Ask Django

클래스 상속

상속 (Inheritance)

- 코드 중복을 최소화하기 위한 목적으로 등장
- 파이썬 클래스는 최상위 클래스인 **object**를 상속
  - 파이썬2에서는 object상속유무에 따라 old/new style class로 분리
  - 파이썬3에서는 old-style class가 제거
- 클래스 간에 상속관계에 놓이게 되면, 부모/자식 관계가 성립
- 자식 클래스는 부모 클래스의 모든 내역을 물려받음
- 다중상속 지원 : 직계 부모가 다수

상속을 알기 전에는 ...

```
class Doctor:
    def __init__(self, name):
        self.name = name

    def run(self):
        print('뜁니다.')

    def eat(self, food):
        print('{}을 먹습니다.'.format(food))

    def sleep(self):
        print('잠을 잡니다.')

    def study(self):
        print('열심히 공부합니다.')

    def research(self):
        print('열심히 연구합니다.')
```

```
class Programmer:
    def __init__(self, name):
        self.name = name

    def run(self):
        print('뜁니다.')

    def eat(self, food):
        print('{}을 먹습니다.'.format(food))

    def sleep(self):
        print('잠을 잡니다.')

    def study(self):
        print('열심히 공부합니다.')

    def coding(self):
        print('열심히 코딩합니다.')
```

```
class Designer:
    def __init__(self, name):
        self.name = name

    def run(self):
        print('뜁니다.')

    def eat(self, food):
        print('{}을 먹습니다.'.format(food))

    def sleep(self):
        print('잠을 잡니다.')

    def study(self):
        print('열심히 공부합니다.')

    def design(self):
        print('열심히 디자인합니다.')
```

Doctor, Programmer, Designer는 모두 사람

사람과 관련된 코드 중복이 발생

상속으로 해결

사람과 관련된 코드는 모두 Person으로 집결 !!!

```
class Person(object):
    def __init__(self, name):
        self.name = name

    def run(self):
        print('뜁니다.')

    def eat(self, food):
        print('{}을 먹습니다.'.format(food))

    def sleep(self):
        print('잠을 잡니다.')

    def study(self, target):
        print('{}을 열심히 공부합니다.'.format(target))
```

코드가 보다 간결해졌습니다.

상속을 받고, 클래스 개별 코드만 구현합니다.

```
class Doctor(Person):
    def research(self):
        print('열심히 연구합니다.')

class Programmer(Person):
    def coding(self):
        print('열심히 개발합니다.')

class Designer(Person):
    def design(self):
        print('열심히 디자인을 합니다.')
```

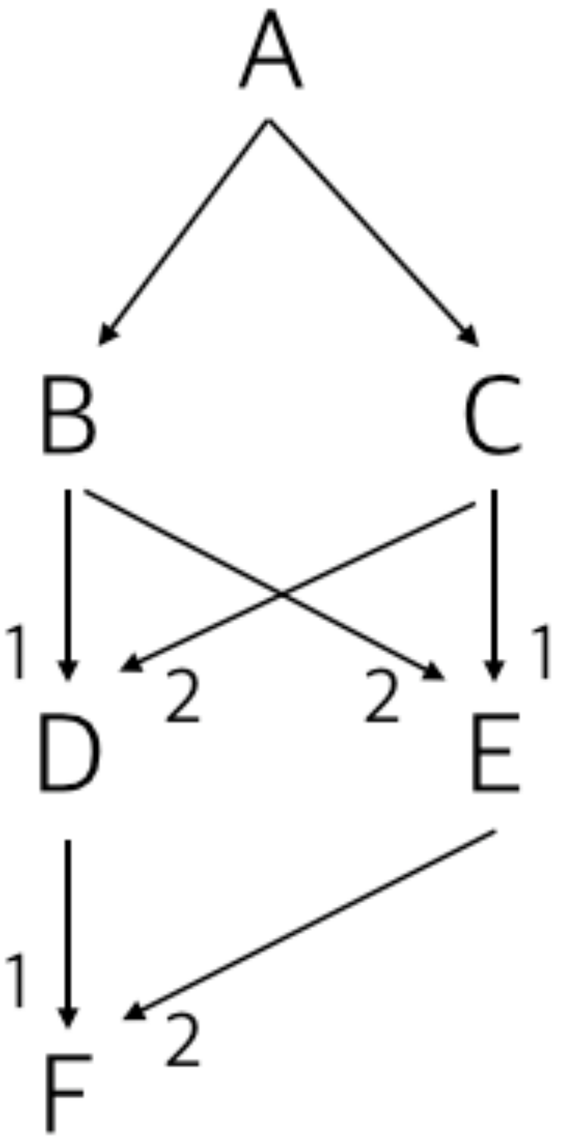
MRO (Method Resolution Order)

- 파이선의 클래스 탐색순서는 MRO 를 따릅니다.
  - Class.**mro** 를 통해 확인 가능
- MRO 가 꼬이도록 클래스를 설계할 수는 없습니다.
  - TypeError: Cannot create a consistent method resolution order (MRO)

잘못된 MRO 예시

F 클래스는 **MRO** TypeError로 인해 정의할 수 없습니다.

```
class A: pass

class B(A): pass

class C(A): pass

class D(B, C): pass

class E(C, B): pass

class F(D, E): pass   # 정의 불가 !!!
```



부모의 함수 호출

- 내장함수 super를 통해 부모의 함수 호출
  - D의 mro순서는 D > B > C > A
  - D().fn() 의 실행결과로서 A, C, B, D 가 출력
- super 호출 시에 MRO에 기반하여 호출

```python
class A:
    def fn(self, arg):
        print('A', arg)

class B(A):
    def fn(self, arg):
        super().fn(arg)
        print('B', arg)

class C(A):
    def fn(self, arg):
        super().fn(arg)
        print('C', arg)

class D(B, C):
    def fn(self, arg):
        super().fn(arg)
        print('D', arg)
# ---
>>> print(D.__mro__)
(__main__.D, __main__.B, __main__.C, __main__.A, object)
>>> print(D().fn('python'))
A python
C python
B python
D python
```