x

K-Digital Training 스마트 팩토리 3기

클래스

클래스

- 클래스는 자료형을 위한 일종의 템플릿
- 파이썬에서 객체 지향 프로그래밍을 사용하기 위한 방법
  - 객체지향 프로그래밍(OOP : Object-Oriented Programming)
  - 프로그램을 객체들 간의 소통으로 바라보는 것
- 객체란? 속성값과 행동을 가지고 있는 데이터
  - 파이썬의 모든 것들 (숫자, 문자, 함수 등)은 여러 속성과 행동을 가진 데이터
  - Ex) 자동차는 바퀴 4개, 색깔 등의 속성과 전진, 후진 등의 행동을 가진 객체

클래스와 객체

- 클래스
  - 새로운 형식을 정의
  - 객체를 만드는 틀 (ex. 붕어빵 틀)
- 객체
  - 클래스로부터 만들어진 것
  - 인스턴스 라고도 함 (ex. 붕어빵)
- 하나의 클래스로 여러 개의 객체를 생성함
- 각각의 객체는 서로 영향을 주지 않음

클래스 선언

- class 클래스 이름으로 선언
- 변수 = 클래스 이름()로 호출, 인스턴스 생성

```
class 클래스이름:
  # 코드

# 호출 시
변수 = 클래스이름()

class TestClass:
  pass # pass 는 말 그대로 아무것도 안 하고 넘어가겠다는 뜻. class를 아예 비워놓으면 에러!

test_instance = TestClass()
```

클래스 변수 (필드, Field)

- 필드는 클래스를 정의할 때 정의되는 변수
- 각각의 인스턴스에서 변수를 사용할 수 있다

```
class Movie:
    title = "BossBaby"

movie1 = Movie()
movie2 = Movie()

print( movie1.title ) # 결과 : BossBaby
print( movie2.title ) # 결과 : BossBaby
```

클래스 함수 (메소드, Method)

- 메소드는 클래스 안에 정의된 함수
- 메소드 선언 시 첫 번째 매개변수로 self 필수 작성
  - self는 자기 자신을 뜻함
  - 클래스를 인스턴스화 하지 않고 호출해서 사용하는 경우에는 필요 없음
  - 호출할 때 self에 값을 직접 넘겨주지 않음 (자동 할당)
  - self는 키워드가 아닌 식별자 즉 다른 이름을 사용할 수 있지만,
    현재 객체 자신을 의미하기 때문에 대부분의 개발자가 self 사용

클래스 함수 (메소드, Method)

```python
class Movie:
    #field
    name: ''

    #method
    def print_msg(msg):
        print(msg)

    # self 사용, 메서드 선언
    def modify(self, movie):
        self.name = movie
```

```python
movie1 = Movie()
movie2 = Movie()

#메서드 호출
Movie.print_msg('print하기')
movie1.modify('name1')
movie2.modify('name2')

print(movie1.name)
print(movie2.name)
```

```
print하기
name1
name2
```

생성자 (Constructor)

- 클래스를 호출할 때 가장 먼저 실행되는 부분
- 클래스 호출과 동시에 자동으로 함수를 실행시키거나
  무언가 동작을 하고 싶을 때 생성자에 정의한다
- 생성자 정의는 메서드 정의와 동일하게 하며 메서드 이름을
  __init__ (언더바 두개)으로 지정한다

```
class Movie:
  def __init__(self):
    print( "Hello I'm Movie Class." )

movie = Movie()
```

매개변수가 있는 생성자

```
class Movie:
    count = 0

    def __init__(self, title, audience):
        self.title = title
        self.audience = audience


movie1 = Movie("bossbaby", 100)
movie2 = Movie("bossbaby2", 200)

print(movie1.title, movie1.audience)
print(movie2.title, movie2.audience)
```

```
BossBaby 100
BossBaby2 200
```

클래스 변수 vs 인스턴스 변수

- 인스턴스 변수 : 인스턴스 공간 내에 존재,
  인스턴스를 생성해야 사용할 수 있음
- 클래스 변수 : 클래스 공간 내에 존재,
  메서드 안에서 클래스이름.변수명으로 접근
- 클래스 변수는 인스턴스 변수와 다르게
  모든 인스턴스가 다 같은 값을 갖는다
  => 공유해야 하는 값을 클래스 변수로 사용

```
class Movie:
    count = 0

    def __init__(self, title, audience):
        self.title = title
        self.audience = audience
        Movie.count += 1


movie1 = Movie("bossbaby", 100)
movie2 = Movie("bossbaby2", 200)

print(movie1.count)
print(movie2.count)
print(Movie.count)
```

접근제어

파이썬은 기본이 public임!

Public: 어디서나 접근 가능

Private: 해당 클래스 내에서만 접근 가능

해당 멤버 앞에 __(double underscore)를 붙여서 표시

Protected: 보통은 해당 클래스 & 하위 클래스 내에서만 접근 가능 이라는 의미

하지만! 파이썬에서는 해당 멤버의 앞에 _(single underscore) 를 붙여서 표시 하며, 실제 제약되지는 않고 일종의 경고 표시로 사용됨

접근제어

```
class Movie:
    count = 0

    def __init__(self, title, audience):
        self.__title = title
        self.__audience = audience
        Movie.count += 1

    def get_title(self):
        return self.__title

    def set_title(self, title):
        self.__title = title

movie1 = Movie("bossbaby", 100)

# print(movie1.__title) #오류  발생!!
print(movie1.get_title())
print(Movie.count)
```

```
class Movie:
    count = 0

    def __init__(self, title, audience):
        self.__title = title
        self._audience = audience
        Movie.count += 1

    def get_title(self):
        return self.__title

    def set_title(self, title):
        self.__title = title

    def get_audience(self):
        return self._audience

    def set_audience(self, audience):
        self._audience = audience

movie1 = Movie("bossbaby", 100)

print(movie1._audience) #오류가 발생하지 않음. 다른 언어와의 차이점.
print(movie1.get_audience())
```

클래스 상속

클래스 상속

- 기존의 클래스가 가지고 있는 필드와 메서드를 그대로 물려받는 새로운 클래스를 만드는 것
- 새로운 클래스에서 필드, 메서드 추가 가능
- 공통점을 가진 내용을 상위 클래스에 둬서 일관되고 효율적인 코딩 가능

클래스 상속

- 부모(슈퍼) 클래스 , 자식(서브) 클래스
- ex) 국가 클래스 (부모), 한국,일본,중국 (자식)
- 자식 클래스를 선언할 때 소괄호로 부모 클래스 포함시키기
- super() : 자식 클래스에서 부모 클래스의 값 사용할 때 쓰는 키워드

```
class Parent:
  # Parent 클래스 코드

class Child(Parent):
  # Child 클래스 코드
```

클래스 상속

```
class Country:
    def __init__(self):
        self.name = "나라이름"
        self.population = "인구"
        self.capital = "수도"

    def show(self):
        print('국가 클래스의 메소드입니다')


class Korea(Country):
    def __init__(self, name):
        self.name = name

    def show_name(self):
        print('국가 이름은 : ', self.name)

country = Korea("대한민국")
country.show()
print(country.name)
country.show_name()
```

```
국가 클래스의 메소드입니다.
대한민국
국가 이름은 :  대한민국
```

메서드 오버라이딩

```
class Country:
    def __init__(self):
        self.name = "나라이름"
        self.population = "인구"
        self.capital = "수도"

    def show_detail(self):
        print('국가 클래스의 메소드입니다.')


class Korea(Country):
    def __init__(self, name, population, capital):
        self.name = name
        self.population = population
        self.capital = capital

    def show_detail(self):
        print(f"국가의 이름은 {self.name},  인구는 {self.population}, 수도는 {self.capital} 입니다." )

country = Korea("대한민국", "5천만", "서울")
country.show_detail()
```

- 부모 클래스의 메서드를
  자식 클래스에서 재정의 하는 것