1 delegate

In [16]:

```
class A(object, metaclass =type):
    # 정확한 꼴은 기본적으로 이래야 한다... 부모는 object, 메타클래스는 타입 (둘은 다르다)
    a = 1
    def a(self):
        print('a')  # 함수가 있는데 a = 1 이 이를 뒤엎는다.
    # a =함수라고 이해할 수 있다.
    # a = 1 이었는데, a = 함수로 바뀌었다
        # ; 재할당 (first class 이기 때문에; 함수가 값, 식으로 쓰일 수 있다.)
    # 내부적으로는 a = 함수로 저장하였지만, 이름이 같으니 뒤엎어 버린다.


# object는 안써도 자동으로 상속에 포함시켜진다.
```

In [17]:

```
class B(A): # 상속...이 무엇이냐?
    pass
# 파이썬에서는 B.a가 1이었다.
```

In [18]:

```
B.a
```

Out[18]:

```
<function __main__.A.a(self)>
```

In [19]:

```
id(B.a)
```

Out[19]:

```
1862589985560
```

In [ ]:

```
B.a is A.a

# B의 a와 A의 a가 메모리가 같다. >> 두 개가 똑같다.
```

In [20]:

```
class B(A):
    a = 3
```

In [ ]:

```
B.a

```

1.1 상속 관계를 맺으면, 내가 할 수 없는 것을 부모한테 위임시킨

다.

- (다른 프로그래밍 언어에서는 상속 시 부모의 내용을 위임하지 않는 언어도 있다)
- 상속을 쓰면 최적화된 구조. 내가 할 수 없으면 부탁하기 때문에, 내 자체에서 많이 정해놓지 않으면 부모 것 그냥 사용하면 된다.
- 파이썬에서는 상속을 써도 나오는 단점이 없다.

1.2 다중 상속

In [24]:

```
class A(object):
    a = 1
```

In [27]:

```
class B:
    a = 2
```

In [28]:

```
class C(A,B):
    pass
# 항상 정확하게 순서대로 상속되지 않는다.
# 이는 MetaClass와 관계가 있다.
```

In [32]:

```
# 똑같은 거 실행할 때 어떤 순서 대로 나오는 가
# method resolution order >> 어떤 순서로 실행할지 내부적인 규칙이 존재한다.
C.mro() # 너무 중요...
        #>> 남의 것 가지고 쓸 때 어떤 구조, 상속체계를 가졌는가?를 알고 싶을 때 사용한다.
```

Out[32]:

```
[__main__.C, __main__.A, __main__.B, object]
```

In [33]:

```
C.a
```

Out[33]:

```
1
```

In [45]:

```
dir(C) # .mro() 가 없다...근데 쓸 수 있는데 왜?
# A가 쓸수 있는 애는 자기 자신 뿐만 아니라,
#metaclass에 있는 것을 사용할 수 있다 ( object, metaclass =type 가 자동으로 붙음)
# 그러므로, .mro()사용가능
# 내 metaclass는 type(C)로 구할 수 있다. 이를 dir(type(C))를 하면 .mro() 가 있다.
```

Out[45]:

```
['__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__module__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',
 'a']
```

In [46]:

```
1 dir(type(C))
```

Out[46]:

```
['__abstractmethods__',
 '__base__',
 '__bases__',
 '__basicsize__',
 '__call__',
 '__class__',
 '__delattr__',
 '__dict__',
 '__dictoffset__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__flags__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__instancecheck__',
 '__itemsize__',
 '__le__',
 '__lt__',
 '__module__',
 '__mro__',
 '__name__',
 '__ne__',
 '__new__',
 '__prepare__',
 '__qualname__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasscheck__',
 '__subclasses__',
 '__subclasshook__',
 '__text_signature__',
 '__weakrefoffset__',
 'mro']
```

In [29]:

```
1 C.a
```

Out[29]:

```
1
```

In [38]:

```
import tensorflow as tf
tf.keras.models.Sequential.mro()
# 상속은 내용을 추가하는 것보다 제약을 추가하는 경우가 많다
# metaclass는 클래스 자체의 행동이고, A에 관한 행동이고, 부모는 A의 인스턴스에 대한 행동이다.
# 객체 지향은 기본적으로 class를 사용하는 것이 아니라,
# 파생된 인스턴스를 사용하는 것인데, 그 인스턴스는 부모쪽에서 기능을 물려받음

```

Out[38]:

```
[tensorflow.python.keras.engine.sequential.Sequential,
 tensorflow.python.keras.engine.training.Model,
 tensorflow.python.keras.engine.network.Network,
 tensorflow.python.keras.engine.base_layer.Layer,
 tensorflow.python.module.module.Module,
 tensorflow.python.training.tracking.tracking.AutoTrackable,
 tensorflow.python.training.tracking.base.Trackable,
 object]
```

In [43]:

```
C.__class__ # C의 부모 클래스 type
```

Out[43]:

```
type
```

In [42]:

```
C.__mro__ # mro를 활용하여 C가 위임받은 순서를 알 수 있다.
```

Out[42]:

```
(__main__.C, __main__.A, __main__.B, object)
```

In [44]:

```
1 dir(C)
```

Out[44]:

```
['__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__module__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',
 'a']
```

In [ ]:

```
1 # 메타클래스 metaclass를
```

In [ ]:

```
1 A.a가 정수 ->> 이에 대한 type은 정수의 type이므로, int이다.
```

In [41]:

```
1 a.__class__
```

Out[41]:

```
__main__.A
```

2 왜 mro를 알아야 하나?

- 다중 상속이기 때문에 내 값을 누구한테 위임받았냐를 알고 싶은데 그를 알 수 있는게 mro이다.
- 메소드가 충돌할 때 어떤 순서대로 진행되는지 확인