01_1 Introduction to OOP

Object-Oriented Programming

이번 강의에서는 객체지향프로그래밍의 개념에 대해 알아보겠습니다.

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.

Object-Oriented Programming, 즉, 객체지향프로그래밍,
줄여서 OOP라 표시하는 이것은 무엇일까요?
OOP는 object 개념을 기반으로 하는 프로그래밍 패러다임입니다.
여기서 object란 데이터와 이를 처리하는 operation (또는 method, function)을
하나의 단위로 묶은 것을 말합니다.
OOP는 1960년대에 코드의 유지보수와 재사용 가능성을 높이기 위해 개발되었습니다.
Smalltalk와 같은 언어를 통해 처음 소개되었으며,
이후 C++와 Java 같은 언어들에 의해 널리 사용되었습니다.
요즘의 프로그래밍 언어들은 대부분 OOP 언어적인 특징들을 갖추고 있습니다.
그 예로는 Java, C++, Python, C#, Ruby, Swift, Kotlin, JavaScript, Dart, Go 등이 있습니다.

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.

OOP를 사용하면 어떤 장점이 있을까요?
첫째, 모듈성입니다.
코드를 독립된 object들로 나누어 작성함으로써 코드 관리가 용이해집니다.
둘째, 재사용성입니다.
한 번 작성한 object는 여러 프로그램에서 재사용할 수 있습니다.
셋째, 확장성입니다.
대규모 프로그래밍 프로젝트도 체계적으로 관리할 수 있습니다.
넷째, 유지보수가 용이합니다.
디버깅과 업데이트가 쉽습니다.

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, 즉, 절차적 프로그래밍과
Object-Oriented Programming, 즉, 객체지향 프로그래밍을 비교해봅시다.
Procedural 프로그래밍은 function과 procedure를 중심으로 코드를 작성하며,
주로 top-down 즉 하향식 구조로 되어 있습니다.
반면, OOP는 object들과 그들의 interaction을 중심으로 코드를 작성합니다.
Object는 real-world의 개체들을 좀 더 자연스럽게 모델링할 수 있게 합니다.

Procedural Programming Example

```python
# 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}')
```


Procedural Programming 기법을 사용한 Python 코드를 보도록 하겠습니다.
먼저 학생들의 이름과 성적을 저장하기 위한 list인 student_grades를 준비합니다.
Function "add_student_grade" 를 정의하는데, 학생 이름과 성적을 parameter로 받아
student_grades list에 추가합니다. list의 "append" function이 이를 위해 사용됩니다.
Function "get_average_grade"는 현재 list 안의 모든 grade들의 평균을 구하는 것입니다.
지금까지 이렇게 보면 프로그램에 사용되는 data인 list와 function들이 모두 독립적으로 정의되어 있습니다.
이제 procedure, 즉 function들을 실행시키면서 원하는 작업을 완성합니다.
먼저 add_student_grade를 두번 call하여 Alice와 Bob이라는 두 학생의 성적들을 student_grades list에 저장합니다.
그 다음에 get_average_grade function을 call하여 평균 점수를 계산하고, 화면에 출력합니다.

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}')
```


이 Python program은 OOP 의 특징을 잘 보여주고 있습니다.
Object를 나타내는 Class들이 정의되어 있는 것을 볼 수 있습니다.
먼저 Student class에는 data로 name과 grade라는 두 변수를 가지고 있어서,
학생들의 이름과 성적을 저장하는 하나의 entry object를 나타내고 있습니다.
Classroom object는 data로 students list를 가지고 있습니다.
Constructor, 즉, 생성자에서는 list를 빈 list로 초기화 합니다.
Classroom class는 data 이외에 data를 다루는 function들을 함께 가지고 있습니다.
먼저 add_student function은 주어진 학생 이름과 점수 parameter로 students list에 하나의 entry를 추가합니다.
get_average_grade function은 현재 list에 있는 모든 entry들의 점수의 평균을 계산합니다.
이렇게 OOP에서는 data와 function (혹은 method) 들이 함께 하나의 object를 이루게 됩니다.
프로그램 실행은 object들을 적절한 data와 함께 생성하고, object들의 method를 call 하는 것으로 이루어 집니다.
여기서는 Classroom class의 object를 하나 생성한 후
그 object의 add_student function을 두번 call하여 두 개의 data entry를 list에 추가합니다.
이 때 추가하는 data entry parameter가 Student class의 object임을 볼 수 있습니다.
Data가 추가되고 나면 classroom object의 get_average_grade function을 call하여 평균을 계산한 후
화면에 출력합니다.
이 example에서 procedural programming과 object-oriented programming의 차이가 없는 것처럼 보일 수도 있습니다.
그러나 OOP에서는 Student와 Classroom이 data + function의 조합으로 이루어진 class이며, 이 class들은 그 코드를
그대로 다른 프로그램에 이식하여 사용하거나 관리하기에 편하게 되어있다는 점을 강조하고 싶습니다.