Lesson 7: Introduction to Object-Oriented Programming in Python

Marcin Kurzyna

Lecture Goals

This lecture introduces:

- The basic ideas behind object-oriented programming (OOP).
- How to define classes and create objects in Python.
- How to use attributes and methods.
- How inheritance works.
- Special methods such as `__init__` and `__str__`.

1 What is Object-Oriented Programming?

Object-oriented programming (OOP) is a way of structuring programs using objects. An object bundles together:

- **data** (attributes)
- **behaviour** (methods)

OOP helps organize larger programs, makes code reusable, and mirrors real-world concepts.

2 Defining a Class

A class defines the structure and behaviour of an object.

Example: A Simple Class

```
class Car:
    def __init__(self, brand, year):
        self.brand = brand
        self.year = year

    def info(self):
        return f"{self.brand} (year {self.year})"
```

```
my_car = Car("Toyota", 2010)
print(my_car.info())
```

The init Method

`__init__` is the constructor. It runs automatically when a new object is created.

Attributes and Methods

- Attributes store data inside an object.
- Methods define actions the object can perform.

3 Working With Objects

You create (instantiate) objects using the class name:

```
car1 = Car("Honda", 2015)
car2 = Car("Ford", 2020)

print(car1.brand)
print(car2.info())
```

4 Encapsulation

Encapsulation means keeping data and methods together. You can also make attributes "private" using a naming convention:

```
class BankAccount:
    def __init__(self, balance):
        self._balance = balance  # protected attribute

    def deposit(self, amount):
        self._balance += amount

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

Python does not enforce strict access control, but _ indicates intended protection.

5 Inheritance

Inheritance allows one class to take features from another class.

Example

```
class Animal:
    def speak(self):
        return "..."

class Dog(Animal):
    def speak(self):
        return "Woof!"

d = Dog()
print(d.speak())
```

Here, `Dog` inherits from `Animal` but overrides the `speak` method.

6 Special Methods

Python provides special ("magic") methods to customize class behaviour.

Common Special Methods

- `__init__` – constructor
- `__str__` – string representation
- `__len__` – length (if object has a size)

Example

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

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

    def __len__(self):
        return self.pages

b = Book("Python Basics", 250)
print(str(b))
print(len(b))
```

Summary

In this lecture, we covered:

- The concepts of classes and objects.

- Attributes, methods, and constructors.
- Inheritance and method overriding.
- Special Python methods like `__str__` and `__len__`.

7 Exercises

1. Create a class `Person` with attributes `name` and `age`. Add a method `greet()` that prints a friendly greeting.
2. Create a class `Rectangle` with attributes `width` and `height`. Add methods `area()` and `perimeter()`.
3. Create a `Student` class that inherits from `Person` and adds a `grade` attribute.
4. Create a class `Counter` with methods `increment()`, `decrement()`, and `value()`. Initialize it at zero.
5. Create a class `Movie` that overrides $_{str_{and}}$

5. `Challenge: Write a class Vector representing a 2D vector with x and y coordinates. Implement:`
    - `addition (__add__)`
    - `subtraction (__sub__)`
    - `length (__len__)`