



모듈과 패키지

파이썬 프로그래밍

Contents

❖ 모듈

- 두 개의 소스 파일로 만드는 하나의 프로그램 예제
- import에 대하여
- 모듈을 찾아서
- 메인 모듈과 하위 모듈

❖ 패키지

- __init__.py에 대하여
- site-packages에 대하여



❖ 모듈

- 일반적으로는 “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**



❖ 수학과 관련된 기능

```
import math
```

❖ 여러 변수와 함수를 가진 집합체

자동 완성 기능으로 살펴보는 math 모듈의 변수와 함수

```
1 import math
2
3 math.
```

acos	def acos(x)
acosh	Return the arc cosine (measured in radians) of x.
asin	
asinh	
atan	
atan2	
atanh	
ceil	
copysign	
cos	
cosh	
degrees	



❖ 수학/삼각함수

```
>>> import math
```

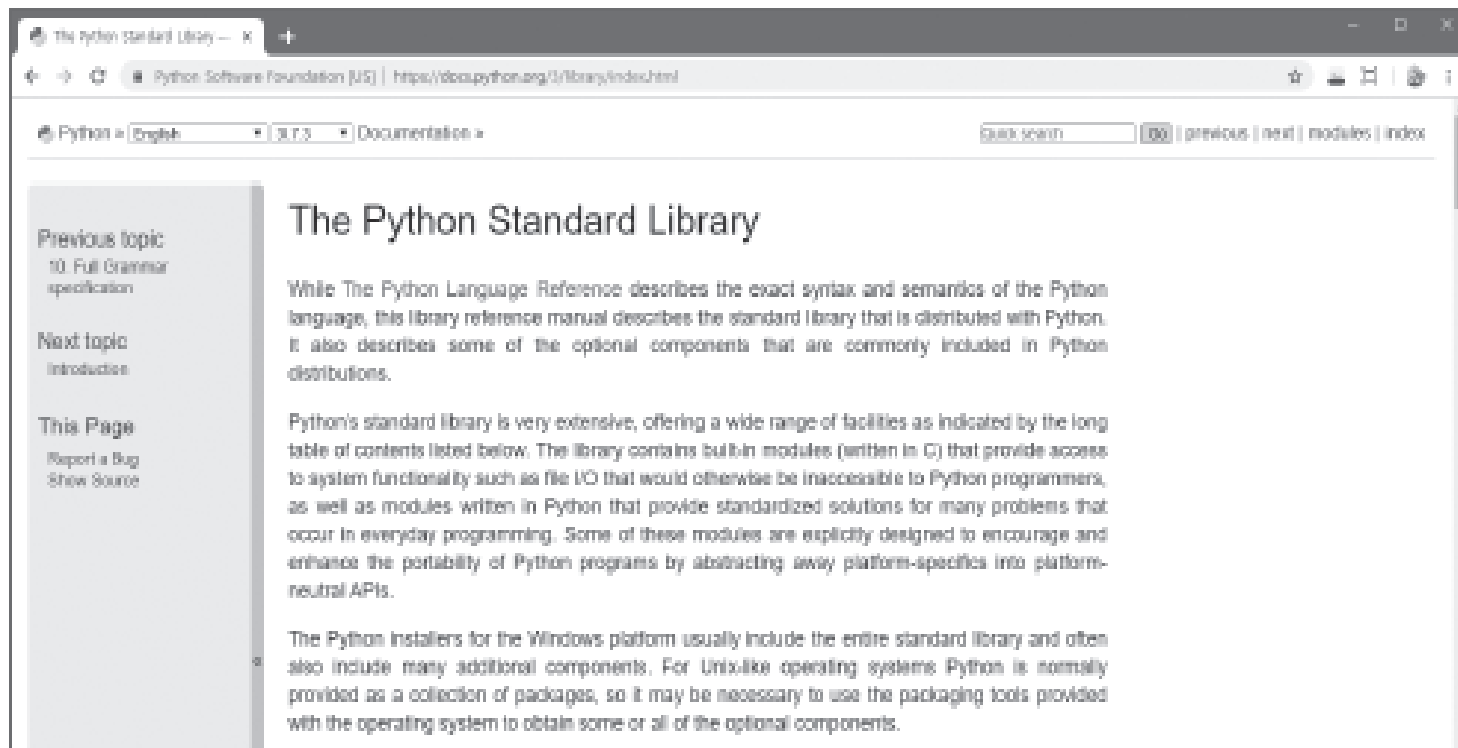
```
>>> math.sin(1)      # 사인
0.8414709848078965
>>> math.cos(1)      # 코사인
0.5403023058681398
>>> math.tan(1)      # 탄젠트
1.5574077246549023
>>>
>>> math.floor(2.5)   # 내림
2
>>> math.ceil(2.5)    # 올림
3
```

변수 또는 함수	설명
<code>sin(x)</code>	사인값을 구합니다.
<code>cos(x)</code>	코사인값을 구합니다.
<code>tan(x)</code>	탄젠트값을 구합니다.
<code>log(x[, base])</code>	로그값을 구합니다.
<code>ceil(x)</code>	올림합니다.
<code>floor(x)</code>	내림합니다.



❖ 파이썬 공식 문서에서 표준 모듈 정보 확인 가능

- <http://docs.python.org/3/library/index.html>




random 모듈

❖ random 모듈

```
import random
```

- random 모듈 문서
 - <http://docs.python.org/3/library/random.html#examples-and-recipes>



The screenshot shows a web browser window with the title 'random — generate pseudorandom numbers and distributions'. The address bar shows the URL: <https://docs.python.org/3/library/random.html#examples-and-recipes>. The page content is titled 'Examples and Recipes' and lists 'Basic examples:'. The examples are as follows:

```
>>> random()                                # Random float: 0.0 <= x < 1.0
0.37444887175646646

>>> uniform(2.5, 10.0)                       # Random float: 2.5 <= x < 10.0
1.1800146073117523

>>> expvariate(1 / 5)                         # Interval between arrivals averaging 5 seconds
5.148557573865803

>>> randrange(10)                             # Integer from 0 to 9 inclusive
7

>>> randrange(0, 101, 2)                      # Even integer from 0 to 100 inclusive
26

>>> choice(['win', 'lose', 'draw'])            # Single random element from a sequence
'draw'

>>> deck = 'ace two three four'.split()
>>> shuffle(deck)                             # Shuffle a list
>>> deck
['four', 'two', 'ace', 'three']

>>> sample([10, 20, 30, 40, 50], k=4)          # Four samples without replacement
[40, 10, 50, 30]
```



random 모듈

- 예시

```
01 import random
02 print("# random 모듈")
03
04 # random(): 0.0 <= x < 1.0 사이의 float를 리턴합니다.
05 print("- random():", random.random())
06
07 # uniform(min, max): 지정한 범위 사이의 float를 리턴합니다.
08 print("- uniform(10, 20):", random.uniform(10, 20))
09
10 # randrange(): 지정한 범위의 int를 리턴합니다.
11 # - randrange(max): 0부터 max 사이의 값을 리턴합니다.
12 # - randrange(min, max): min부터 max 사이의 값을 리턴합니다.
13 print("- randrange(10)", random.randrange(10))
```



random 모듈

```
14
15 # choice(list): 리스트 내부에 있는 요소를 랜덤하게 선택합니다.
16 print("- choice([1, 2, 3, 4, 5]):", random.choice([1, 2, 3, 4, 5]))
17
18 # shuffle(list): 리스트의 요소들을 랜덤하게 섞습니다.
19 print("- shuffle([1, 2, 3, 4, 5]):", random.shuffle([1, 2, 3, 4, 5]))
20
21 # sample(list, k=<숫자>): 리스트의 요소 중에 k개를 뽑습니다.
22 print("- sample([1, 2, 3, 4, 5], k=2):", random.sample([1, 2, 3, 4, 5], k=2))
```

실행결과

```
# random 모듈
- random(): 0.5671614057098718
- uniform(10, 20): 18.627114055572356
- randrange(10) 6
- choice([1, 2, 3, 4, 5]): 2
- shuffle([1, 2, 3, 4, 5]): None
- sample([1, 2, 3, 4, 5], k=2): [5, 4]
```



random 모듈

- 5행의 random.random()처럼 random을 계속 입력하는 것은 효율적이지 못하므로 from 구문 활용해서 임포트

```
from time import random, randrange, choice
```



❖ sys 모듈

- 시스템과 관련된 정보 가진 모듈
- 명령 매개변수 받을 때 많이 사용

```
01  # 모듈을 읽어 들입니다.  
02  import sys  
03  
04  # 명령 매개변수를 출력합니다.  
05  print(sys.argv)
```



sys 모듈

```
06  print("----")
07
08  # 컴퓨터 환경과 관련된 정보를 출력합니다.
09  print("getwindowsversion()", sys.getwindowsversion())
10  print("----")
11  print("copyright:", sys.copyright)
12  print("----")
13  print("version:", sys.version)
14
15  # 프로그램을 강제로 종료합니다.
16  sys.exit()
```

```
> python module_sys.py 10 20 30
```

- 5행 sys.argv
 - 아래와 같이 실행하면 ['module_sys.py', '10', '20', '30'] 리스트 들어옴



['module_sys.py', '10', '20', '30'] → 명령 매개변수입니다. 입력한 명령어에 따라 달라집니다.

```
getwindowsversion:() sys.getwindowsversion(major=10, minor=0, build=14393,  
platform=2, service_pack='')
```

copyright: Copyright (c) 2001-2019 Python Software Foundation.

All Rights Reserved.

...생략...

```
version: 3.7.3 (v3.7.3:ef4ecbed12, Mar 21 2019, 17:54:52) [MSC v.1916 32  
bit (Intel)]
```



❖ os 모듈

- 운영체제와 관련된 기능 가진 모듈
- 새로운 폴더 만들거나 폴더 내부 파일 목록 보는 등

```
01  # 모듈을 읽어 들입니다.  
02  import os  
03  
04  # 기본 정보를 몇 개 출력해봅시다.  
05  print("현재 운영체제:", os.name)  
06  print("현재 폴더:", os.getcwd())  
07  print("현재 폴더 내부의 요소:", os.listdir())  
08  
09  # 폴더를 만들고 제거합니다[폴더가 비어있을 때만 제거 가능].  
10  os.mkdir("hello")  
11  os.rmdir("hello")  
12
```



```
13  # 파일을 생성하고 + 파일 이름을 변경합니다.  
14  with open("original.txt", "w") as file:  
15      file.write("hello")  
16  os.rename("original.txt", "new.txt")  
17  
18  # 파일을 제거합니다.  
19  os.remove("new.txt")  
20  # os.unlink("new.txt")  
21  
22  # 시스템 명령어 실행  
23  os.system("dir")
```



현재 운영체제: nt

현재 폴더: C:\Users\hasat\sample

현재 폴더 내부의 요소: ['.vscode', 'beaut.py', 'download-png1.py', 'file.txt', 'freq.json', 'ghostdriver.log', 'iris.csv', 'lang-plot.png', 'mnist', 'mtest.py', 'newFile.xlsx', 'output.png', 'proj', 'rint.py', 'stats_104102.xlsx', 'test', 'test.csv', 'test.html', 'test.png', 'test.py', 'test.rb', 'test.txt', 'test_a.txt', 'train', 'underscore.js', 'Website.png', 'Website_B.png', 'Website_C.png', 'Website_D.png', '__pycache__']

C 드라이브의 볼륨: BOOTCAMP

볼륨 일련 번호: FCCF-6067

C:\Users\hasat\sample 디렉터리

```
2019-05-01 오전 12:18 <DIR>      .
2019-05-01 오전 12:18 <DIR>      ..
...생략...
2019-05-28 오전 04:49 <DIR>      __pycache__
                24개 파일            1,908,017 바이트
                8개 디렉터리 16,895,188,992 바이트 남음
```

→ 명령 프롬프트에서
그냥 dir을 입력했을 때의
결과와 동일합니다.
단지 파이썬에서
dir 명령어를
호출했을 뿐입니다.



datetime 모듈

❖ datetime 모듈

- date(날짜) 및 time(시간)과 관련된 모듈로, 날짜 형식 만들 때 자주 사용되는 코드들로 구성

```
01  # 모듈을 읽어 들입니다.  
02  import datetime  
03  
04  # 현재 시각을 구하고 출력하기  
05  print("# 현재 시각 출력하기")  
06  now = datetime.datetime.now()  
07  print(now.year, "년")  
08  print(now.month, "월")  
09  print(now.day, "일")  
10  print(now.hour, "시")  
11  print(now.minute, "분")  
12  print(now.second, "초")  
13  print()  
14
```

실행결과

```
# 현재 시각 출력하기  
2019 년  
4 월  
23 일  
3 시  
51 분  
41 초  
  
# 시간을 포맷에 맞춰 출력하기  
2019.04.23 03:51:41  
2019년 4월 23일 3시 51분 41초  
2019년 04월 23일 03시 51분 41초
```



datetime 모듈

```
15 # 시간 출력 방법
16 print("# 시간을 포맷에 맞춰 출력하기")
17 output_a = now.strftime("%Y.%m.%d %H:%M:%S")
18 output_b = "{}년 {}월 {}일 {}시 {}분 {}초".format(now.year,\
19     now.month,\
20     now.day,\
21     now.hour,\
22     now.minute,\
23     now.second)
24 output_c = now.strftime("%Y{} %m{} %d{} %H{} %M{} %S{}").format(*"년월일시분초")
25 print(output_a)
26 print(output_b)
27 print(output_c)
28 print()
```

↓
문자열, 리스트 등 앞에 *을 붙이면
요소 하나하나가 매개변수로 지정됩니다.



datetime 모듈

- output_a처럼 strftime() 함수 사용하면 시간을 형식에 맞춰 출력 가능
- 그 외 다양한 시간 처리 기능

```
01  # 모듈을 읽어 들입니다.  
02  import datetime  
03  now = datetime.datetime.now()  
04  
05  # 특정 시간 이후의 시간 구하기  
06  print("# datetime.timedelta로 시간 더하기")  
07  after = now + datetime.timedelta(\  
08      weeks=1,\  
09      days=1,\  
10      hours=1,\  
11      minutes=1,\  
12      seconds=1)
```



datetime 모듈

```
13 print(after.strftime("%Y{} %m{} %d{} %H{} %M{} %S{}").format(*"년월일시분초"))
14 print()
15
16 # 특정 시간 요소 교체하기
17 print("# now.replace()로 1년 더하기")
18 output = now.replace(year=(now.year + 1))
19 print(output.strftime("%Y{} %m{} %d{} %H{} %M{} %S{}").format(*"년월일시분초"))
```

실행결과

```
# datetime.timedelta로 시간 더하기
2019년 05월 01일 03시 39분 26초

# now.replace()로 1년 더하기
2020년 04월 23일 02시 38분 25초
```



datetime 모듈

- timedelta() 함수 사용하면 특정한 시간의 이전 또는 이후 구함
 - “1년 후” 구할 때는 replace() 함수 사용해 날짜 값을 교체



time 모듈

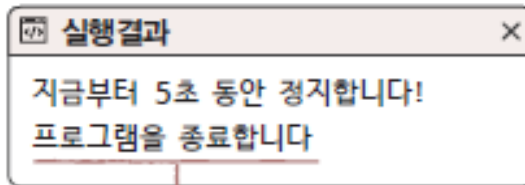
❖ time 모듈

- 시간과 관련된 기능

```
import time
```

- time.sleep() 함수
 - 특정 시간 동안 코드 진행을 정지
 - 정지하고 싶을 시간을 초 단위로 입력

```
01 import time
02
03 print("지금부터 5초 동안 정지합니다!")
04 time.sleep(5)
05 print("프로그램을 종료합니다")
```



↓
5초 동안 정지한 이후에 출력합니다.



urllib 모듈

❖ urllib 모듈

- URL 다루는 라이브러리

```
01  # 모듈을 읽어 들입니다.  
02  from urllib import request  
03  
04  # urlopen() 함수로 구글의 메인 페이지를 읽습니다.  
05  target = request.urlopen("https://google.com")  
06  output = target.read()  
07  
08  # 출력합니다.  
09  print(output)
```

- urlopen() 함수 : URL 주소의 페이지 열기



urllib 모듈

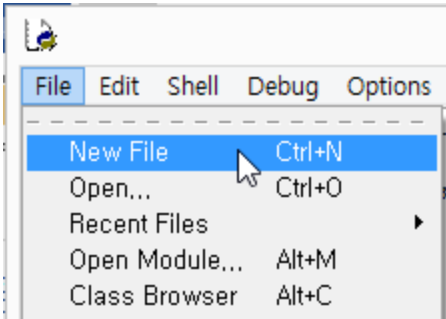
```
b'<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage"
lang="ko"><head><meta content="text/html; charset=UTF-8" http-equiv="Content-
Type"><meta content="/logos/doodles/2019/amy-johnsons-114th-birthday-
5154304993263616.2-law.gif" itemprop="image">
...생략...
```

- 바이너리 데이터



모듈 - 두 개의 소스 파일로 만드는 하나의 프로그램 예제

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



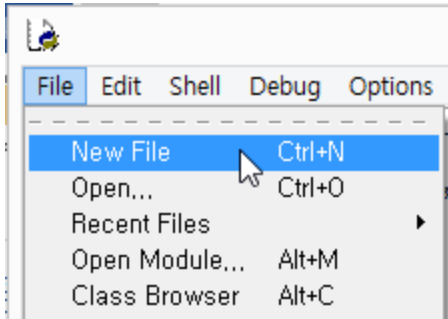
IDLE 편집창에서 [File]→[Save]
메뉴 항목을 선택하고, 디렉토리를
하나 골라 그곳에
"calculator.py"라는 이름으로
모듈을 저장

- ❖ 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]
메뉴 항목을 선택하고, 디렉토리를
하나 골라 그곳에
"calc_tester.py"라는 이름으로
모듈을 저장

- ❖ 예제 : 08/calc_tester.py

```
import calculator
```

불러올 모듈의 이름 calculator.py
에서 ".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 모
듈의 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
```

import 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 minus
```

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

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

multiply()와 divide()는 import하지 않았습니니다. 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-모듈을 찾아서

❖ 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-모듈을 찾아서

❖ 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

❖ 예제 : 08/sys_path.py

```
import sys

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

• 실행 결과

```
C:\Users\SangHyun\OneDrive\문서\뇌 자극\파이썬3.0\08>sys_path.py
C:\Users\SangHyun\OneDrive\문서\뇌 자극\파이썬3.0\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 라
이브러리

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.

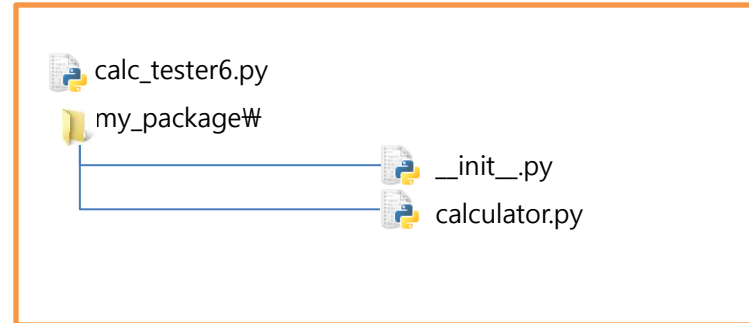
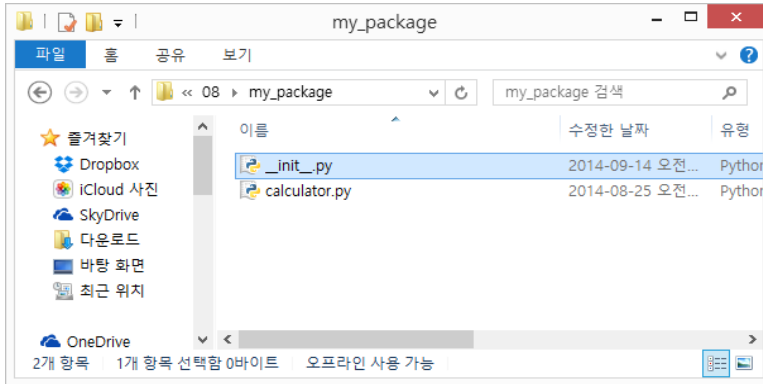
❖ 패키지

- 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__.py`에 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

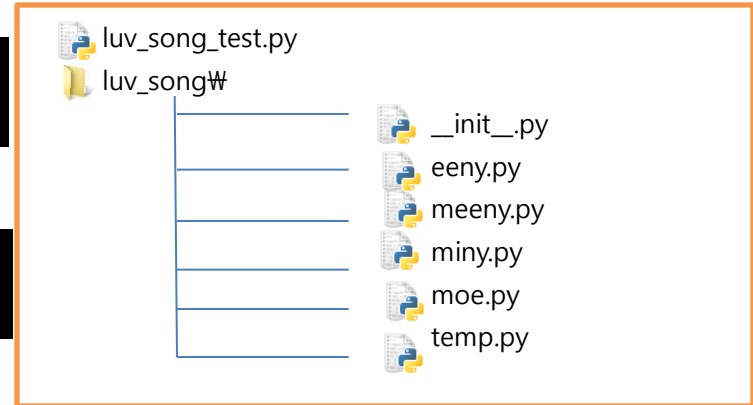
```
def test():  
    print('module name : {0}'.format(__name__))
```

❖ 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()
```

• 실행 결과

```
>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']
```

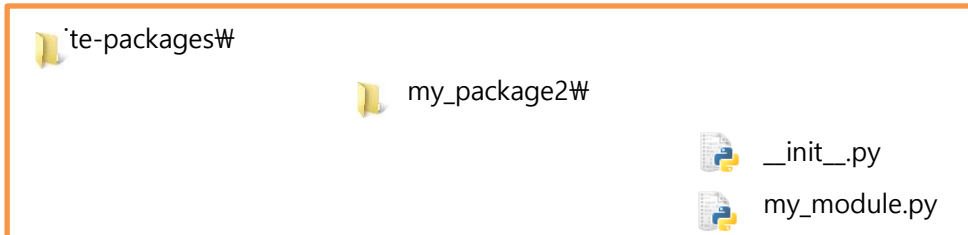
site-package는 파이썬이 기본적으로 모듈을 탐색하는 경로에 포함되어 있습니다.



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
```





Thank You !

파이썬 프로그래밍