(15-D) Abstract Class in Python

What is Abstract Class?

- Abstract classes are classes that contain one or more abstract methods.
- An abstract method is a method that is declared, but contains no implementation.
- Abstract classes should not be instantiated, and require subclasses to provide implementations for the abstract methods.

- 상황: 모든 Automobile의 subclass들 (예, 2Door_Auto, 4Door_Auto, Sedan, Truck..)은 모두 Engine_Oil_Check() 이라는 function을 자체적으로 구현하는것을 의무로 하고 싶다

- Python comes with a module which provides the infrastructure for defining Abstract Base Classes (ABCs)

유사한 Class들에 특정 Function을 의무화 [1/2]

```
class BaseClass:
    def func1(self):
            pass
    def func2(self):
            pass
```

Programmer's Intention:
Baseclass에서 파생된 모든 concrete class는 func1, func2를 선언하고 구현되어 있어야 한다

```
class DerivedClass1(BaseClass):
    def func1(self):
        print( "FUNC 1 in Derived1“)

    def func2(self):
        print ("FUNC 2 in Dervied1“)
```

```
class DerivedClass2(BaseClass):
    def func1(self):
        print ("====================“)
        print ( "FUNC 1 in Derived2“)
        print ("====================“)

    def func2(self):
        print ("====================“)
        print ("FUNC 2 in Derived2“)
        print ("====================“)
```

Subclass에서 func1 or func2를 구현안해도 어쩔수 없다!

유사한 Class들에 특정 Function을 의무화 [2/2]

```
class BaseClass:
    def func1(self):
        raise NotImplementedError()
    def func2(self):
        raise NotImplementedError()
```

BaseClass를 instantiation을 하는것을 막을수 없다!

```
class DerivedClass3(BaseClass):
    def func1(self):
        print( "FUNC1 in Derived3“)

    """
    func2 method is not implemented yet..
    """
```

DerivedClass3는 BaseClass 의 func2를 상속받는다!

```
X = DerivedClass3()
x.func1()
x.func2()
```

```
FUNC1 in Dervied3
Traceback (most recent call last):
  File "D.py", line 5, in <module>
    _m.func2()
  File "/home/ubuntu/BaseClass.py", line 6, in func2
    raise NotImplementedError()
NotImplementedError
```

ABC (Abstract Base Class) using abc module

```
from abc import ABC, abstractmethod
class BaseClass (ABC):
    @abstractmethod
    def func1(self):
        pass
    @abstractmethod
    def func2(self):
        pass
```

- 추상화 시키고자 하는 Method에 decorato로 @abstractmethod 를 선언

- ABC를 적용하게 되면, Instantiation도 못하게 하고 BaseClass를 상속받는 모든 파생 클래스에서 해당 Method를 선언해서 구현하지 않으면, Error를 발생시키게 된다.

- This module provides 'Abstract Base Class' to Python
  - It is one of the key features of Object Oriented Programming
- Collections module has some 'concrete classes' that derive from ABCs
  - This can be further derived from itself

ABC vs NotImplementedError

- abc 클래스를 이용하게 되면, 해당 BaseClass 는 인스턴스 생성이 불가

(단지 파생 클래스 구현을 위한 추상화 기능 제공 역할을 하게 될 뿐이다.)

```
 >>> base = BaseClass()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't instantiate abstract class BaseClass with abstract methods func1, func2
```

- 두번째, abc 클래스를 이용하게 될 경우 에러 발생 시점이 다르다.
  - 메서드에 raise를 이용해 NotImplementedError 를 선언해 놓은 경우에는 해당 메서드가 실제로 호출이 되는 시점(runtime)에 에러를 발생
  - abc 를 이용하는 경우에는 해당 모듈이 import 되는 순간부터 에러를 발생시키게 된다. 즉, abc 클래스 경우는 좀더 strict 한 모듈 관리가 가능

Some Relaxations in Python Abstract Class [1/3]

```
from abc import ABC, abstractmethod

class AbstractClassExample(ABC):

    def __init__(self, value):
        self.value = value
        super().__init__()

    @abstractmethod
    def do_something(self):
        pass
```

Python Abstract Class에는 normal function 이 있을수도 있다!
그러나 subclass에서만 사용가능하다

Abstract class에 __init__( )를 만들어도 instantiation은 이루어지지 않는다. 그러나 concrete subclass가 생기면 inherit를 해줄수 있다

```
class DoAdd42(AbstractClassExample):
    pass
x = DoAdd42(4)
```

DoAdd42 class에 아무것도 없으므로 DoAdd42 class는 abstract class

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-9-83fb8cead43d> in <module>()
      2     pass
      3
----> 4 x = DoAdd42(4)
TypeError: Can't instantiate abstract class DoAdd42 with abstract methods do_something
```

Some Relaxations in Python Abstract Class [2/3]

```
class DoAdd42(AbstractClassExample):
    def do_something(self):
        return self.value + 42

class DoMul42(AbstractClassExample):

    def do_something(self):
        return self.value * 42

x = DoAdd42(10)
y = DoMul42(10)
print(x.do_something())
print(y.do_something())
```

```
52
420
```

Abstract class에 파생된 DoAdd42 class와 DoMul42 class는 concrete한 method가 있으므로 concrete class이고, concrete class이므로

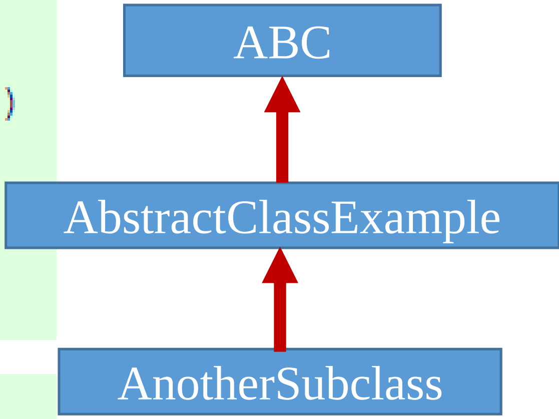
AbstractClassExample의 __init__()를 inherit받는다

Some Relaxations in Python Abstract Class [3/3]

Python Abstract Method에는 normal implementation 이 있을수도 있다!
그러나 subclass에서만 사용가능하다

```
from abc import ABC, abstractmethod

class AbstractClassExample(ABC):

    @abstractmethod
    def do_something(self):
        print("Some implementation!")

class AnotherSubclass(AbstractClassExample):
    def do_something(self):
        super().do_something()
        print("The enrichment from AnotherSubclass")

x = AnotherSubclass()
x.do_something()
```

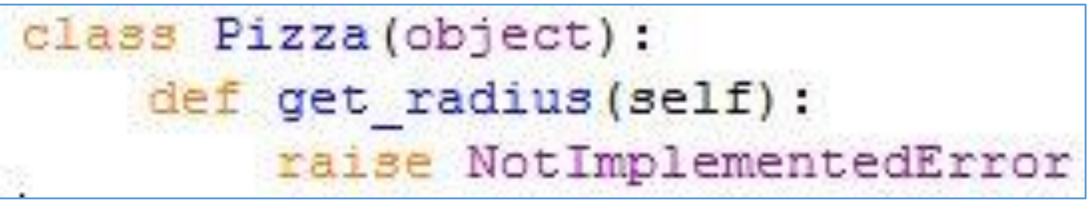
```
Some implementation!
The enrichment from AnotherSubclass
```



An abstract method in Python is a method defined in base class, which may not provide any implementation (즉, implementaion을 가져도 된다)

Another Motivational Example [1/2]

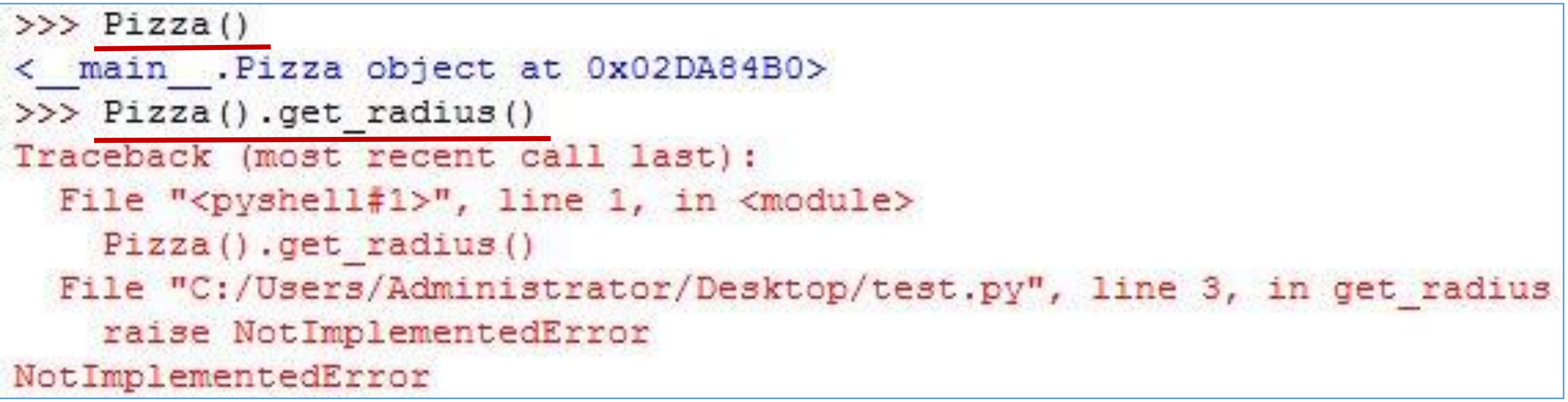
Programmer's Intention: Pizza class에서 파생된 모든 concrete class는 get_radius라는 method를 선언하고 구현되어 있어야 한다



```
class Pizza(object):
    def get_radius(self):
        raise NotImplementedError
```

abc module을 안쓰고
NotImplmentedError 를 쓰는 경우

Result



```
>>> Pizza()
<__main__.Pizza object at 0x02DA84B0>
>>> Pizza().get_radius()
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    Pizza().get_radius()
  File "C:/Users/Administrator/Desktop/test.py", line 3, in get_radius
    raise NotImplementedError
NotImplementedError
```

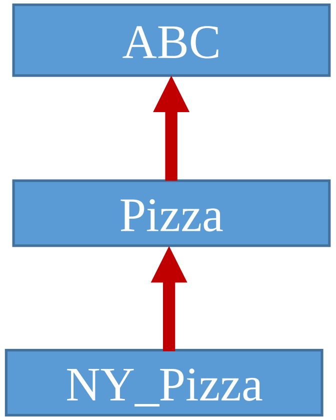
- Any class inheriting from Pizza should implement and override the 'get_radius' method, otherwise exception would be made
  - If you write a class that inherits from Pizza and forget to implement 'get_radius', an error would be raised if you use it

Another Motivational Example [2/2]

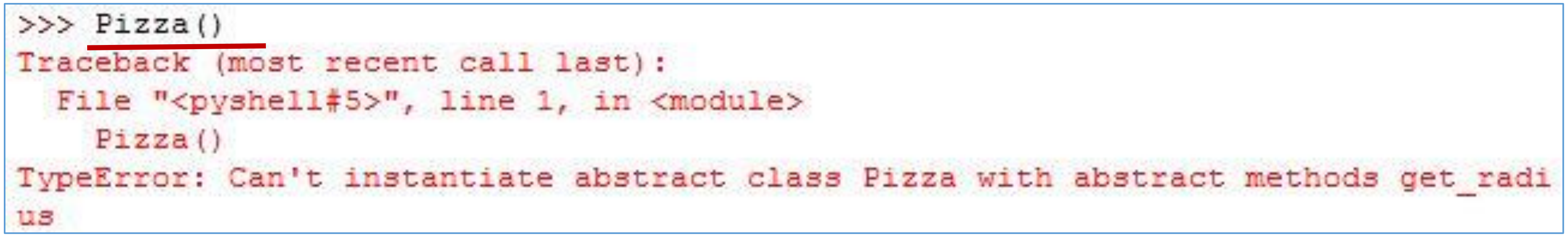
- There is a way to trigger this error earlier, by using @abstractmethod

abc module를 쓰는 경우

```
>>> from abc import ABC, abstractmethod
>>> class Pizza(ABC):
        @abstractmethod
        def get_radius(self):
            print("This will not appear on print")
        pass
>>>
```



Result



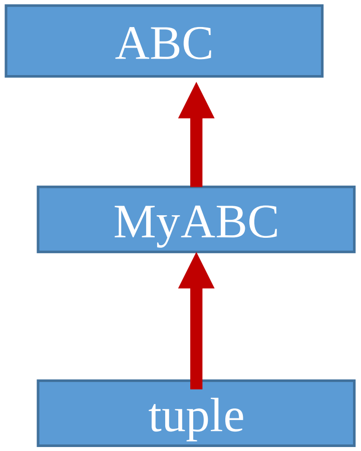
```
>>> Pizza()
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    Pizza()
TypeError: Can't instantiate abstract class Pizza with abstract methods get_radi
us
```

- Using abc and its special class, as you try to use Pizza or any class inheriting from it, you'll get an error in the stage of initiation

Registering a class as a subclass of Abstract Class

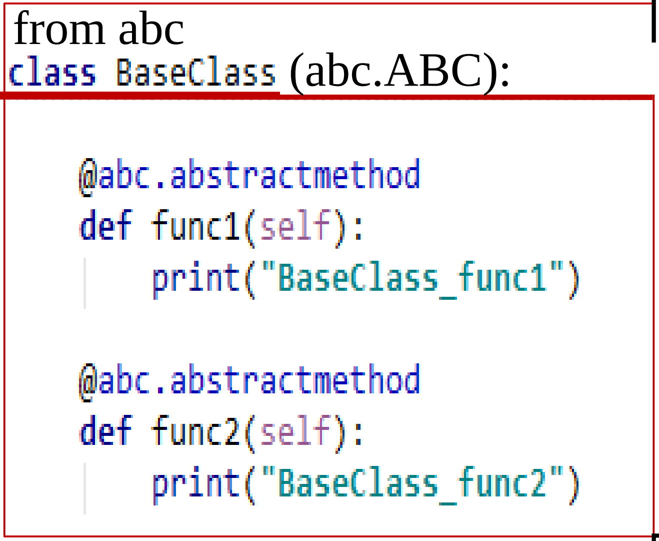
- **register**(subclass) : Register subclass as a ‘virtual subclass’ of ABC abstract class
- **issubclass**(subclass, superclass)
- **isinstance**(object, class)

```
>>> from abc import ABC, abstractmethod
>>> class MyABC(ABC)
            pass
>>> MyABC.register(tuple)
>>> issubclass(tuple, MyABC)
True
>>> isinstance((3,4), MyABC)
True
```

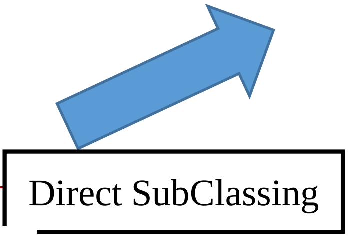


Subclass from Abstract Class

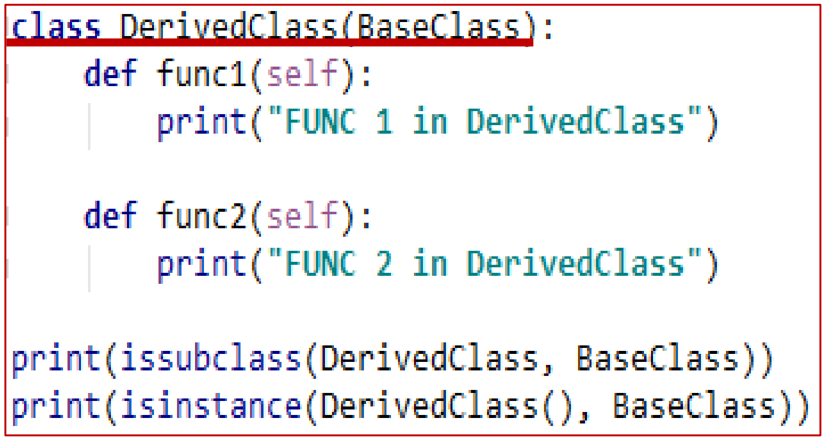
Marking methods of the base class as abstract, and then registering concrete classes as implementations of the abstract base



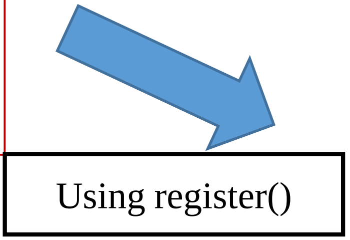
```
from abc
class BaseClass (abc.ABC):

    @abc.abstractmethod
    def func1(self):
        print("BaseClass_func1")

    @abc.abstractmethod
    def func2(self):
        print("BaseClass_func2")
```

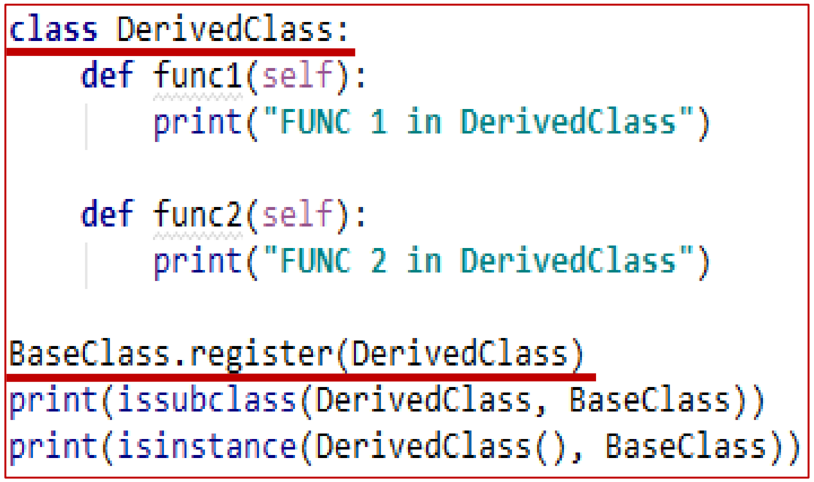


from abc



```
class DerivedClass(BaseClass):
    def func1(self):
        print("FUNC 1 in DerivedClass")

    def func2(self):
        print("FUNC 2 in DerivedClass")

print(issubclass(DerivedClass, BaseClass))
print(isinstance(DerivedClass(), BaseClass))
```

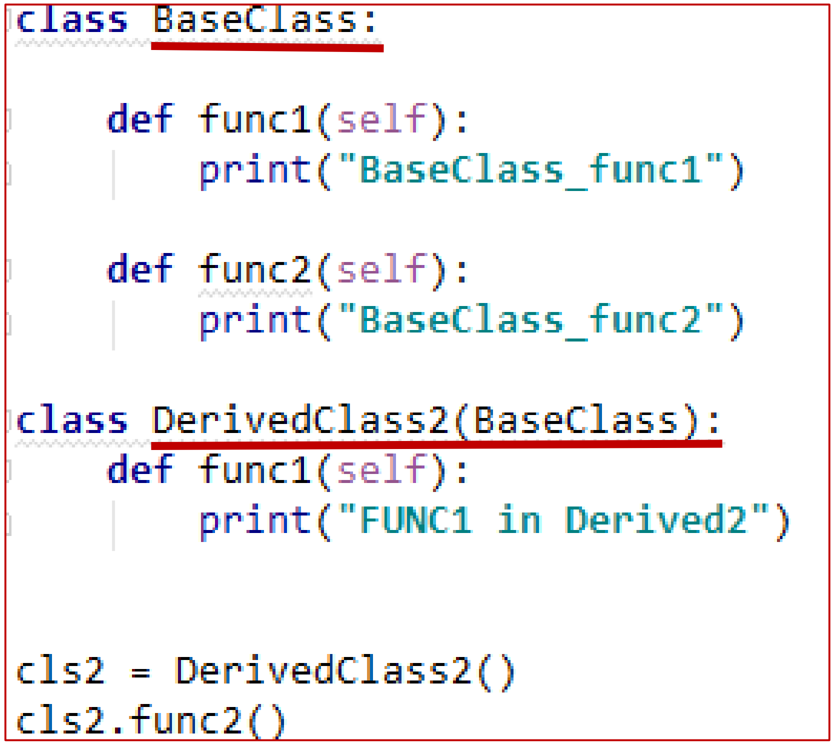




```
class DerivedClass:
    def func1(self):
        print("FUNC 1 in DerivedClass")

    def func2(self):
        print("FUNC 2 in DerivedClass")

BaseClass.register(DerivedClass)
print(issubclass(DerivedClass, BaseClass))
print(isinstance(DerivedClass(), BaseClass))
```

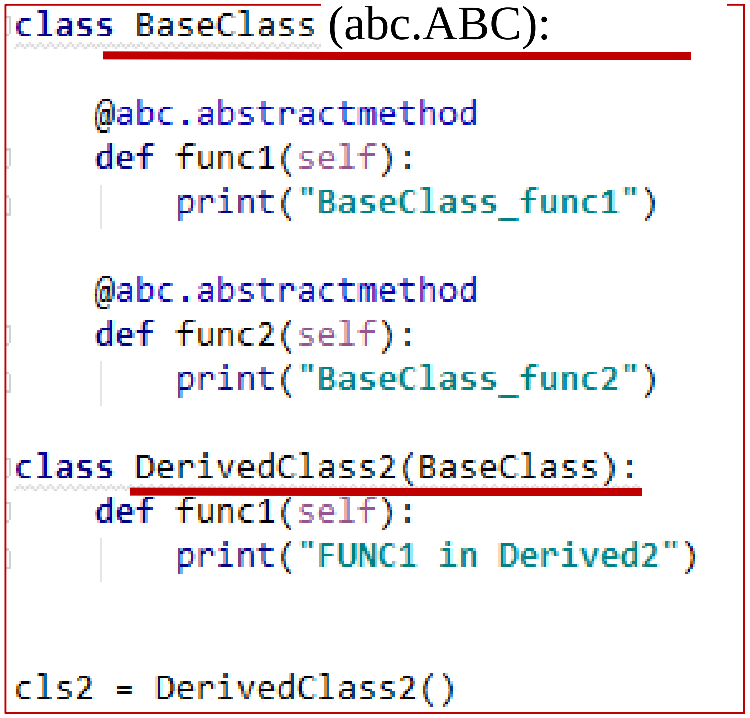
- **The abstraction class can not be instantiated**
- **Eg) cls = BaseClass() : error**

Early Detection of Errors in ABC



```python
class BaseClass:

    def func1(self):
        print("BaseClass_func1")

    def func2(self):
        print("BaseClass_func2")

class DerivedClass2(BaseClass):
    def func1(self):
        print("FUNC1 in Derived2")


cls2 = DerivedClass2()
cls2.func2()
```

“BaseClass_func2”



```python
class BaseClass (abc.ABC):

    @abc.abstractmethod
    def func1(self):
        print("BaseClass_func1")

    @abc.abstractmethod
    def func2(self):
        print("BaseClass_func2")

class DerivedClass2(BaseClass):
    def func1(self):
        print("FUNC1 in Derived2")


cls2 = DerivedClass2()
```

TypeError: Can't instantiate abstract class DerivedClass2 with abstract methods func2

- ABC causes an TypeError at the moment of importing module
  - Cf) Using raise method : raising error at run time
  - Only when directly subclassing