클래스

- 변수나 함수들을 모아서 필요할 때 사용할 수 있는 집합체

인스턴스

- 클래스에 의해 만들어진 객체
- 인스턴스는 각자 자신의 값을 가지고 있음

모델링

- 클래스로 현실의 개념을 표현

메소드

- 함수와 비슷한 개념
- 클래스에 묶여서 클래스 안에 있는 인스턴스와 관계되는 함수

[실습]

1. 클래스 선언
(변수나 함수들을 모아 필요할 때 사용하는 집합체)

```
#사람이라는 클래스를 작성
class Human():
    '''이야 사람이다'''
```

2. 인스턴스 생성
(클래스에 의해 만들어진 객체)

```
    #Human이라는 클래스의 인스턴스로 person1, person2 생성
person1 = Human()#즉 Human이라는 클래스에 person1이라는 인스턴스가 생성됨
person2 = Human()#즉 Human이라는 클래스에 person2라는 인스턴스가 생성됨
```

3. 클래스의 특성 만들기

```
#클래스 생성#
class alphago():
    '''나는 판별하는 알파고다'''
#클래스 생성 끝#

##인스턴스 생성##
choice1 = alphago()
choice2 = alphago()
##인스턴스 생성 끝##

###클래스 특성 만들기###
choice1.what = '강아지'
choice2.what = '사람'

choice1.speak = '말해'
choice2.speak = '짖어'

def rst(choice):
    print('엄마!! 저기 {}이(가) {}요!!'.format(choice.what, choice.speak))
alphago.rst = rst

choice1.rst()
choice2.rst()
###클래스 특성 만들기 끝###
```

4. 클래스 특성 만들기 실행

```
엄마!! 저기 강아지이(가) 말해요!!
엄마!! 저기 사람이(가) 짖어요!!
```

5. 모델링
(클래스의 개념을 현실에 표현)

```
#사람이라는 클래스를 작성
class Human():
```

```
    '''!사람 클래스 생성!'''


'''사람 만들기 함수'''
def create_Human(name, age): #사람의 이름과 나이를 넣어 만든다
    person = Human()    #Human클래스를 person에 넣어 사용!. @@인스턴스 생성@@
    person.name = name        #class의 이름은 name로 사용!
    person.age = age          #class의 나이는 age로 사용!
    return person             #함수를 끝내고 person으로 돌아간다!
#이 함수(def)는 인간의 형태를 만드는데 위에 선언한 클래스를 사용하여 이름과
나이를 집어넣는다는 뜻이다.

Human.create = create_Human
#클래스를.만든다 = 함수를 통해서
#즉, create_Human(함수에서 만든 내용을) 클래스에 집어 넣는다.

###========여기까지가 클래스의 뼈대 만들기 이다.=========###

'''사용자가 값을 집어넣는다.'''
person = Human.create('짱구',10) #person에 짱구, 10이라는 값을 넣어준다.

###========여기까지가 클래스의 살붙이기이다.=========###


①def old(person):    #person에 old 가 들어온다면
    person.age += 1 #1살을 먹이겠다!
    print('{}가 나이를 먹어서 {}살이 되었습니다.'.format(person.name,
                  person.age)) #1살 먹인 것 확인

②def young(person):    #person에 young가 들어오면
    person.age -= 1   #1살을 줄여주겠다!
    print('{}가 과거로 돌아가 {}살이 되었습니다.'.format(person.name,
                               person.age))#1살 줄이기 확인

Human.old = old             #① 클래스에 넣어준다.
Human.young = young         #② 클래스에 넣어준다.
```

person.old() #이렇게 인스턴스에 함수를 호출하는 것을 메소드를 호출한다고 한다.
person.young() #이렇게 인스턴스에 함수를 호출하는 것을 메소드를 호출한다고 한다.
person.old() #이렇게 인스턴스에 함수를 호출하는 것을 메소드를 호출한다고 한다.

6. 모델링 실행결과

```
짱구가 나이를 먹어서 11살이 되었습니다.
짱구가 과거로 돌아가 10살이 되었습니다.
짱구가 나이를 먹어서 11살이 되었습니다.
```

위와 같은 방법은 실행하는 데는 문제가 없으나
빈 클래스를 만들고 함수는 따로 만들고 클래스에 함수를 넣는 것은 번거로우므로
클래스는 메소드를 다루기 위해 제공하는 문법을 사용하여 구현한다.

위 코드에서 사용하는 짱구가 1살 먹고 줄어들고 하는 모델링 예시를 이용한다.

```
#사람이라는 클래스를 작성
class Human():    #클래스 생성
    '''이야 사람이다'''
    def create(name, age): #create라는 이름의 함수로 사람 뼈대만들기!
        person = Human()
        person.name = name
        person.age = age
        return person

    def old(person):
        person.age += 1
        print('{}가 나이를 먹어서 {}살이 되었습니다.'.format(person.name,
                    person.age))

    def young(person):
        person.age -= 1
        print('{}가 과거로 돌아가 {}살이 되었습니다.'.format(person.name,
                    person.age))

#클래스안에 이 모든 것을 넣어준다#
```

```
person = Human.create('짱구',10)
person.old()
person.young()
person.old()
```

```
짱구가 나이를 먹어서 11살이 되었습니다.
짱구가 과거로 돌아가 10살이 되었습니다.
짱구가 나이를 먹어서 11살이 되었습니다.
```

클래스안에 함수들만 넣어준 것 빼고는 달라진 코드가 없는데 훨씬 사용하기 쉬워졌다.

```
class Family():
    def father(self):
        print("아빠")

    def mother(self):
        print("엄마")

    def son(self):
        print("아들")

class Face():
    def handsome(self):
        print("잘생겼다")

    def pretty(self):
        print("이쁘다")

    def fat(self):
        print("뚱뚱하다")

we = Family() #Family클래스를 we에 인스턴스 생성
we.father()
we.mother()
we.son()
```

```
depiction = Face() #Face클래스를 depiction에 인스턴스 생성
depiction.handsome()
depiction.pretty()
depiction.fat()
```

Family와 Face Class 2개를 생성한다.
그리고 각각의 함수(def)를 생성한다.
자신의 값을 가지도록 인스턴스들을 생성해준다.

[출력 결과]

```
아빠
엄마
아들
잘생겼다
이쁘다
뚱뚱하다
```

여기서 상속을 사용해야하는 이유

```
class Family():
    def father(self):
        print("아빠")

    def mother(self):
        print("엄마")

    def son(self):
        print("아들")

class Face():
    def handsome(self):
        print("잘생겼다")

    def pretty(self):
        print("이쁘다")

    def fat(self):
```

```
        print("뚱뚱하다")

we = Family() #Family클래스를 we에 인스턴스 생성
we.father()
we.mother()
we.son()

depiction = Face() #Face클래스를 depiction에 인스턴스 생성
depiction.handsome()
depiction.pretty()
depiction.fat()
```

Family, Face 클래스가 두개 있는데 이 두개는 공간을 차지하므로 이 두개의 공통점을 모아 Introduce(소개)로 묶을 수 있다.

이 두개의 클래스를 특징대로 묶어 사용을 해보면 불필요한 코드를 낭비하는 경우가 줄어든다.

그대로 상속을 받아본다.

1. 상속시켜줄 부모클래스 생성(introduce)
2. 부모클래스에 상속해줄 매소드 추가
3. 부모가 가지고 있는 메소드가 자식이 가지고 있다면 불필요하므로 삭제
4. 자식클래스()안에 부모클래스를 넣어주면 해당 부모클래스에게 상속받겠다는 뜻

```
class introduce(): #부모 클래스
    def father(self):
        print("아빠")

    def handsome(self):
        print("잘생겼다")
```

```
class Family(introduce): #자식클래스(부모클래스) 부모메소드를 상속받겠다
    def mother(self):
        print("엄마")

    def son(self):
        print("아들")

class Face(introduce): #자식클래스(부모클래스) 부모메소드를 상속받겠다
    def pretty(self):
        print("이쁘다")

    def fat(self):
        print("뚱뚱하다")

we = Family() #Family클래스를 we에 인스턴스 생성
we.father()
we.mother()
we.son()

depiction = Face() #Face클래스를 depiction에 인스턴스 생성
depiction.handsome()
depiction.pretty()
depiction.fat()
```

[출력결과]

```
아빠
엄마
아들
잘생겼다
이쁘다
뚱뚱하다
```

자식에겐 아빠, 잘생겼다 메소드가 없지만 부모에게 상속받아 정상적으로 출력되는 모습을 확인할 수 있다.

```
class human:
    eyes = 2
    nose = 1
    mouth = 1
    ears = 2

    def eat(self):
        print('먹고 있다.')

    def sleep(self):
        print('자고 있다.')

    def hear(self):
        print('듣고 있다.')

# man 클래스는 human 클래스를 상속 받는다.
class man(human):
    def walk(self):
        print('걷고 있다.')


kkw = man()

kkw.hear()
kkw.walk()
print(kkw.nose)
print('')

www = human()
www.eat()
www.walk() # 이 부분에서 오류가 생긴다.
# man이 human을 상속받았지, human이 man을 상속받지는 않았다.
```