SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

객체 비교

== & is

- ==
  - 동등한(equal)
  - 변수가 참조하는 객체가 동등한(내용이 같은) 경우 True
  - 두 객체가 같아 보이지만 실제로 동일한 대상을 가리키고 있다고 확인해 준 것은 아님

- is
  - 동일한(identical)
  - 두 변수가 동일한 객체를 가리키는 경우 True

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

클래스 변수와 인스턴스 변수

클래스 변수와 인스턴스 변수

- 클래스 변수
  - 클래스 정의 안에(+ 인스턴스 메서드 밖에) 선언
  - 특정 클래스 인스턴스에 묶여 있지 않음
  - 클래스 자체의 내용을 저장
  - 같은 클래스에서 생성된 모든 객체는 동일한 클래스 변수를 공유

- 인스턴스 변수
  - 항상 특정 인스턴스에 묶여 있음
  - 클래스에 저장되지 않고 클래스에서 생성된 개별 객체에 저장
  - 인스턴스 마다 완전히 독립적이므로 변수의 값을 수정하면 오로지 해당 객체에만 영향을 미침

클래스 변수와 인스턴스 변수의 함정

- 새 Cat 인스턴스를 만들고 각 인스턴스는 name이라는 인스턴스 변수를 얻음

```
class Cat:
    num_tails = 1               # 클래스 변수

    def __init__(self, name):
        self.name = name        # 인스턴스 변수
```

```
alice = Cat('alice')
james = Cat('james')
print(alice.name)
print(james.name)
```

```
alice
james
```

클래스 변수와 인스턴스 변수의 함정

- 각 Cat 인스턴스 또는 클래스 자체에서 직접 클래스 변수(num_tails)에 접근 가능

```
print(alice.num_tails)
print(james.num_tails)
```

```
1
1
```

```
print(Cat.num_tails)
```

```
1
```

클래스 변수와 인스턴스 변수의 함정

- 그러나 클래스를 통해 인스턴스 변수에 접근할 수 없음
- 인스턴스 변수는 각 객체 인스턴스에 특정되고 __init__ 생성자가 실행될 때 만들어짐
  - 클래스 자체에는 존재하지 않음

```
print(Cat.name)
```

```
---------------------------------------------------------------------
----
AttributeError                            Traceback (most recent call l
ast)
~\AppData\Local\Temp/ipykernel_22212/1900428471.py in <module>
----> 1 print(Cat.name)

AttributeError: type object 'Cat' has no attribute 'name'
```

클래스 변수와 인스턴스 변수의 함정

- 클래스 변수를 수정하고 각 인스턴스가 클래스 변수에 접근해보자

```
Cat.num_tails = 2
```

```
print(alice.num_tails)
print(james.num_tails)
```

```
2
2
```

클래스 변수와 인스턴스 변수의 함정

- 클래스 변수를 다시 원래 값으로 변경
- 인스턴스로 클래스 변수를 다시 변경해보자
- 그런데 클래스 변수가 변경 된 것이 아닌 james 인스턴스에 num_tails 인스턴스 변수가 생겼음

```
Cat.num_tails = 1
```

```
james.num_tails = 2
```

```
print(alice.num_tails)
print(james.num_tails)
print(Cat.num_tails)
```

```
1
2
1
```

클래스 변수와 인스턴스 변수의 함정

- 클래스 변수가 동기화되지 않는 것처럼 보이지만,

  실제로는 클래스 변수와 똑같은 이름의 인스턴스 변수가 만들어졌기 때문

- 이는 잘못된 것은 아니지만 수 많은 버그를 만들어 낼 수 있는 주범이 될 수 있음
- “인스턴스를 통해 클래스 변수를 수정하려고 시도하는 것”
  - 즉, 이름이 같은 인스턴스 변수를 만들어서 원래 클래스 변수를 가리는 것은

    객체 지향 프로그래밍 스타일의 함정이라고 볼 수 있음

클래스 변수와 인스턴스 변수의 함정 – 정리

- 클래스 변수는 모든 클래스 인스턴스에서 공유하는 데이터를 위한 변수
- 인스턴스 변수는 각 인스턴스에 고유한 데이터를 위한 것
- 클래스 변수는 동일한 이름의 인스턴스 변수에 의해 가려질 수 있기 때문에 주의해야 함
  (버그나 원치 않는 동작을 유발할 수 있음)

SAMSUNG SW ACADEMY FOR YOUTH

함께가요 미래로!
Enabling People

인스턴스 / 클래스 / 스태틱 메서드

인스턴스 메서드 / 클래스 메서드 / 스태틱 메서드

- 인스턴스 메서드
  - self 매개 변수를 통해 동일한 객체에 정의된 속성 및 다른 메서드에 자유롭게 접근 가능
  - 뿐만 아니라 클래스 자체에 접근할 수 있음
    - 즉, 인스턴스 메서드가 클래스 상태를 수정할 수도 있음

- 클래스 메서드
  - 클래스를 가리키는 cls 매개 변수를 받음
  - cls 인자에만 접근할 수 있기 때문에 객체 인스턴스 상태를 수정할 수는 없음

인스턴스 메서드 / 클래스 메서드 / 스태틱 메서드

- 스태틱 메서드
  - 임의 개수의 매개 변수를 받을 수 있지만, self나 친 매개 변수는 사용하지 않음
  - 즉, 객체 상태나 클래스 상태를 수정할 수 없음
  - 일반 함수처럼 동작하지만 클래스의 이름공간에 귀속 됨
    - 주로 해당 클래스로 한정하는 용도로 사용

인스턴스 메서드 / 클래스 메서드 / 스태틱 메서드

```
class MyClass:

    def method(self):
        return 'instance method', self

    @classmethod
    def classmethod(cls):
        return 'class method', cls

    @staticmethod
    def staticmethod():
        return 'static method'
```

인스턴스 메서드 / 클래스 메서드 / 스태틱 메서드

- 인스턴스 메서드를 호출한 결과

```
obj = MyClass()
```

```
obj.method()
```

```
('instance method', <__main__.MyClass at 0x185fd086a00>)
```

```
MyClass.method(obj)
```

```
('instance method', <__main__.MyClass at 0x185fd086a00>)
```

인스턴스 메서드 / 클래스 메서드 / 스태틱 메서드

- 인스턴스는 클래스 메서드와 스태틱 메서드 모두 접근할 수 있음

```
obj.classmethod()
```

```
('class method', __main__.MyClass)
```

```
MyClass.classmethod()
```

```
('class method', __main__.MyClass)
```

```
obj.staticmethod()
```

```
'static method'
```

인스턴스 메서드 / 클래스 메서드 / 스태틱 메서드

- 클래스 자체에서 각 타임의 메서드를 호출하는 경우
- 인스턴스 메서드는 호출할 수 없음
  - 파이썬이 self 인자를 채울 수 있는 방법이 없으므로 TypeError 예외 발생

```
MyClass.classmethod()
```

```
('class method', __main__.MyClass)
```

```
MyClass.staticmethod()
```

```
'static method'
```

```
MyClass.method()
```

```
---------------------------------------------------------------------
----
TypeError                                 Traceback (most recent call l
ast)
~\AppData\Local\Temp/ipykernel_22212/715810027.py in <module>
----> 1 MyClass.method()

TypeError: method() missing 1 required positional argument: 'self'
```

스태틱 메서드는 언제 사용해야 할까?

- 스태틱 메서드는 self, cls 인자를 취하지 않기 때문에 사용에 있어 큰 제약이 있어 보임
    - 하지만, 반대로 특정한 메서드가 주변의 다른 것들과 독립적일 수 있음을 뜻하는 것이기도 함
- 스태틱 메서드와 클래스 메서드를 사용하는 것은 개발자의 의도를 전달하는 동시에

  개발자가 자신의 의도를 강제해 버그로 인해 설계를 깨뜨리지 않도록 함
- self, cls 인자를 전달하지 않기 때문에 객체 인스턴스, 클래스 상태에 접근할 수 없음을 보장
- 또한 일반 함수를 사용하는 것처럼 실행할 수 있기 때문에

  객체 지향 프로그래밍과 절차 지향 프로그래밍 스타일 사이를 연결하는 역할을 하기도 함

인스턴스 메서드 / 클래스 메서드 / 스태틱 메서드 정리

- 인스턴스 메서드는 인스턴스가 필요하며 self를 통해 인스턴스에 접근
- 클래스 메서드는 인스턴스가 필요하지 않음
    - 인스턴스(self)에는 접근할 수 없지만 cls를 통해 클래스 자체에 접근할 수 있음
- 스태틱 메서드는 self, cls에 접근할 수 없으며, 일반 함수처럼 작동하지만

  자신이 정의된 클래스의 이름공간에 속함
- 스태틱 및 클래스 메서드는 클래스 설계에 대한 개발자의 의도를 전달하고 강제함
    - 이러한 점을 통해 코드의 유지 보수를 하는 데 많은 도움을 줄 수 있음