

import, from, as

다음과 같이 `my_module.py`라는 파이썬 파일을 생성해본다.

```
In [1]: %%writefile my_module.py

# my_module.py

_value = 1

class MyClass:
    _num = 10

    def __init__(self, value, **kwargs):
        self.value = value
```

Writing my_module.py

모듈을 불러올 때는 `import` 키워드를 통해 불러온다.

```
In [2]: import my_module
```

```
In [3]: import inspect
```

```
In [4]: print(inspect.getsource(my_module))

# my_module.py

_value = 1

class MyClass:
    _num = 10

    def __init__(self, value, **kwargs):
        self.value = value
```

접근 연산자 `dot(.)`을 통해 모듈 안에 있는 `MyClass`를 불러온다.

```
In [5]: my_class = my_module.MyClass(10)
```

```
In [6]: my_class.value
```

```
Out[6]: 10
```

모듈안에 포함된 식별자나 함수 앞에 언더스코어(`_`)로 선언된 식별자는 `import` 되지 않는다.

```
In [7]: dir(my_module)
```

```
Out[7]: ['MyClass',
         '_builtins_',
         '_cached_',
         '_doc_',
         '_file_',
         '_loader_',
         '_name_',
         '_package_',
         '_spec_',
         '_value']
```

`from` 키워드를 통해서 `my_module` 모듈안에 있는 `MyClass`만 따로 불러온다.

```
In [8]: from my_module import MyClass
```

```
In [9]: my_class = MyClass(20)
```

```
In [10]: my_class.value
```

```
Out[10]: 20
```

`as` 키워드는 별칭을 붙이는 `alias`를 의미한다.

```
In [11]: import my_module as mm
```

```
In [12]: mm.MyClass
```

```
Out[12]: my_module.MyClass
```

언더스코어(`_`)

앞서 살펴봤듯, 파이썬에서 언더스코어(`underscore`, `_`)는 조금 특별한 의미를 가진다.

아래처럼 언더스코어를 넣어 숫자를 리터럴로 선언 시 단위별로 구분하여 선언하는 것도 가능하다.

```
In [13]: 100_000_000
```

```
Out[13]: 100000000
```

```
In [14]: 1_0000_0000
```

```
Out[14]: 100000000
```

파이썬(pythonic)하게 아래처럼 필요없는 값을 무시하고자 할 때 사용하기도 한다.

```
In [15]: a, *_ , b = [1, 2, 3, 4, 5]
```

```
In [16]: a
```

```
Out[16]: 1
```

```
In [17]: b
```

```
Out[17]: 5
```

```
In [18]: _
```

```
Out[18]: [2, 3, 4]
```

```
In [19]: data = zip([1,2,3,4,5], [6,7,8,9,10])
```

```
In [20]: for i, (_, j) in enumerate(data):  
         print(i, j)
```

```
0 6  
1 7  
2 8  
3 9  
4 10
```

파이썬 인터프리터에선 마지막으로 실행된 결과값이 _ 라는 식별자에 저장된다.

```
PS C:\Users\zmfls> python  
Python 3.6.8 |Anaconda, Inc.| (default, Feb 21 2019, 18:30:04) [MSC v.1916 64 bit (AMD64)] on win32  
Type "help", "copyright", "credits" or "license" for more information.  
>>> '안녕'  
'안녕'  
>>>  
'안녕'  
>>> 10000  
10000  
>>> _  
10000  
>>> █
```

먼저, 새로운 클래스 Test를 정의하였다.

```
In [21]: class Test:  
         a = 10  
         _b = 20  
         __c = 30  
  
         def test(self):  
             print('call test()')  
  
         def _f(self):  
             print('call _f()')  
  
         def __func(self):  
             print('call __func()')
```

```
In [22]: test = Test()
```

PEP8에 의하면 언더스코어는 클래스 혹은 모듈 안에서만 사용할 수 있도록 외부에서 접근을 하지 못하도록(=캡슐화) 하는 접근 지정자 private와 같은 용도로써 사용하도록 권장하는 방식이다. 다만, 다른 객체지향 언어들과 같이 해당 멤버 혹은 메소드에 접근하지 못하는 것은 아니다.

언더스코어를 붙인 멤버 혹은 메소드는 그대로 호출이 가능하다. 그렇지만 test. 까지 입력하고 자동 완성키 Tab을 눌러보면 언더스코어(_)가 붙은 멤버 혹은 메소드는 보이지 않는다.

```
In [23]: test._b
```

```
Out[23]: 20
```

```
In [24]: test._f()
```

```
call _f()
```

맹글링 (Mangling)

파이썬에서 맹글링은 코드에 선언된 식별자명 혹은 함수명을 인터프리터에서 한번 변형하는 것을 의미한다.

```
In [25]: Test.__c
```

```
-----
AttributeError                                Traceback (most recent call last)
<ipython-input-25-a126715e3230> in <module>
----> 1 Test.__c

AttributeError: type object 'Test' has no attribute '__c'
```

```
In [26]: test.__func()
```

```
-----
AttributeError                                Traceback (most recent call last)
<ipython-input-26-c4ca12aa8d4a> in <module>
----> 1 test.__func()

AttributeError: 'Test' object has no attribute '__func'
```

dir 함수를 통해 클래스에 정의된 멤버와 메소드를 살펴보면 클래스 안에 멤버명 혹은 메소드명 앞에 더블 언더스코어(__)를 붙이면 다른 이름으로 망글링(Mangling) 되어진 것을 확인할 수 있다.

```
In [27]: dir(test)
```

```
Out[27]: ['_Test_c',
'_Test_func',
'_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__',
'_b',
'_f',
'_a',
'test']
```

```
In [28]: test._Test_c
```

```
Out[28]: 30
```

```
In [29]: test._Test_func()
```

```
call __func()
```