapsulation

- **What is Encapsulation?**
  - Encapsulation is one of the fundamental principles of OOP.
  - It refers to the bundling of data (attributes) and methods (functions) that operate on the data within one unit, or class.
  - The internal representation of an object is hidden from the outside; only the object's own methods can directly inspect or manipulate its fields (the object's attributes).
- **Benefits of Encapsulation:**
  - **Data Hiding**: Protects the integrity of the data by preventing external code from accessing or modifying internal states directly.
  - **Modularity**: Classes can be developed, tested, and maintained independently.
  - **Control Access**: Provides controlled access to an object's attributes and methods.

17

Access Modifiers in Python

- **Understanding Access Modifiers:**
  - Unlike some other programming languages, Python does not have built-in keywords for access modifiers like **private**, **protected**, or **public**.
  - However, by convention, we use certain naming conventions to indicate the intended visibility of attributes and methods.
- **Public Attributes and Methods:**
  - **Definition**: Attributes and methods that can be accessed from outside the class.
  - **Naming Convention**: Names that do not start with an underscore (_) are considered public.
- Example 8: Public Attribute Example

18

Access Modifiers in Python

- **Private Attributes and Methods:**
  - **Definition**: Intended to be accessed only within the class itself.
  - **Naming Convention**: Names that start with double underscores (__) are considered **private**.
- **Name Mangling in Python:**
  - Python applies name mangling to private attributes to prevent accidental access.
  - The interpreter changes the name of the variable in a way to make it harder to create collisions when the class is extended.

Access Modifiers in Python

- **Protected Attributes:**
  - Definition: A convention indicating that the attribute should not be accessed directly from outside of the class, but is accessible in subclasses.
  - Naming Convention: Names that start with a single underscore (_).
- Example 10: Protected Attribute

Practical Examples

- Using Getters and Setters
  - **Purpose**: Provide controlled access to private attributes.
- Example 11: Implementing Getters and Setters

Practical Example

- **Using Property Decorators (@property)**
  - **Purpose:** Simplify getter and setter methods.
- Example 12: Using property Decorators

Reference

- Eric Matthes - Python Crash Course 3rd Edition A Hands-On, Project-Based Introduction to Programming (2023, No Starch Press)
- Mark Lutz – Learning Python 5th Edition O'REILLY

Next Class

OOP - Part Ⅱ