BIND SOFT

클래스의 기본

BIND SOFT ㈜바인드소프트

BIND SOFT

목차

시작하기 전에

[핵심 키워드] : 객체, 객체 지향 프로그래밍 언어, 추상화, 클래스, 인스턴스, 생성자, 메소드

[핵심 포인트]

클래스와 객체에 대해 알아본다.

시작하기 전에

- 객체 지향 프로그래밍 (Object Oriented Programming)
  - 객체를 우선으로 생각해서 프로그래밍하는 것
  - 클래스 기반의 객체 지향 프로그래밍 언어는 클래스를 기반으로 객체 만들고, 그러한 객체를 우선으로 생각하여 프로그래밍함
    - 클래스 (class)
    - 객체 (object)

객체

- 데이터 (data)
  - 예시 – 딕셔너리로 객체 만들기

```
01  # 학생 리스트를 선언합니다.
02  students = [
03      { "name": "윤인성", "korean": 87, "math": 98, "english": 88, "science": 95 },
04      { "name": "연하진", "korean": 92, "math": 98, "english": 96, "science": 98 },
05      { "name": "구지연", "korean": 76, "math": 96, "english": 94, "science": 90 },
06      { "name": "나선주", "korean": 98, "math": 92, "english": 96, "science": 92 },
07      { "name": "윤아린", "korean": 95, "math": 98, "english": 98, "science": 98 },
08      { "name": "윤명월", "korean": 64, "math": 88, "english": 92, "sci
09  ]
10
11  # 학생을 한 명씩 반복합니다.
12  print("이름", "총점", "평균", sep="\t")
```

객체

```
13  for student in students:
14      # 점수의 총합과 평균을 구합니다.
15      score_sum = student["korean"] + student["math"] +\
16          student["english"] + student["science"]
17      score_average = score_sum / 4
18      # 출력합니다.
19      print(student["name"], score_sum, score_average, sep="\t")
```

객체

- 객체 (object)
  - 여러 가지 속성 가질 수 있는 모든 대상
  - 예시 – 객체를 만드는 함수

```
01  # 딕셔너리를 리턴하는 함수를 선언합니다.
02  def create_student(name, korean, math, english, science):
03      return {
04          "name": name,
05          "korean": korean,
06          "math": math,
07          "english": english,
08          "science": science
09      }
10
```

객체


```
11  # 학생 리스트를 선언합니다.
12  students = [
13      create_student("윤인성", 87, 98, 88, 95),
14      create_student("연하진", 92, 98, 96, 98),
15      create_student("구지연", 76, 96, 94, 90),
16      create_student("나선주", 98, 92, 96, 92),
17      create_student("윤아린", 95, 98, 98, 98),
18      create_student("윤명월", 64, 88, 92, 92)
19  ]
20
21  # 학생을 한 명씩 반복합니다.
22  print("이름", "총점", "평균", sep="\t")
23  for student in students:
24      # 점수의 총합과 평균을 구합니다.
25      score_sum = student["korean"] + student["math"] +\
26          student["english"] + student["science"]
27      score_average = score_sum / 4
28      # 출력합니다.
29      print(student["name"], score_sum, score_average, sep="\t")
```

객체

- 학생을 매개변수로 받는 형태의 함수로 만들면 코드가 더 균형 잡히게 됨

```
01  # 딕셔너리를 리턴하는 함수를 선언합니다.
02  def create_student(name, korean, math, english, science):
03      return {
04          "name": name,
05          "korean": korean,
06          "math": math,
07          "english": english,
08          "science": science
09      }
10
11  # 학생을 처리하는 함수를 선언합니다.
12  def student_get_sum(student):
13      return student["korean"] + student["math"] +\
14          student["english"] + student["science"]
15
16  def student_get_average(student):
17      return student_get_sum(student) / 4
18
19  def student_to_string(student):
20      return "{}\t{}\t{}".format(
21          student["name"],
22          student_get_sum(student),
23          student_get_average(student))
```

→ 01~23행까지 학생 객체와 관련된 부분

객체

```
24
25  # 학생 리스트를 선언합니다.
26  students = [
27      create_student("윤인성", 87, 98, 88, 95),
28      create_student("연하진", 92, 98, 96, 98),
29      create_student("구지연", 76, 96, 94, 90),
30      create_student("나선주", 98, 92, 96, 92),
31      create_student("윤아린", 95, 98, 98, 98),
32      create_student("윤명월", 64, 88, 92, 92)
33  ]
34
35  # 학생을 한 명씩 반복합니다.
36  print("이름", "총점", "평균", sep="\t")
37  for student in students:
38      # 출력합니다.
39      print(student_to_string(student))
```

→ 25~39행까지 객체를 활용하는 처리

클래스 선언하기

- 클래스 (class)
  - 객체를 조금 더 효율적으로 생성하기 위해 만들어진 구문

```
class 클래스 이름:
    클래스 내용
```

```
인스턴스 이름(변수 이름) = 클래스 이름() → 생성자 함수라고 부릅니다.
```

  - 인스턴스 (instance)
    - 생성자 사용하여 이러한 클래스 기반으로 만들어진 객체

클래스 선언하기

```
# 클래스를 선언합니다.
class Student:
    pass

# 학생을 선언합니다.
student = Student()

# 학생 리스트를 선언합니다.
students = [
    Student(),
    Student(),
    Student(),
    Student(),
    Student(),
    Student()
]
```

생성자

- 생성자 (constructor)
  - 클래스 이름과 같은 함수

```
class 클래스 이름:
    def __init__(self, 추가적인 매개변수):
        pass
```

  - self : '자기 자신' 나타내는 객체
  - self.<식별자> 형태로 접근

생성자


```
# 클래스를 선언합니다.
class Student:
    def __init__(self, name, korean, math, english, science):
        self.name = name
        self.korean = korean
        self.math = math
        self.english = english
        self.science = science

# 학생 리스트를 선언합니다.
students = [
    Student("윤인성", 87, 98, 88, 95),
    Student("연하진", 92, 98, 96, 98),
    Student("구지연", 76, 96, 94, 90),
    Student("나선주", 98, 92, 96, 92),
```

BIND SOFT

생성자

```
    Student("윤아린", 95, 98, 98, 98),
    Student("윤명월", 64, 88, 92, 92)
]

# Student 인스턴스의 속성에 접근하는 방법
students[0].name
students[0].korean
students[0].math
students[0].english
students[0].science
```

메소드

- 메소드 (method)
  - 클래스가 가지고 있는 함수

```
class 클래스 이름:
    def 메소드 이름(self, 추가적인 매개변수):
        pass
```

메소드

- 예시 – 클래스 내부에 함수 선언하기

```
01  # 클래스를 선언합니다.
02  class Student:
03      def __init__(self, name, korean, math, english, science):
04          self.name = name
05          self.korean = korean
06          self.math = math
07          self.english = english
08          self.science = science
09
10      def get_sum(self):
11          return self.korean + self.math +\
12              self.english + self.science
13
14      def get_average(self):
15          return self.get_sum() / 4
16
```

메소드

```
17      def to_string(self):
18          return "{}\t{}\t{}".format(\
19              self.name,\
20              self.get_sum(),\
21              self.get_average())
22
23  # 학생 리스트를 선언합니다.
24  students = [
25      Student("윤인성", 87, 98, 88, 95),
26      Student("연하진", 92, 98, 96, 98),
27      Student("구지연", 76, 96, 94, 90),
28      Student("나선주", 98, 92, 96, 92),
29      Student("윤아린", 95, 98, 98, 98),
30      Student("윤명월", 64, 88, 92, 92)
31  ]
32
33  # 학생을 한 명씩 반복합니다.
34  print("이름", "총점", "평균", sep="\t")
35  for student in students:
36      # 출력합니다.
37      print(student.to_string())
```

실행결과

```
이름    총점    평균
윤인성  368     92.0
연하진  384     96.0
구지연  356     89.0
나선주  378     94.5
윤아린  389     97.25
윤명월  336     84.0
```

키워드로 정리하는 핵심 포인트

- **객체** : 속성 과 메소드을 가질 수 있는 모든 것 의미
- **객체 지향 프로그래밍 언어** : 객체를 기반으로 프로그램 만드는 프로그래밍 언어
- **추상화** : 복잡한 자료, 모듈, 시스템 등으로부터 핵심적인 개념 또는 기능을 간추려 내는 것
- **클래스** : 객체를 쉽고 편리하게 생성하기 위해 만들어진 구문
- **인스턴스** : 클래스를 기반으로 생성한 객체
- **생성자** : 클래스 이름과 같은 인스턴스 생성할 때 만드는 함수
- **메소드** : 클래스가 가진 함수

확인문제

- 같은 객체라도 사용되는 프로그램에 따라서 속성이 달라질 수 있습니다. 예를 들어 가게 정보를 생각해 볼까요? 음식 주문 애플리케이션에서 가게 정보를 저장한다면 가게 이름, 전화 번호, 주소, 메뉴, 리뷰 목록 등을 저장할 것입니다. 반면 세금 관리 애플리케이션에서 가게 정보를 저장한다면 메뉴 같은 것이 무엇이 있는지는 저장할 필요가 없죠. 대신 사업자등록증 번호, 매출 상세 목록 등의 속성은 필요합니다. 이와 같이 같은 객체라도 다른 속성을 갖게 되는 경우 세 종류 정도를 생각해 적어보세요.

확인문제

- 모든 객체에는 속성과 직접 행위가 따라옵니다. 예를 들어 음식 주문 애플리케이션의 가게 정보를 생각해봅시다. 어떤 버튼을 누르면 전화가 걸리고, 어떤 버튼을 누르면 원하는 메뉴를 주문하고, 어떤 버튼을 누르면 리뷰 목록에 리뷰를 추가하는 등 특정 자극에 대응되는 행위가 있습니다. 참고로 '행위'는 작은 데이터 움직임 등도 지칭하는 것입니다. 프로그램의 객체 두 가지를 선택하여 그 행위를 다섯 가지씩 생각해 적어보세요.

확인문제

- 페이스북이라면 개인 정보, 타임라인 글, 그룹 정보 등의 객체가 있을 것이며, 개인 정보에는 이름, 이메일 ,비밀번호, 프로필 사진, 친구 목록, 타임라인 글 목록 등이 있을 것입니다.

프로그램	객체	속성
페이스북	개인 정보	이름, 이메일, 비밀번호, 프로필 사진, 프로필 설명, 친구 목록, 타임라인 글 목록 등
	타임라인 글	작성자, 게시 시간, 좋아요를 누른 친구, 댓글 등
	그룹 정보	이름, 설명, 멤버 목록 등