데이터 처리 프로그래밍

Data Processing Programming

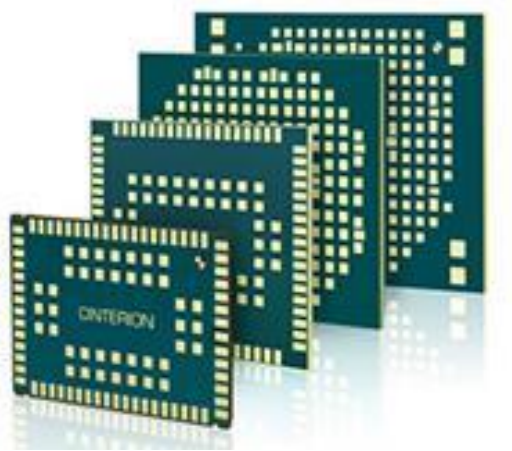
09
모듈과 패키지

목차

1. 모듈

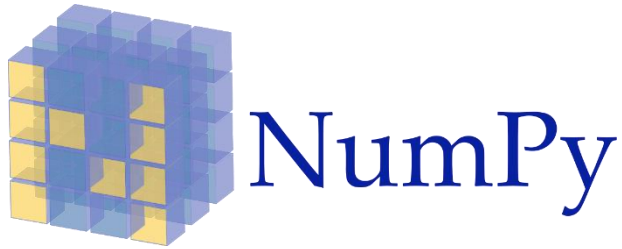
모듈Module

- 함수, 변수 그리고 클래스의 집합
- 다른 파이썬 프로그램에서 가져와 사용할 수 있는 파이썬 파일
- 파이썬에는 다른 사람들이 만들어 놓은 모듈이 굉장히 많음
- 사용자가 모듈은 직접 만들어서 사용할 수도 있음



모듈의 종류

- 표준 모듈: 파이썬에서 기본 제공하는 모듈
- 사용자 정의 모듈: 사용자가 직접 정의해서 사용하는 모듈
- 서드 파티 모듈: 외부에서 제공하는 모듈
  - 파이썬 표준 모듈에 모든 기능이 있지 않음
  - 서드 파티 모듈을 이용해 고급 프로그래밍 가능
  - 게임 개발을 위한 pygame, 데이터베이스 기능의 SQLAlchemy, 데이터 분석 기능의 NumPy



모듈의 장점

- 단순성Simplicity
  - 전체 문제에 초점을 맞추기보다는 문제의 상대적으로 작은 부분에만 초점을 맞춤
  - 단일 모듈로 작업할 수 있는 작은 도메인
  - 개발이 쉬우며 오류 발생이 적음
- 유지보수성Maintainability
  - 일반적으로 모듈은 서로 다른 문제 영역간에 논리적 경계를 설정하도록 설계
  - 상호 의존성을 최소화하는 방식으로 모듈을 작성하여 단일 모듈을 수정하면 프로그램의 다른 부분에 영향을 미칠 가능성이 줄어듬
  - 모듈 외부의 응용 프로그램에 대해 전혀 알지 못해도 모듈을 변경할 수 있음
  - 개발팀이 대규모 응용 프로그램에서 공동으로 작업 할 수 있음
- 재사용성Reusability
  - 단일 모듈에서 정의 된 기능은 응용 프로그램의 다른 부분에서 (적절히 정의 된 인터페이스를 통해) 쉽게 재사용 가능
  - 중복 코드를 만들 필요가 없음
- 범위 지정Scoping
  - 일반적으로 모듈은 프로그램의 여러 영역에서 식별자 간의 충돌을 피하는 데 도움이되는 별도의 네임 스페이스를 정의

파이썬 표준 모듈

```
>>> import sys
>>> sys.builtin_module_names
('_abc', '_ast', '_bisect', '_blake2', '_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', '_sha3',
'_sha512', '_signal', '_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', 'sys', 'time', 'winreg', 'xxsubtype', 'zipimport', 'zlib')
```

```
>>> dir(__builtins__)
['__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
'__getattribute__', '__getitem__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__',
'__lt__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__setitem__', '__sizeof__', '__str__',
'__subclasshook__', 'clear', 'copy', 'fromkeys', 'get', 'items', 'keys', 'pop', 'popitem', 'setdefault', 'update',
'values']
```

수학 계산과 시간 모듈


```
>>> import math
>>> dir(math)
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh',
'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp',
'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p',
'log2', 'modf', 'nan', 'pi', 'pow', 'radians', 'remainder', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']
```

```
>>> import time
>>> dir(time)
['_STRUCT_TM_ITEMS', '__doc__', '__loader__', '__name__', '__package__', '__spec__', 'altzone', 'asctime', 'clock',
'ctime', 'daylight', 'get_clock_info', 'gmtime', 'localtime', 'mktime', 'monotonic', 'monotonic_ns', 'perf_counter',
'perf_counter_ns', 'process_time', 'process_time_ns', 'sleep', 'strftime', 'strptime', 'struct_time', 'thread_time',
'thread_time_ns', 'time', 'time_ns', 'timezone', 'tzname']
```

랜덤 모듈Random Module

- 대표적인 모듈 중 하나인 랜덤 모듈
- 랜덤 모듈을 사용하기 위해서는 import random 필요
  - random.random(): 0.0~1.0 미만의 실수값 반환
  - random.randint(1, 10): 1~10 사이의 정수 반환
  - random.randrange(0, 10, 2): 0~10 미만의 2의 배수만 반환
  - random.choice(): 자료형 변수에서임의의 값 반환
  - random.sample(): 자료형 변수에서 필요한 개수만큼 반환
  - random.shuffle(): 자료형 변수 내용을 랜덤으로 셔플

```
>>> import random
>>> random.random()
0.193971779648763
>>> random.randint(1, 10)
7
>>> random.randrange(0, 10, 2)
6
>>> l = [10, 20, 30, 40, 50]
>>> random.choice(l)
30
>>> random.sample(l, 2)
[10, 30]
>>> random.shuffle(l)
>>> l
[30, 10, 20, 50, 40]
```

네임스페이스Namespace

- 모듈 호출의 범위 지정
- 모듈 이름에 alias를 생성하여 모듈의 이름을 바꿔 사용

```
1    import random as rd
2
3    print(rd.random())
4    print(rd.randrange(0, 10, 2))
```

- from 구문을 사용하여 모듈에서 특정 함수 또는 클래스만 호출

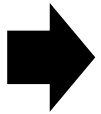
```
1    from random import random, randrange
2
3    print(random())
4    print(randrange(0, 10, 2))
```

- * 을 사용하여 모듈 안에 모든 함수, 클래스, 변수를 가져옴

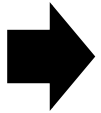
```
1    from random import *
2
3    print(random())
4    print(randrange(0, 10, 2))
```

모듈 생성

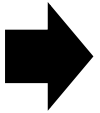

```python
#Module.py
def func1():
    print("Module.py - func1()")

def func2():
    print("Module.py - func2()")

def func3():
    print("Module.py - func3()")
```



```python
import Module

Module.func1()
Module.func2()
Module.func3()
```



```python
from Module import func1, func2, func3

func1()
func2()
func3()
```



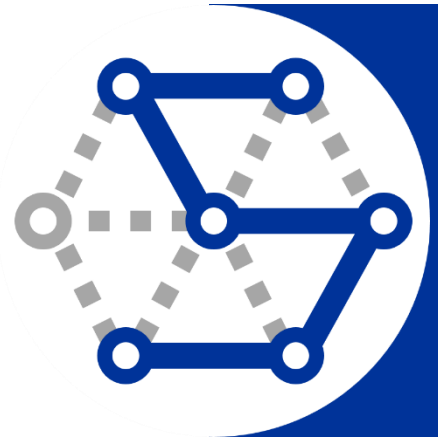
```python
from Module import *

func1()
func2()
func3()
```

Lab. 계산기 모듈 만들기


```
# Calculator.py
```

```
from Calculator import *

print(add(3, 5))
print(sub(3, 5))
print(mul(3, 5))
print(div(3, 5))
print(mod(3, 5))
```

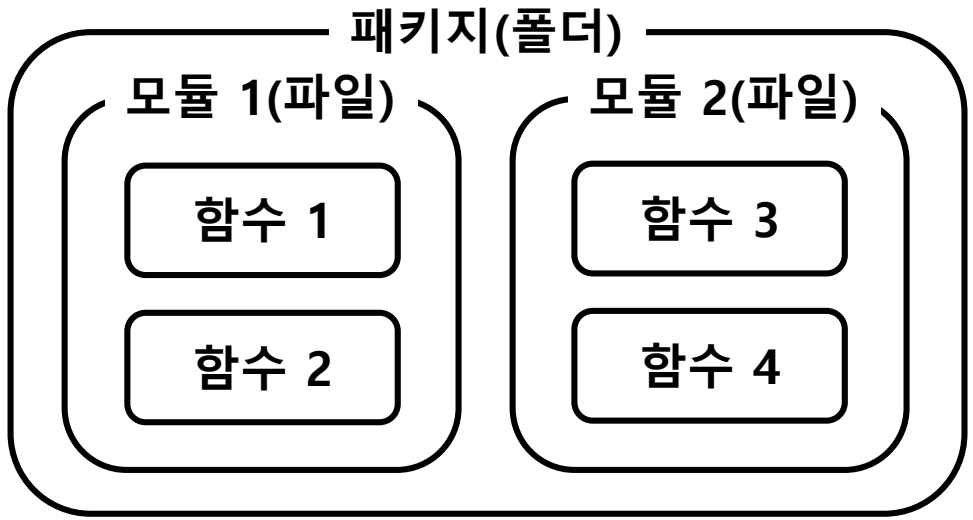
```
8
-2
15
0.6
3
```



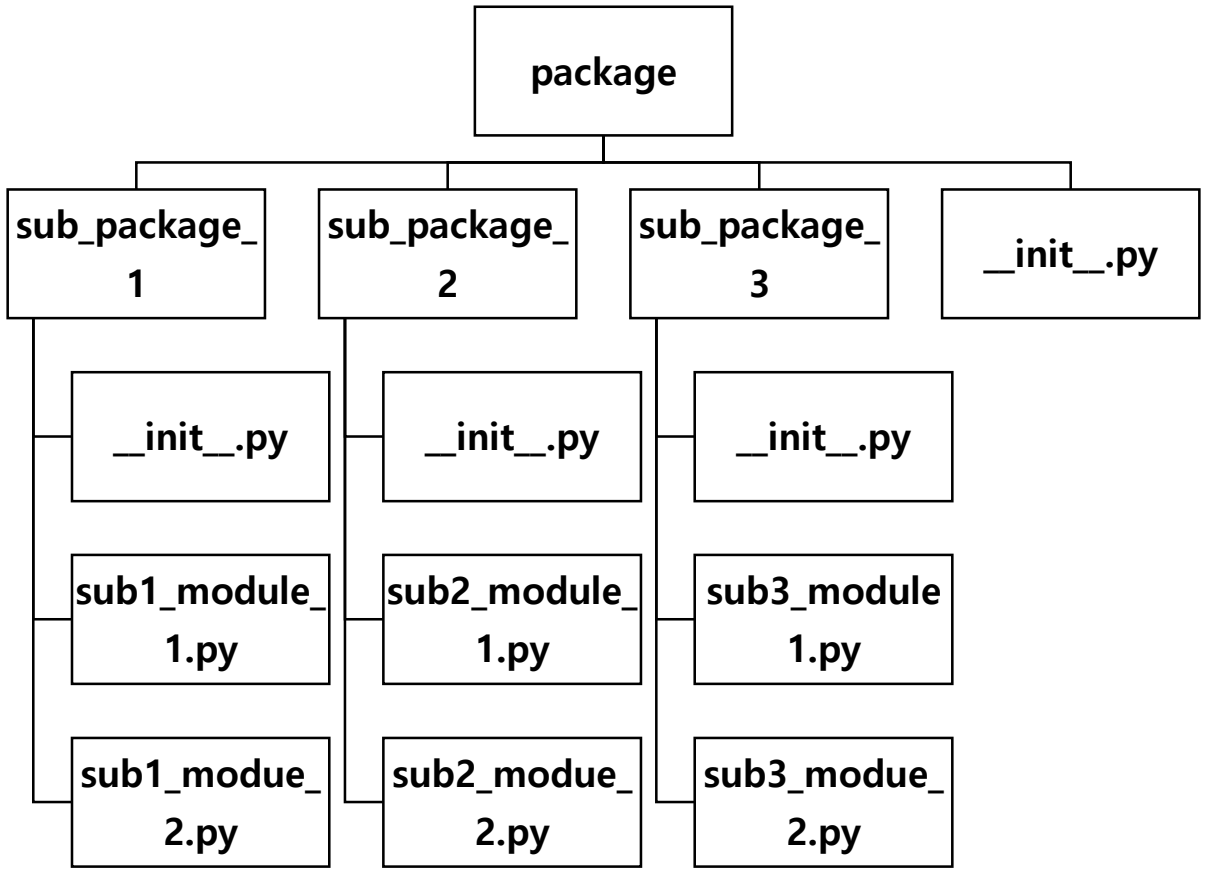
2. 패키지

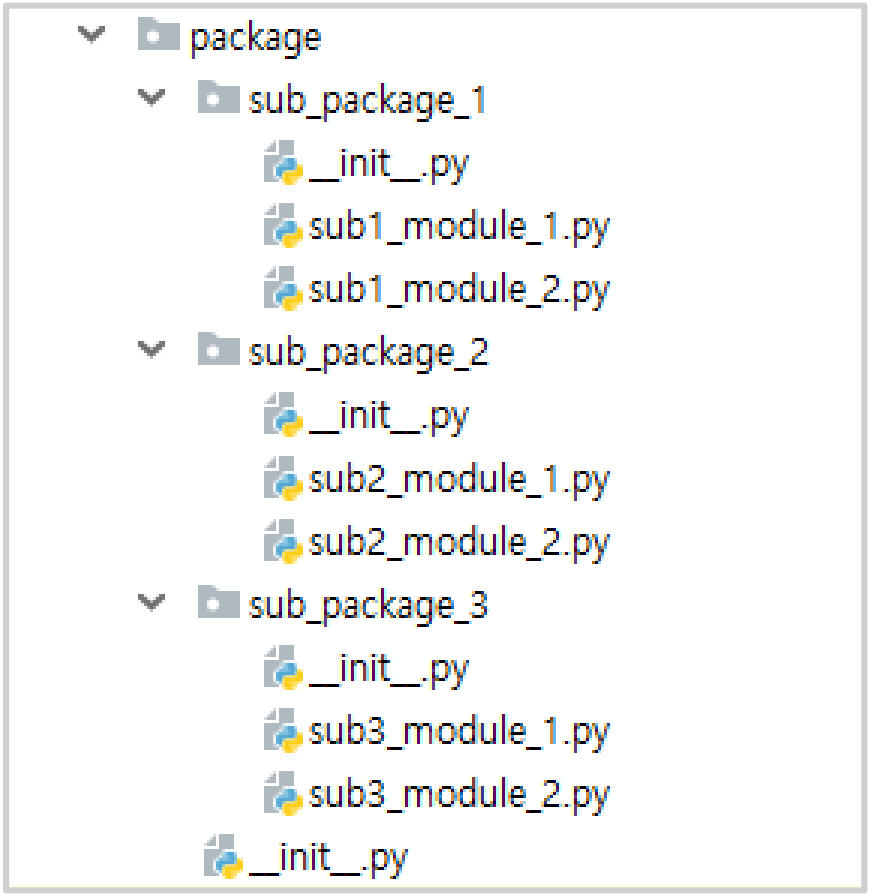
패키지 Packages

- 패키지는 모듈의 집합
- 패키지 안에 여러 모듈이 존재
- 모듈을 주제별로 분리할 때 사용
- 디렉터리와 같이 계층적인 구조로 관리
- 모듈들이 서로 포함 관계를 가지며 거대한 패키지를 가짐
- 파이썬에서는 패키지가 하나의 라이브러리



패키지 구성 및 생성


package
sub_package_1
sub_package_2
sub_package_3
__init__.py
__init__.py
__init__.py
__init__.py
sub1_module_1.py
sub2_module_1.py
sub3_module 1.py
sub1_modue_2.py
sub2_modue_2.py
sub3_modue_2.py


package
sub_package_1
__init__.py
sub1_module_1.py
sub1_module_2.py
sub_package_2
__init__.py
sub2_module_1.py
sub2_module_2.py
sub_package_3
__init__.py
sub3_module_1.py
sub3_module_2.py
__init__.py

패키지 실행


```
def print_module():
    print("sub_package_1 -> module_1")
```

```
from package.sub_package_1 import module_1, module_2
module_1.print_module()
module_2.print_module()

from package.sub_package_2 import module_1, module_2
module_1.print_module()
module_2.print_module()

from package.sub_package_3 import module_1, module_2
module_1.print_module()
module_2.print_module()
```

```
sub_package_1 -> module_1
sub_package_1 -> module_2
sub_package_2 -> module_1
sub_package_2 -> module_1
sub_package_3 -> module_1
sub_package_3 -> module_2
```

패키지 실행


```
1  from package.sub_package_1 import *
2  module_1.print_module()
3  module_2.print_module()
```

```
D:\Python\venv\Scripts\python.exe D:/Python/package_test.py
Traceback (most recent call last):
  File "D:/Python/package_test.py", line 2, in <module>
    module_1.print_module()
NameError: name 'module_1' is not defined
```

__init__.py

- 파이썬 패키지를 선언하는 초기화 스크립트
- 패키지에 대한 메타데이터에 해당하는 내용 포함
- 파이썬의 거의 모든 라이브러리에 포함
- 파이썬 버전 3.3 부터는 __init__.py 파일이 없어도 패키지로 인식
- 파이썬 버전 3.3 밑의 하위 버전과 호환을 위해 __init__.py 파일 생성

패키지 실행

- __all__이라는 리스트형의 변수에 하위 패키지의 이름을 작성

```
#package/__init__.py

__all__=['sub_package_1', 'sub_package_2', 'sub_package_3']
```

```
#package/sub_package_1/__init__.py

__all__=['module_1', 'module_2']
```

```
from package.sub_package_1 import *
module_1.print_module()
module_2.print_module()
```

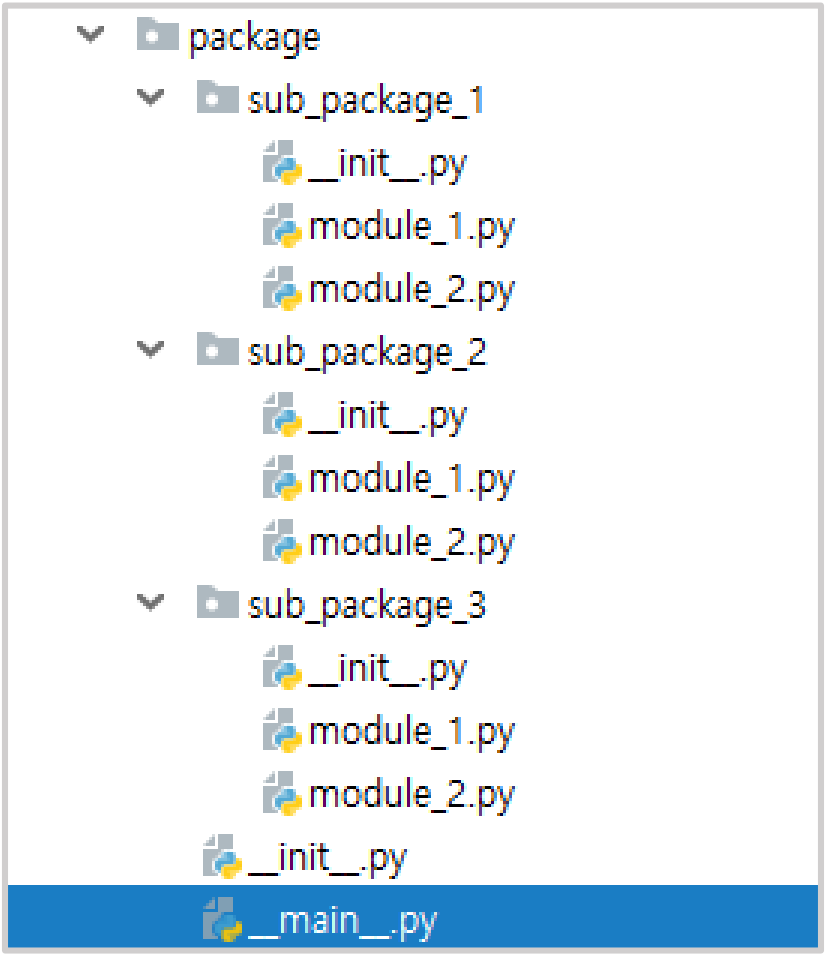
```
sub_package_1 -> module_1
sub_package_1 -> module_2
```

__main__.py

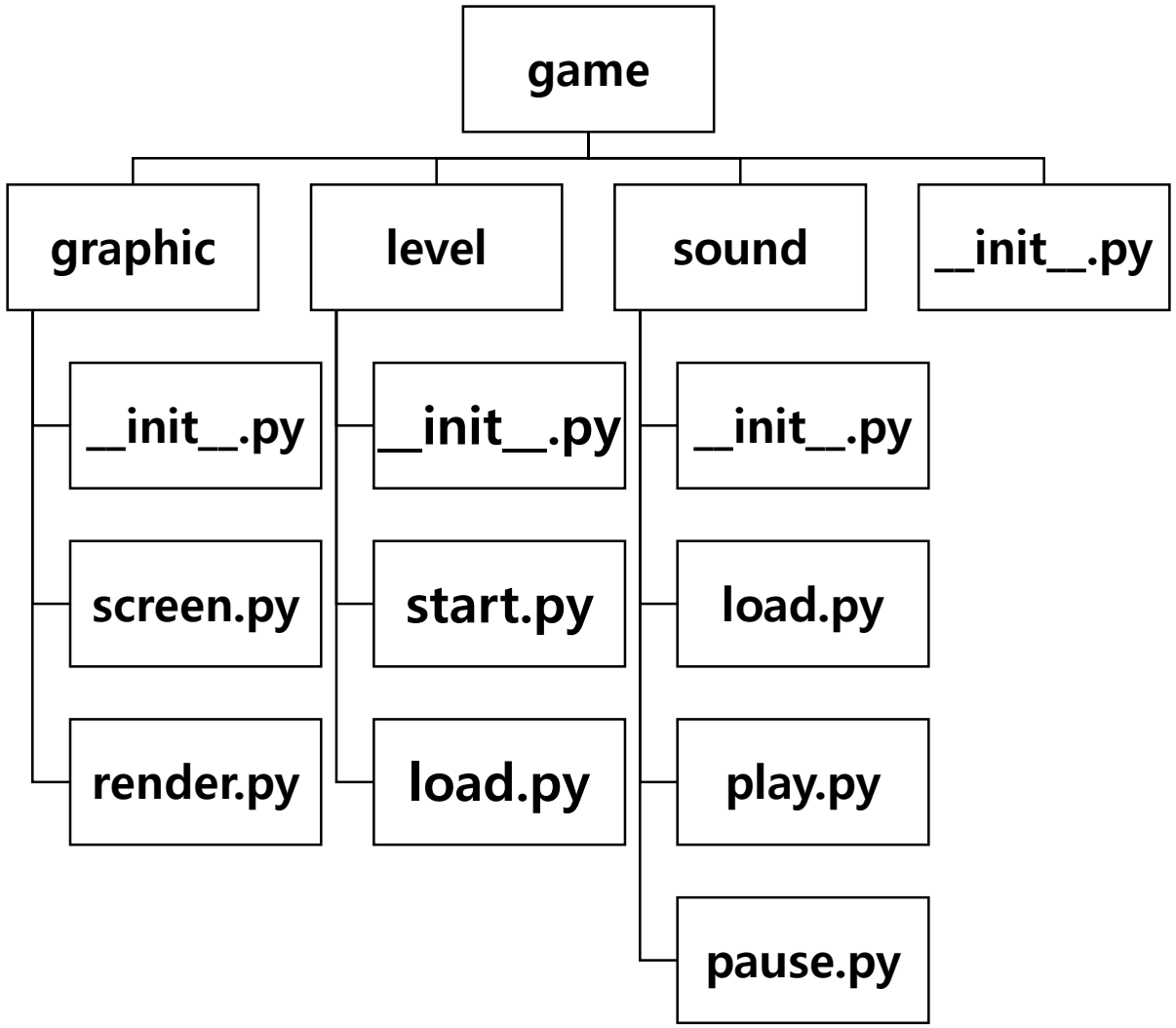
- 패키지 자체를 실행하기 위한 용도
- 패키지를 실행시키면 __main__.py 실행

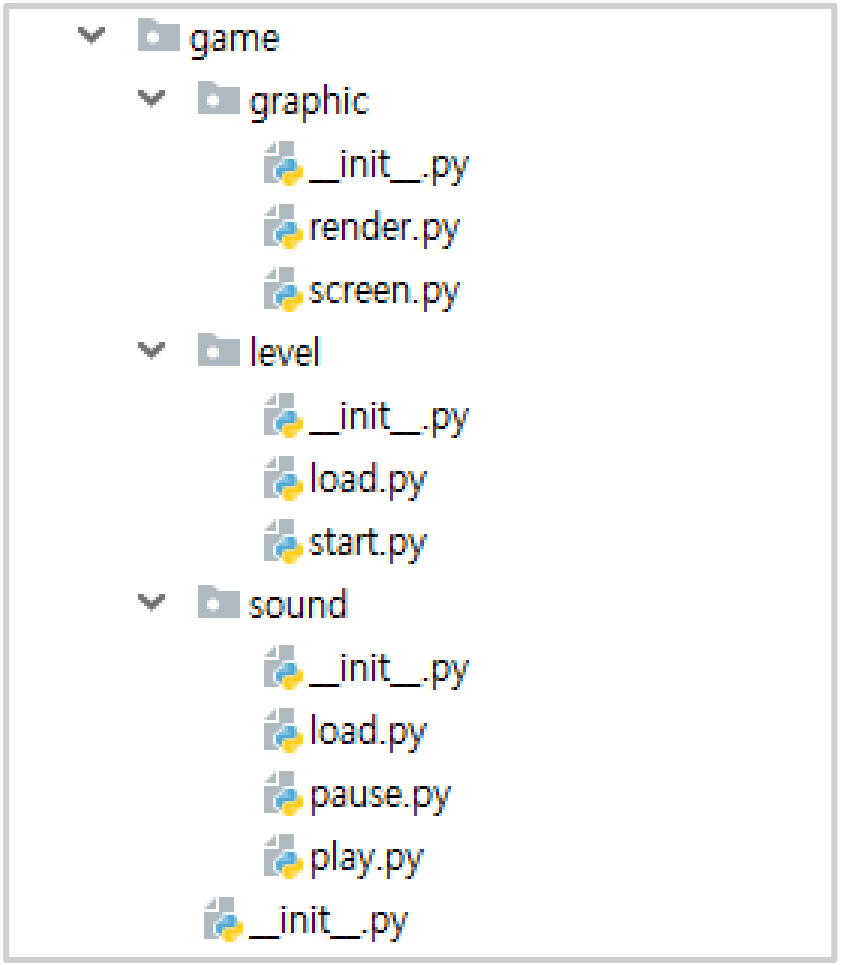
```
1   from sub_package_1 import module_1, module_2
2
3   if __name__ == '__main__':
4       module_1.print_module()
5       module_2.print_module()
```

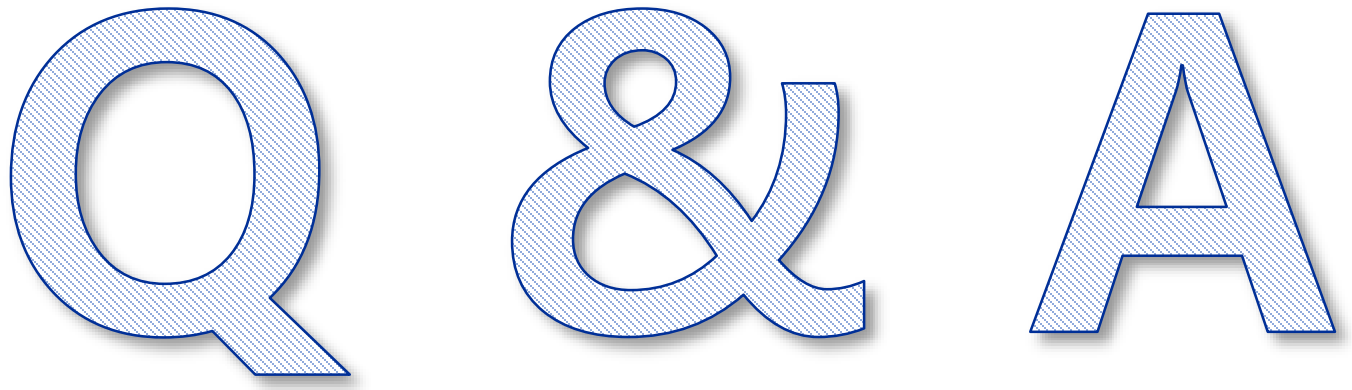
```
D:\Python>python package
sub_package_1 -> module_1
sub_package_1 -> module_2
```



Lab. game 패키지 구성






Q & A