

모듈과 패키지

모듈

❖ 모듈

- 일반적으로 "독자적인 기능을 갖는 구성 요소"를 의미
- 파이썬에서는 개별 소스 파일을 일컫는 말

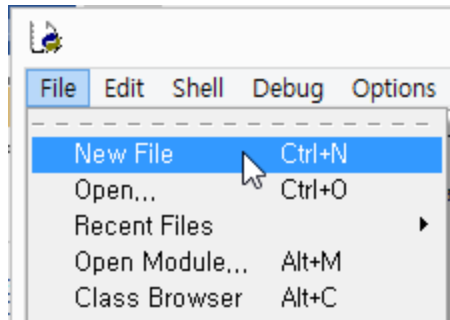
❖ 표준 모듈 : 파이썬과 함께 따라오는 모듈

❖ 사용자 생성 모듈 : 프로그래머가 직접 작성한 모듈

❖ 서드 파티(3rd Party) 모듈 : 파이썬 재단도 프로그래머도 아닌 다른 프로그래머, 또는 업체에서 제공한 모듈

모듈 - 두 개의 소스 파일로 만드는 하나의 프로그램 예제

❖ IDLE을 실행한 후 [File]→[New File] 메뉴항목을 선택하여 편집창 실행



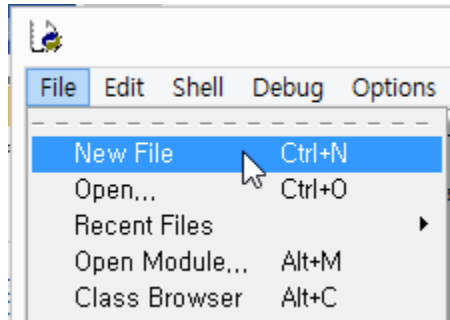
IDLE 편집창에서 [File]→[Save] 메뉴항목을 선택하고, 디렉토리를 하나 골라 그곳에 "calculator.py"라는 이름으로 모듈을 저장

❖ 예제 : 08/calculator.py

```
def plus(a, b):  
    return a+b  
  
def minus(a, b):  
    return a-b  
  
def multiply(a, b):  
    return a*b;  
  
def divide(a, b):  
    return a/b
```

모듈 - 두 개의 소스 파일로 만드는 하나의 프로그램 예제

❖ IDLE을 실행한 후 [File]→[New File] 메뉴항목을 선택하여 편집창 실행



IDLE 편집창에서 [File]→[Save] 메뉴항목을 선택하고, 디렉토리를 하나 골라 그곳에 "calc_tester.py"라는 이름으로 모듈을 저장

❖ 예제 : 08/calc_tester.py

```
import calculator
```

불러올 모듈의 이름 calculator.py에서 ".py"는 생략합니다.

```
print(calculator.plus(10, 5))
print(calculator.minus(10, 5))
print(calculator.multiply(10, 5))
print(calculator.divide(10, 5))
```

모듈이름.함수()의 꼴로 calculator 모듈의 함수를 호출합니다.

❖ 실행 결과:

```
>calc_tester.py
15
5
50
2.0
```

모듈 - import에 대하여

❖ import 의 역할

- "다른 모듈 내의 코드에 대한 접근"을 가능하게 하는 것
- "다른 코드"에는 변수, 함수, 클래스 등이 모두 포함

❖ import문을 사용하는 첫 번째 방법

```
import 모듈 #모듈의 실제 파일명은 "모듈.py"
```

❖ import문을 사용하는 두 번째 방법

```
from 모듈 import 변수 또는 함수
```

from 모듈	from 모듈 import 변수 또는 함수
<pre>import calculator print(calculator.plus(10, 5)) print(calculator.minus(10, 5)) print(calculator.multiply(10, 5)) print(calculator.divide(10, 5))</pre>	<pre>from calculator import plus from calculator import minus from calculator import multiply from calculator import divide print(plus(10, 5)) print(minus(10, 5)) print(multiply(10, 5)) print(divide(10, 5))</pre>

모듈 - import에 대하여

❖ “from 모듈 import 변수 또는 함수”의 세 가지 버전

- 예제 : 08/calc_tester2.py

❖ 사용할 변수나 함수의 이름을 일일이 명기

```
from calculator import plus
from calculator import
```

calculator 모듈의 plus라는 함수를 불러들였으므로 “calculator.” 없이 plus() 이름만으로 함수를 호출할 수 있습니다.

```
print(plus(10, 5))
print(minus(10, 5))
#print(multiply(10, 5))
#print(divide(10, 5))
```

multiply()와 divide()는 import하지 않았습니다. 현재 모듈에서는 보이지 않는 함수입니다.

❖ 예제 : 08/calc_tester3.py

- 콤마(,)를 이용해서 여러 함수(또는 변수)의 이름을 한 줄에 기입

```
from calculator import plus, minus
```

```
print(plus(10, 5))
print(minus(10, 5))
#print(multiply(10, 5))
#print(divide(10, 5))
```

from calculator import plus
from calculator import minus
와 동일한 코드입니다.

모듈 - import에 대하여

❖ 예제 : 08/calc_tester4.py

- 와일드카드 *를 이용

```
from calculator import *  
  
print(plus(10, 5))  
print(minus(10, 5))  
print(multiply(10, 5))  
print(divide(10, 5))
```

❖ 그러나 import *와 같은 코드는 지양할 것을 권장

- 코드가 복잡해지고 모듈의 수가 많아지면 어떤 모듈 또는 어떤 변수, 함수를 불러오고 있는지 파악하기 힘들어짐. 코드 가독성을 떨어뜨림

❖ 예제 : 08/calc_tester5.py (import 모듈 as 새이름)

```
import calculator as c
```

calculator 모듈을 c라는 이름으로 불러옵니다.

```
print(c.plus(10, 5))  
print(c.minus(10, 5))  
print(c.multiply(10, 5))  
print(c.divide(10, 5))
```

calculator라는 이름 대신 c를 이용하여 함수 이름에 접근합니다.

모듈 - import에 대하여

❖ import문을 실행할 때 파이썬이 모듈 파일을 찾는 순서

- 1) 파이썬 인터프리터 내장(Built-In) 모듈
- 2) sys.path에 정의되어 있는 디렉토리

❖ sys.builtin_module_names를 출력하면 파이썬에 내장되어 있는 모듈의 목록을 볼 수 있음.

- import문이 실행되면 파이썬은 가장 먼저 이 목록을 확인함.

❖ 실습 1

```
>>> import sys
>>> print(sys.builtin_module_names)
('_ast', '_bisect', '_codecs', '_codecs_cn', '_codecs_hk',
'_codecs_iso2022', '_codecs_jp', '_codecs_kr', '_codecs_tw',
'_collections', '_csv', '_datetime', '_functools', '_heapq', '_imp',
'_io', '_json', '_locale', '_lsprof', '_md5', '_multibytecodec',
'_opcode', '_operator', '_pickle', '_random', '_sha1', '_sha256',
'_sha512', '_sre', '_stat', '_string', '_struct', '_symtable',
'_thread', '_tracemalloc', '_warnings', '_weakref', '_winapi', 'array',
'atexit', 'audioop', 'binascii', 'builtins', 'cmath', 'errno',
'faulthandler', 'gc', 'itertools', 'marshal', 'math', 'mmap', 'msvcrt',
'nt', 'parser', 'signal', 'sys', 'time', 'winreg', 'xxsubtype',
'zipimport', 'zlib')
```


모듈 - import에 대하여

- ❖ 만약 가져오고자 하는 모듈이 앞에서 출력한 내장 모듈 목록 (`sys.builtin_module_names`)에 없다면, 파이썬은 `sys.path`에 정의되어 있는 디렉토리에서 모듈 파일을 탐색
 - `sys.path`에 정의되어 있는 디렉토리
 - 파이썬 모듈이 실행되고 있는 현재 디렉토리
 - `PYTHONPATH` 환경변수에 정의되어 있는 디렉토리
 - 파이썬과 함께 설치된 기본 라이브러리

모듈 - import에 대하여

❖ 예제 : 08/sys_path.py

```
import sys

for path in sys.path:
    print(path)
```

○ 실행 결과

```
C:\Users\SangHyun\OneDrive\문서\뇌자극파이썬3.0\08>sys_path.py
C:\Users\SangHyun\OneDrive\문서\뇌자극파이썬3.0\08
C:\WINDOWS\SYSTEM32\python34.zip
C:\Python34\DLLs
C:\Python34\lib
C:\Python34
C:\Python34\lib\site-packages
```

파이썬 모듈이 실행되고
있는 현재 디렉토리

파이썬과 함께 설치된 기본
라이브러리

모듈-메인 모듈과 하위 모듈

❖ 메인 모듈 : 최상위 수준으로 실행되는 스크립트

- 파이썬에서는 "어떻게 만드느냐"가 아닌 "어떻게 실행하느냐"에 따라 메인 모듈이 결정
- 파이썬에는 내장 전역 변수인 `__name__`이 있는데, 이 변수는 모듈이 최상위 수준으로 실행될 때 `'__main__'`으로 지정

❖ 예제 : 08/top_level.py

```
print('name : {0}'.format(__name__))
```

- 실행 결과

```
>top_level.py  
name : __main__
```

모듈-메인 모듈과 하위 모듈

❖ 예제 : 08/main_sub/sub.py

```
print("beginning of sub.py...")
print('name : {0}'.format(__name__))
print("end of sub.py...")
```

❖ 예제 : 08/main_sub/main.py

```
import sub

print("beginning of main.py...")
print('name : {0}'.format(__name__))
print("end of main.py...")
```

○ 실행 결과

```
>main.py
beginning of sub.py...
name : sub
end of sub.py...
beginning of main.py...
name : __main__
end of main.py...
```

모듈-메인 모듈과 하위 모듈

❖ 예제 : 08/main sub2/sub.py

```
if __name__ == '__main__':  
    print("beginning of sub.py...")  
    print('name : {0}'.format(__name__))  
    print("end of sub.py...")
```

❖ 예제 : 08/main_sub2/main.py

```
import sub  
  
print("beginning of main.py...")  
print('name : {0}'.format(__name__))  
print("end of main.py...")
```

○ 실행 결과

```
>main.py  
beginning of main.py...  
name : __main__  
end of main.py...
```

sub.py의 출력문들이 실행되지 않았습니다.

```
>sub.py  
beginning of sub.py...  
name : __main__  
end of sub.py...
```

최상위 수준으로 실행하면 sub 모듈의 __name__ 변수는 '__main__'이 되어 출력문들을 실행합니다.

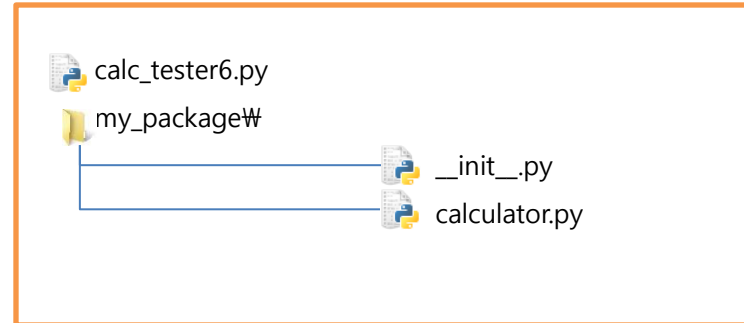
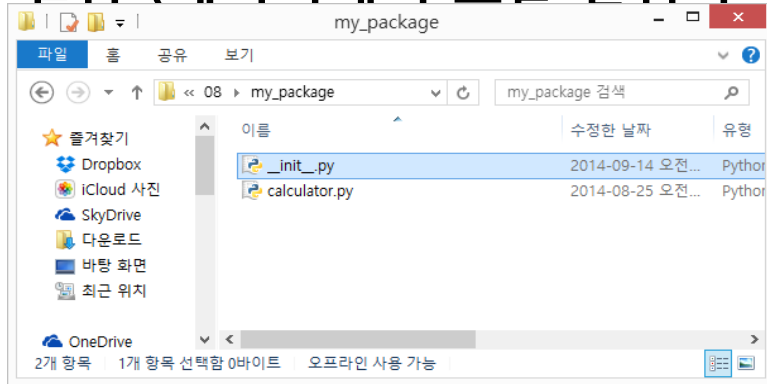
패키지

❖ 패키지

- 모듈을 모아놓는 디렉토리
- 모듈 꾸러미로 해석하면 이해하기 편함
- 디렉토리가 "파이썬의 패키지"로 인정받으려면 `__init__.py` 파일을 그 경로에 갖고 있어야 함

패키지

❖ 실습 (패키지에서 모듈 반입하기)



❖ 08/calc_tester6.py

```
from my_package import calculator
```

```
print(calculator.plus(10, 5))  
print(calculator.minus(10, 5))  
print(calculator.multiply(10, 5))  
print(calculator.divide(10, 5))
```

“from 패키지 import 모듈”의
꼴로 모듈을 불러옵니다.

○ 실행 결과

```
>calc_tester6.py  
15  
5  
50  
2.0
```

패키지 - __init__.py에 대하여

- ❖ 보통의 경우, `__init__.py` 파일은 대개 비워둠
- ❖ 이 파일을 손대는 경우는 `__all__`이라는 변수를 조정할 때 정도
 - `__all__`은 다음과 같은 코드를 실행할 때 패키지로부터 반입할 모듈의 목록을 정의하기 위해 사용

```
from 패키지 import *
```

- ❖ `import *`은 사용을 자제하는 것이 좋음.

패키지 - __init__.py에 대하여

❖ 실습 (__all__ 변수 조정하기)

- 08/luv_song/eeny.py

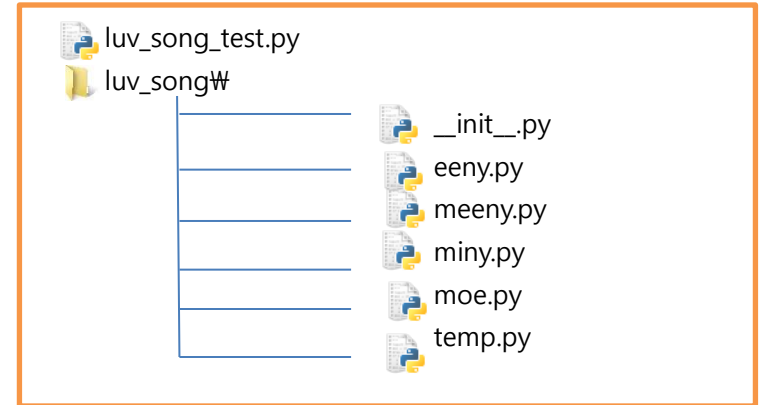
```
def test():  
    print('module name :  
{0}'.format(__name__))
```

- 08/luv_song/meeny.py

```
def test():  
    print('module name :  
{0}'.format(__name
```

- 08/luv_song/miny.py

```
def test():  
    print('module name :  
{0}'.format(__name__))
```



패키지 - __init__.py에 대하여

❖ 실습 (__all__ 변수 조정하기)

- 08/luv_song/moe.py

```
def test():  
    print('module name :  
{0}'.format(__name__))
```

- 08/luv_song/__init__.py

```
__all__ = ['eeny', 'meeny', 'miny',  
'moe']
```

- 08/luv_song_test.py

```
from luv_song import *  
  
eeny.test()  
meeny.test()  
miny.test()  
moe.test()
```

❖ 실행결과

```
>luv_song_test.py  
module name : luv_song.eeny  
module name : luv_song.meeny  
module name : luv_song.miny  
module name : luv_song.moe
```

패키지 - site-packages에 대하여

❖ site-packages

- 파이썬의 기본 라이브러리 패키지 외에 추가적인 패키지를 설치하는 디렉토리
- 각종 서드 파티 모듈을 바로 이 곳에 설치함

❖ 실습 1 (site-packages 확인)

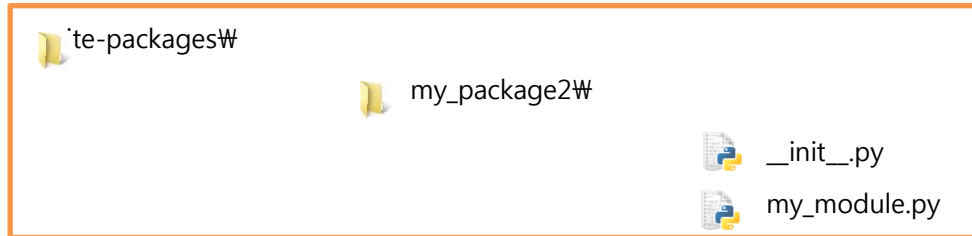
```
>>> import sys
>>> sys.path
['', 'C:\\Python34\\Lib\\idlelib',
'C:\\WINDOWS\\SYSTEM32\\python34.zip', 'C:\\Python34\\DLLs',
'C:\\Python34\\lib', 'C:\\Python34',
'C:\\Python34\\lib\\site-packages']
```

site-package는 파이썬이 기본적으로 모듈을 탐색하는 경로에 포함되어 있습니다.

패키지 - site-packages에 대하여

❖ 실습 2

- sys.path에 있던 site-package 디렉토리에 다음과 같이 my_package2 디렉토리를 만들고, 그 안에 __init__.py와 my_module.py 를 생성



- ...\site-packages\my_package2\my_module.py

```
def info():  
    print(__name__)  
    print(__file__)
```

- 파이썬 셸을 열고 다음 코드를 입력

```
>>> from my_package2 import my_module  
>>> my_module.info()  
my_package2.my_module  
C:\Python34\lib\site-packages\my_package2\my_module.py
```