01_1 Introduction to OOP

Object-Oriented Programming

What is Object-Oriented Programming (OOP)?

- **Definition:**
  - 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).

- **History:**
  - Developed in the 1960s to improve code maintainability and reuse.
  - Popularized by languages like Smalltalk and later C++, Java.
  - Most modern programming languages have OOP language features.
    - Java, C++, Python, C#, Ruby, Swift, Kotlin, JavaScript, Dart, Go, etc.

Why OOP?

- **Benefits:**
  - Modularity: Code is organized into discrete objects.
  - Reusability: Objects and classes can be reused across programs.
  - Scalability: Easier to manage larger program
  - Maintainability: Simplifies debugging and updates.

Procedural Programming vs. OOP

- **Procedural Programming:**
  - Focus on functions and procedures.
  - Code is structured in a top-down manner.
- **OOP:**
  - Focus on objects and their interactions.
  - Code is structured around objects representing real-world entities.

Procedural Programming Example

```
# List to store student grades
student_grades = []

# Function to add a student grade
def add_student_grade(name, grade):
    student_grades.append({'name': name, 'grade': grade})

# Function to get the average grade
def get_average_grade():
    total = 0
    for student in student_grades:
        total += student['grade’]
    return total / len(student_grades)

# Call the functions to work
add_student_grade('Alice', 85)
add_student_grade('Bob', 90)
average = get_average_grade()
print(f'Average grade: {average}')
```

Object-Oriented Programming Example

```python
class Student:
    def __init__(self, name, grade):
        self.name = name
        self.grade = grade

class Classroom:
    def __init__(self):
        self.students = []
    def add_student(self, student):
        self.students.append(student)
    def get_average_grade(self):
        total = 0
        for student in self.students:
            total += student.grade
        return total / len(self.students)

classroom = Classroom()
classroom.add_student(Student('Alice', 85))
classroom.add_student(Student('Bob', 90))
average = classroom.get_average_grade()
print(f'Average grade: {average}')
```