python

#13

TODAY

- 자료형
- 클래스, 인스턴스
- 모델링, 메서드
- 상속, 메서드 오버라이드

>>> 자료형 다루기

type 함수와 자료형

```
>>> s = 'Hello, World!'
>>> type(s)
<class 'str'> # 문자열
>>> f = 4.5
>>> type(f)
<class 'float'> # 실수
```

type 함수와 자료형

```
>>> list1 = ['A','B','C']
>>> type(list1)
<class 'list'> # 리스트
>>> list2 = ['a', 'b', 'c']
>>> dict1 = {key: value for key, value in zip(list1, list2)}
>>> type(dict1)
<class 'dict'> # 딕셔너리
```

정수, 실수의 자료형

```
>>> type(42)
<class 'int'>
>>> type(42.0)
<class 'float'>
>>> 42 == 42.0
True
```

자료형 검사하기

```
>>> isinstance(42, int)
True
>>> isinstance(42, float)
False
```

type 함수로 알 수 있는 것

모든 Python 변수에는 자료형이 있다.

예) 문자열, 정수, 실수, 리스트, 딕셔너리, 튜플, ...

자료형은 type 함수로 알아낼 수 있다.

자료형 확인은 isinstance 함수로 가능하다.

그렇다면 인스턴스(instance)란 무엇일까?

>>> 인스턴스

인스턴스

```
>>> list1 = [1, 2, 3]
>>> list2 = ['one', 'two', 'three']
>>> isinstance(list1, list) # True
>>> isinstance(list2, list) # True
>>> list1 == list2 # False
```

인스턴스

list1, list2는 리스트(list)다.

list1, list2는 리스트(list) 클래스의 인스턴스이다.

그렇다고 list1, list2가 서로 같은 건 아니다.

둘 다 서로 다른 리스트(list) 클래스의 인스턴스

하지만 모두 리스트(list) 클래스의 인스턴스이다.

변수와 인스턴스

```
>>> list1 = [ ]
>>> list2 = list1 # list1, list2는 같은 인스턴스
>>> list3 = [ ] # list3은 다른 인스턴스
>>> list1.append(1)
>>> list2.append(2)
>>> list3.append(3)
```

변수와 인스턴스

변수: 3개

list1, list2, list3

인스턴스: 2개

list1=list2, list3

클래스: 1개

리스트(list)

변수와 인스턴스

```
>>> list4 = [1, 2, 3]
>>> list5 = [1, 2, 3]
>>> list4 == list5 # 같은 값을 가지는지 확인
True
>>> list4 is list5 # 같은 인스턴스인지 확인
False
```

변수와 인스턴스

변수: 2개

list4, list5

인스턴스: 2개

list4, list5

클래스: 1개

리스트(list)

>>> 클래스

나만의 클래스 만들기

```
class Human():
    '''사람'''
person1 = Human()
person2 = Human()
person1.language = '한국어'
person2.language = 'English'
```

나만의 클래스 만들기

```
# continued

print(person1.language) # 한국어
print(person2.language) # English

person1.name = '서울시민'

person2.name = '인도인'
```

나만의 클래스 만들기

```
# continued

def speak(person):
    print('{}이 {}로 말을 합니다'.format(person.name, person.language))

speak(person1)

speak(person2)
```

나만의 클래스 만들기

```
# continued
Human.speak = speak # 클래스에 담기
person1.speak() # 클래스의 인스턴스도 사용가능
person2.speak()
```

>>> 모델링

모델링

코드로 현실의 개념을 표현하는 것을 '모델링'이라 함

여기서는 Human 클래스를 예시로 들어서

create, walk, eat 함수로 현실의 개념을 표현함

모델링

```
class Human():
    '''인간'''
person = Human()
person.name = '철수'
person.weight = 60.5
```

모델링

```
# continued

def create_human(name, weight):
    person = Human()

    person.name = name

    person.weight = weight

    return person
```

모델링

```
# continued
Human.create = create_human

person = Human.create('철수', 60.5)
```

>>> 메서드

메서드

클래스 안에 함수를 적는 것

보통 클래스 하나에 메서드가 여러 개 들어감

클래스의 인스턴스 자기 자신을 호출할 수 있음

메서드

```
class Human():

    def create(name, weight):
        person = Human()
        person.name = name
        person.weight = weight
        return person
```

메서드

```
# continued

    def eat(self):

        self.weight += 0.1

        print('{}가 먹어서 {}kg이 되었습니다'.format(self.name, self.weight))

    def walk(self):

        self.weight -= 0.1

        print('{}가 걸어서 {}kg이 되었습니다'.format(self.name, self.weight))
```

메서드

```
person = Human.create('철수', 60.5)

person.eat()
```

 # 매개변수를 넣어주지 않는데 왜 실행이 될까?

메서드

```
# class Human 내부에 추가
    def speak(self, message): # 매개변수 2개
        print(message)
# class Human 외부에 추가
person = Human.create('철수', 60.5)
person.speak('안녕하세요') # 매개변수 1개
```

특수한 메서드

__init__ (초기화)

클래스의 인스턴스를 만드는 순간 자동 호출

__str__ (문자열화)

클래스의 인스턴스 자체를 문자열로 나타낼 때 호출

>>> 상속

상속

```
class Animal():

    def walk(self):
        print('걷는다')
    def eat(self):
        print('먹는다')
```

상속

```
class Human(Animal):
    def wave(self):
        print('손을 흔든다')
class Dog(Animal):
    def wag(self):
        print('꼬리를 흔든다')
```

상속(inheritance)

부모의 속성이 자식에게로 이어짐

현실의 상속: 부모는 상속한 재산이 남아 있지 않음

프로그래밍의 상속: 속성을 옮기는 것이 아닌 '복제'

'부모에게 어떤 유전적 특징을 물려받는다'(inherit)

>>> 메서드 오버라이드

메서드 오버라이드

기본적으로 자식 클래스는 부모 클래스의 메서드를 상속받으나, 그것을 무시하고 같은 이름의 메서드를 만든다면?

자식 클래스에서 부모 클래스에 있는 똑같은 이름의 메서드에 다른 동작을 하는 코드를 만들면, 부모의 동작을 덮어쓰게 됨

메서드 오버라이드를 하지 않은 자식 클래스에서는 부모 클래스의 메서드가 그대로 출력됨

super()

메서드 오버라이드 시, 부모의 동작을 불러옴

```
class Human(Animal):
    def greet(self):
        self.wave()
        super().greet()
```

THANK YOU