클래스

클래스 – 클래스란?

- 객체지향프로그래밍에서 객체를 만들기 위한 틀
- 데이터와 메소드(method)로 구성됨
- 함수와 클래스의 차이점
  - 데이터와 메소드들이 한 틀로 묶여 있음
  - 메소드가 접근할 수 있는 데이터를 제한시켜 데이터를 보호하거나 숨길 수 있음
  - 문제에 포함된 객체를 정확히 표현 가능

묻고 답하기

Q 묻고 답하기

문 문제를 해결할 때 언제 클래스를 사용하나요?

클래스 – 클래스란?

- 클래스 정의

```
class 클래스명:
   데이터 형 멤버명1
        :
   def 메소드명1():
     문장 1
     문장 2
        :
```

- 클래스 정의 예제

```
class Student:
   pass
```

클래스 – 객체 생성

- 객체
  - 클래스를 이용하여 생성됨
- 클래스와 객체 간의 관계
  - 클래스 –붕어빵 틀
  - 객체 - 붕어빵

클래스 객체

[그림 12-2] 클래스와 객체 간의 관계

- 객체 생성 방법

```
a=클래스명()
```

클래스 – 객체 생성

- 학생 클래스에 대한 객체 생성 예제

```
student1 = Student()
student1.name='Kim'
student1.no = '2019-123456'

print("학생이름=", student1.name)
print("학번=", student1.no)
```

수행 결과

```
학생이름= Kim
학번= 2019-123456
```

실습해보기

실습해보기 12-6

또 다른 학생을 student2라는 레퍼런스 명으로 객체를 생성해 보자.

클래스 – 객체 생성

- 데이터 속성을 클래스 멤버로 포함 예제
  - self는 인스턴스 객체 자신을 가리킴

```
class Student:
    name ="서진"
    no = "2019-123456"

    def prStudent(self):
        print("학생이름=", self.name)
        print("학번=", self.no)
student1 = Student()
student1.prStudent()
```

수행 결과

```
학생이름= Kim
학번= 2019-123456
```

묻고 답하기

Q 묻고 답하기

문 호출시 실제로 인자 전달이 없는데 메소드 인자에 self는 꼭 필요한가?

클래스 – 객체 초기화

- 객체 생성시 데이터 속성 초기화
  - 생성자 사용
    - __init__(self)
  - 예제

```
class Student:
    def __init__(self):
        self.name ="서진"
        self.no = "2019-123456"

    def prStudent(self):
        print("학생이름=", self.name)
        print("학번=", self.no)

student1 = Student()
student2 = Student()
```

```
student1.prStudent()
student2.prStudent()

student1.name ="준성"
student1.no ="2019-456789"

student1.prStudent()
```

수행 결과

```
학생이름= 서진
학번= 2019-123456
학생이름= 서진
학번= 2019-123456
학생이름= 준성
학번= 2019-456789
```

클래스 – 객체 초기화

- 객체 생성시 인자 전달 가능
- student1과 student2 인자 전달로 객체 생성

```
class Student:
    def __init__(self, name, studentNo):
        self.name = name
        self.no = studentNo

    def prStudent(self):
        print("학생이름=", self.name)
        print("학번=", self.no)

student1 = Student("준성", "2019-456789")
student1.prStudent()
```

수행 결과

```
학생이름= 준성
학번= 2019-456789
```

실습해보기

실습해보기 12-7

위의 프로그램에 다른 학생을 한 명 더 추가해 보자.

예외 처리

- 프로그램 수행 중에 에러 또는 예외 상황 발생할 수 있음
  - 예를 들면 0으로 나누는 경우
- 파이썬은 예외 상황 처리를 위해 try문 제공

```
try:
   예외 상황 발생 가능 코드

except [예외 유형]:
   try 절 수행 중에 지정된 유형의 예외 상황이 발생했을 경우 수행되는
   예외 처리 코드

else:
   try 절 수행 중에 예외 상황이 발생하지 않았을 경우 수행되는 코드
```

예외 처리

- 다음 예제는 0으로 나눈 경우 Zero-Division Error 이벤트 발생
- 예외 이벤트가 발생하지 않으면 else 절이 실행됨

```
try:
    dividend= int(input("피젯수를 입력하세요"))
    divisor = int(input("젯수를 입력하세요"))

    result = dividend/divisor

except ZeroDivisionError:

    print("피젯수가 0이므로 나눌 수 없습니다.")

except ValueError:
    print("피젯수나 젯수가 숫자가 아닙니다.")

else:
    print(result)
```

수행 결과 1

```
피젯수를 입력하세요7
젯수를 입력하세요x
피젯수나 젯수가 숫자가 아닙니다.
```

수행 결과 2

```
피젯수를 입력하세요7
젯수를 입력하세요0
피젯수가 0이므로 나눌 수 없습니다.
```

예외 처리

- 예외 처리 문법 구조

```
try:
   예외 상황 발생 가능 코드

except [예외 유형]:
   try 절 수행 중에 지정된 유형의 예외 상황이 발생했을 경우 수행되는
   예외 처리 코드

except [예외 유형1], [예외 유형2], ....:
   try 절 수행 중에 나열된 유형 중 어느 한 유형의
   예외 상황이 발생했을 경우 수행되는 예외 처리 코드

except [예외 유형] as 변수명:
   try 절 수행 중에 지정된 유형의 예외 상황이 발생했을 경우 수행되는
   예외 처리 코드

else:
   try 절 수행 중에 예외 상황이 발생하지 않았을 경우 수행되는 코드

finally:
   예외 상황 발생 여부와 상관없이 try 문 수행 완료 직전에 반드시 수행되는 코드
```