Introduction to Software Development Week 6: Introduction to Object-Oriented Programming

1. Learning Objectives

- Understand the limitations of procedural programming for complex systems.
- Define the core concepts of OOP: Classes, Objects, Attributes, and Methods.
- Create a simple class to serve as a blueprint for objects.
- Instantiate objects from a class and use their methods and attributes.

2. Core Concepts

- **From Procedural to Object-Oriented:**
  - Procedural programming focuses on functions (actions).
  - OOP focuses on objects, which bundle data (attributes) and behaviour (methods) together. This models the real world more closely.
- **The Four Pillars of OOP (brief introduction):**
  - **Encapsulation:** Bundling data and methods that operate on the data within one unit (the class).
  - **Abstraction:** Hiding complex implementation details and showing only the necessary features of an object.
  - **Inheritance:** Creating new classes based on existing ones.
  - **Polymorphism:** Allowing objects to take on more than one form.
- **Classes and Objects:**
  - **Class:** A blueprint for creating objects. It defines a set of attributes and methods. E.g., `class Student:`.
  - **Object (Instance):** A specific instance of a class. E.g., `student_one = Student()`.
  - **Attributes:** Data associated with an object (variables within a class). E.g., `student_one.name`.
  - **Methods:** Functions associated with an object (functions defined in a class). E.g., `student_one.enroll()`.
  - **The `__init__` method:** A special "constructor" method that is called when an object is created.

3. Code Examples

```
class Dog:

  # The constructor method, initializes the object's attributes

  def __init__(self, name, age):

    self.name = name

    self.age = age



  # A method (a function belonging to the class)
```

```
    def bark(self):

        return "Woof!"


# Instantiating (creating) two Dog objects

my_dog = Dog("Fido", 5)

neighbours_dog = Dog("Rex", 2)


# Accessing attributes and calling methods

print(f"{my_dog.name} is {my_dog.age} years old.")

print(f"{my_dog.name} says: {my_dog.bark()}")
```

4. Summary

OOP is a powerful paradigm for building complex, scalable, and maintainable software. Thinking in terms of "objects" that have both data and behaviour is a fundamental shift in software design.