

# CS2 Workbook: Object-Oriented Programming (Chapter 9)

Jeremy Evert  
Southwestern Oklahoma State University

October 27, 2025



# Contents



# Chapter 1

## Classes: Introduction

### 1.1 Grouping Related Items into Objects

The physical world is made up of materials such as **wood**, **metal**, **plastic**, and **fabric**. To make sense of it, we group materials into higher-level concepts such as *chairs*, *tables*, and *televisions*. Similarly, in programming, we group lower-level data and functions into **objects**.

An **object** is a bundle of data (variables) and the operations (methods) that act on that data.

#### Example: Thinking in Objects

Object	Operations (Methods)
Chair	<code>sit()</code>
Couch	<code>sit()</code> , <code>lie_down()</code>
Drawer	<code>put_item()</code> , <code>take_item()</code>

Objects let us think about the world in terms of *what things do*, rather than what they are made of.

#### Participation Discussion

- What real-world object do you interact with daily that could be modeled as a class?
- What are its attributes (data) and behaviors (methods)?

#### 1.1.1 Programs Viewed as Objects

A program consists of variables and functions, but object-oriented programming encourages us to group related data and actions together.

Object Type	Possible Actions
Restaurant	<code>set_name()</code> , <code>add_cuisine()</code> , <code>add_review()</code>
Hotel	<code>set_name()</code> , <code>add_amenity()</code> , <code>add_review()</code>


*By organizing code this way, we create programs that are easier to read, extend, and maintain.*

## 1.2 Abstraction and Information Hiding

Abstraction occurs when we use an interface (like an oven's knob) to hide complex inner details (like heating elements).

Objects simplify complexity by hiding details and exposing only essential operations.

**Example:**



`../images/oven_abstraction_example.png`

- A car hides the details of its engine behind a steering wheel, pedals, and a dashboard.
- A Python object hides the details of its data, offering you methods like `.append()` or `.lower()`.

## 1.3 Built-in Objects in Python

Python automatically provides built-in objects, like:

- `str` — string data type (ex: `"Hello"`)
- `int` — integer data type (ex: `42`)

**Example:**

```
s1 = "Hello!"  
print(s1.upper())    # Output: HELLO!  
i1 = 130  
print(i1.bit_length()) # Output: 8
```

*Even built-in types are objects with data and methods!*

## Reflection Questions

1. What does it mean to say “a program is made of objects”?
2. Why does abstraction make code easier to understand?
3. Can you think of three real-world items that could become classes in code?





# Chapter 2

## Classes: Grouping Data

### 2.1 Why Group Data into Classes?

Many variables in a program are closely related and should be bundled together. For instance, a time value consists of hours and minutes. Instead of managing two separate variables, we can define a **class** that groups them into one logical unit.

A **class** defines a new data type that groups related data (called *attributes*) and the operations that act on them (called *methods*).

### 2.2 Constructing a Simple Class

#### The class Keyword

```
class ClassName:
    # Statement-1
    # Statement-2
    # ...
    # Statement-N
```

A class defines both the data and the behaviors of an object. The example below defines a class named **Time** with two attributes.

#### Example: Defining a Class with Two Data Attributes

```
class Time:
    """A class that represents a time of day."""
    def __init__(self):
        self.hours = 0
        self.minutes = 0
```

Here, the `__init__()` function is a special method called a **constructor**. It runs automatically when a new object (or instance) of **Time** is created.

## 2.3 Creating and Using an Object

```
my_time = Time()
my_time.hours = 7
my_time.minutes = 15

print(f"{my_time.hours} hours and {my_time.minutes} minutes")
```

**Output:**

```
7 hours and 15 minutes
```

Each variable created from the class (`my_time`) is called an **instance**. The attributes of that instance are accessed using the **dot operator** (`.`).

## 2.4 Multiple Instances of a Class

You can create multiple independent instances, each maintaining its own data.

```
time1 = Time()
time1.hours = 7
time1.minutes = 30

time2 = Time()
time2.hours = 12
time2.minutes = 45

print(f"{time1.hours} hours and {time1.minutes} minutes")
print(f"{time2.hours} hours and {time2.minutes} minutes")
```

**Output:**

```
7 hours and 30 minutes
12 hours and 45 minutes
```

## 2.5 Key Terms

**class** A grouping of related data and behaviors.

**attribute** A variable stored within a class or instance.

**method** A function that belongs to a class.

**\_\_init\_\_** The constructor method called automatically when creating a new object.

**self** Refers to the instance itself within class methods.

**instance** An individual object created from a class.

## 2.6 Practice Activity

### Activity 9.2.1 – Define and Instantiate a Class

```
class Person:
    def __init__(self):
        self.name = ""

person1 = Person()
person1.name = "Van"
print(f"This is {person1.name}")
```

#### Output:

This is Van

### Activity 9.2.2 – Create Your Own Class

Define a class called `BookData` with three attributes: `year_published`, `title`, and `num_chapters`. Create an instance of the class and assign values to its attributes.

```
class BookData:
    def __init__(self):
        self.year_published = 0
        self.title = "Unknown"
        self.num_chapters = 0

my_book = BookData()
my_book.year_published = 2001
my_book.title = "A Tale of Two Cities"
my_book.num_chapters = 45

print(f"{my_book.title} ({my_book.year_published}) has {my_book.num_chapters} chapters.")
```


#### Output:

A Tale of Two Cities (2001) has 45 chapters.

## Reflection Questions

1. What is the difference between a class and an instance?
2. Why is the `self` keyword required in class definitions?
3. How does `__init__()` help organize data?

## Visual Summary



`../images/grouping_data_into_classes.png`

# Chapter 3

## Instance Methods

### 3.1 What Are Instance Methods?

A **method** is a function that belongs to a class. An **instance method** operates on a specific object created from that class.

Each method must include the special first parameter **self**, which refers to the current instance of the class.

### 3.2 Example: Adding a Method to a Class

```
class Time:
    def __init__(self):
        self.hours = 0
        self.minutes = 0

    def print_time(self):
        print(f"Hours: {self.hours}", end=" ")
        print(f"Minutes: {self.minutes}")
```

```
time1 = Time()
time1.hours = 7
time1.minutes = 15
time1.print_time()
```

**Output:**

Hours: 7 Minutes: 15

—

### 3.3 Understanding self

The first parameter **self** provides a reference to the instance itself. When a method is called using dot notation, like `time1.print_time()`, Python automatically passes the instance

(`time1`) as the first argument.

---

## 3.4 Adding Behavior to a Class

You can add more methods to model real behavior. The example below shows an `Employee` class with a method that calculates pay.

```
class Employee:
    def __init__(self):
        self.wage = 0
        self.hours_worked = 0

    def calculate_pay(self):
        return self.wage * self.hours_worked

alice = Employee()
alice.wage = 9.25
alice.hours_worked = 35
print(f"Alice's Net Pay: ${alice.calculate_pay():.2f}")
```

**Output:**

Alice's Net Pay: \$323.75

---

## 3.5 Common Mistake: Forgetting `self`

If you forget to include `self` as the first parameter of a method, Python will raise an error:

```
class Employee:
    def __init__(self):
        self.wage = 0
        self.hours_worked = 0

    def calculate_pay():
        return self.wage * self.hours_worked

alice = Employee()
alice.wage = 9.25
alice.hours_worked = 35
print(alice.calculate_pay())
```

**Error:**

`TypeError: calculate_pay() takes 0 positional arguments but 1 was given`

---

## 3.6 Practice: Define and Use a Method

### Example Activity 9.3.1 – Adding a Method

```
class Person:
    def __init__(self):
        self.first_name = ""

    def print_name(self):
        print(f"He is {self.first_name}")

person1 = Person()
person1.first_name = "Bob"
person1.print_name()
```

**Output:**

He is Bob

---

## 3.7 Challenge: Seat Class with Instance Method

```
class Seat:
    def __init__(self):
        self.row = 0
        self.col = 0


    def print_attributes(self):
        print(f"Row: {self.row}, Column: {self.col}")

seat1 = Seat()
seat1.row = 3
seat1.col = 5
seat1.print_attributes()
```

**Output:**

Row: 3, Column: 5

## Visual Summary



`../images/instance_methods_example.png`