LESSON

# 모듈과 패키지

# 모듈과 패키지의 필요성



- 규모가 큰 프로그램을 작성할 때 **기능별로 분류해서 만들고 그것들을 조립해서 사용**하는 방식을 사용한다.
- **재사용성** : 자주 사용되는 코드가 있는 경우 재사용할 수 있도록 모듈로 만들면 된다.
- **단순화** : 대규모 애플리케이션 작성시 해당 문제들을 작게 분해해서 작성한다.
- **유지 관리** : 모듈화하면 문제 발생시 문제가 발생한 부분만 수정하면 된다.

# 프로젝트, 패키지, 모듈



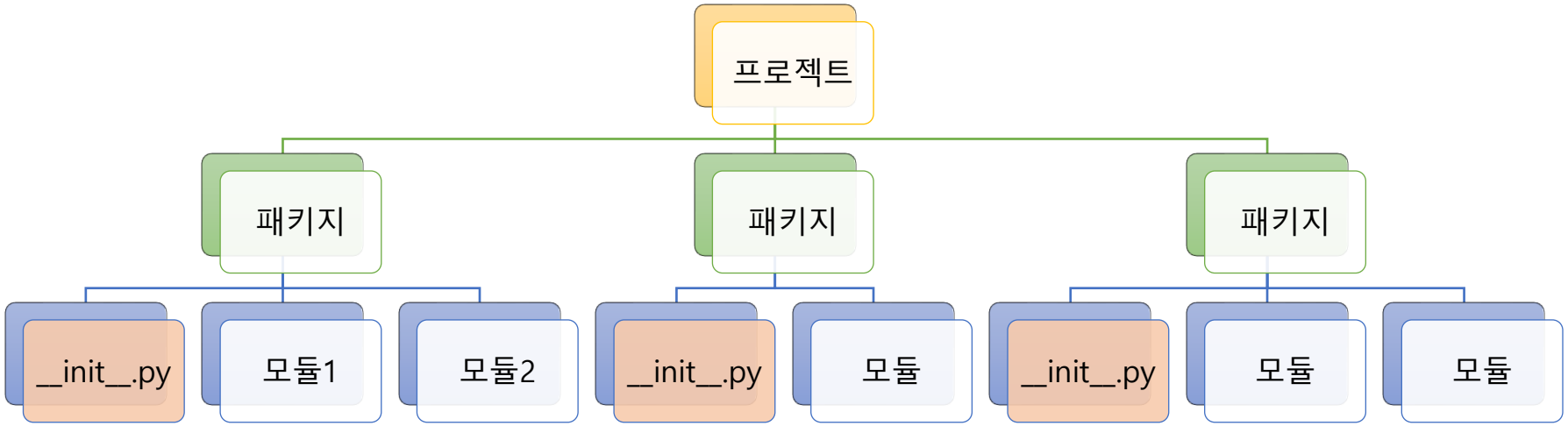
**프로젝트(project)**
하나의 완전한 프로그램

**패키지(package)**
모듈을 모아둔 디렉토리(폴더)

**모듈(module)**
함수, 변수, 클래스를 저장한 파일

디렉토리(폴더)가 **패키지로 인식되려면 __init__.py**가 있어야 한다.

모듈 : 파이썬 파일(.py)



# 모듈(module)



- 변수나 함수 클래스를 모아 놓은 **파이썬 파일(.py)**
- 자주 사용되는 코드를 재사용할 수 있게 하려면 모듈로 만들면 됨
- 모듈의 종류

표준(내장) 모듈

사용자 정의 모듈

외부 모듈

3rd Party 모듈: 파이썬이 아닌 외부 회사나 단체가 제공하는 모듈

# if __name__=='__main__':



- 파이썬의 시작점을 알려주는 기능을 한다.
- 프로젝트에서 한 번만 사용 가능하다.

```
if __name__=='__main__':
    함수 또는 실행 코드
```

대부분의 프로그래밍 언어들은
프로그램의 시작을 알리는 main이라는
함수를 가지고 있는데 파이썬에서는 위의
코드를 사용하여 시작을 알려줘야 한다.

# __name__



- 모듈의 이름을 저장하는 변수.

test1.py 파일

```
def add(x,y):
        return x+y
```

test2.py 파일

```
import test1

print(test1.__name__)
print(__name__)
```

[결과]
**test1**
**__main__**

**__name__ : 모듈의 이름을 저장하는 변수.**
이 변수의 값을 통해 현재 파일이 시작점인지 모듈이 시작점인지 판단할 수 있다.

다른 파일에 import 될 때에는 name 변수의 값이 __main__이 아니다.
즉, 호스트인 경우에는 name 변수의 값이 __main__이다.

# 파이썬 내장 math 모듈 사용해 보기



**import 모듈명, 모듈명...**

모듈명.함수
모듈명.변수

```
import math

if __name__=='__main__':
    print(math.pi)
    print(math.sqrt(25))
```

**import 모듈명 as 별명**

별명.함수
별명.변수

```
import math as m

if __name__=='__main__':
    print(m.pi)
    print(m.sqrt(25))
```

**import**는 현재 디렉토리(폴더)에 있는 모듈이나 파이썬 라이브러리가 저장된 디렉토리(폴더)에 있는 모듈만 불러올 수 있다.

**as 키워드 사용**
모듈이나 함수의 이름이 긴 경우 as 키워드를 사용해서 별명(alias) 을 지정하면 편리한다.

# 파이썬 내장 math 모듈 사용해 보기



## from으로 모듈의 일부만 가져오기

**from 모듈명 import 함수**

함수
변수

```
# math모듈에서 pi,sqrt만 가져오기
from math import pi,sqrt

if __name__=='__main__':
  print(pi)
  print(sqrt(3))
```

## from으로 모듈의 모든 내용을 가져오기

**from 모듈명 import ***

함수
변수

```
from math import *

if __name__=='__main__':
  print(pi)
  print(sqrt(3))
```

# 모듈 작성하기



파일명: mycalc.py

```
PI=3.141592

def add(x,y):
    print("결과:",x+y)

def sub(x,y):
    print("결과:",x-y)
```

# 모듈 사용하기



파일명: main.py

```
import mycalc

if __name__=='__main__':
    mycalc.add(1,2)
    mycalc.sub(4,3)
```

**from**으로 모듈의 모든 내용을 가져오기

```
from mycalc import *

if __name__=='__main__':
    add(1,2)
    sub(4,3)
```

**from**으로 모듈의 일부만 가져오기

```
from mycalc import add

if __name__=='__main__':
    add(1,2)
    # sub(4,3)   # 오류 발생
```

# 패키지(package)

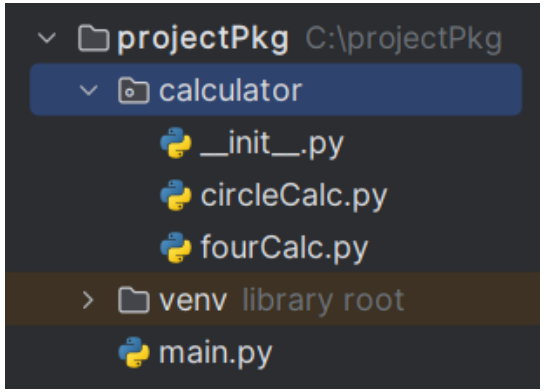


- 파이썬 모듈을 계층적으로 관리할 수 있게 해준다.
- 패키지는 디렉토리(폴더)와 파이썬 모듈로 구성된다.
- 복잡한 구조의 개발을 진행할 경우 공동 작업이나 유지 보수에 유리하다.
- 패키지 구조로 모듈을 만들면 다른 모듈과 이름이 겹치더라도 안전하게 사용할 수 있다.

# 패키지 작성하기



1. 현재 프로젝트에 calculator라는 이름의 파이썬 패키지를 만든다.
2. 패키지 안에 fourCalc.py, circleCalc.py을 만든다.




__init__.py 이 있으면
해당 디렉토리가
패키지로 인식된다.

파일명: fourCalc.py

fourCalc.py

```
1   def add(x, y):
2       print(x + y)
3
4   def sub(x, y):
5       print(x - y)
6
7   def mul(x, y):
8       print(x * y)
9
10  def div(x, y):
11      print(x / y)
```

파일명: circleCalc.py

circleCalc.py

```
1   def area(radius):
2       print(3.14*radius**2)
3
4   def circumference(radius):
5       print(2*3.14*radius)
```

# 패키지 사용하기



**import 패키지.모듈명**
**import 패키지.모듈명1, 패키지명.모듈명2,...**
**import 패키지.모듈명 as 별명**
패키지.모듈명.함수
패키지.모듈명.변수

파일명: main.py

```
import calculator.fourCalc
import calculator.circleCalc

if __name__=='__main__':
  calculator.fourCalc.add(3,4)
  calculator.circleCalc.area(5)
```

파일명: main.py

```
import calculator.fourCalc as fc
import calculator.circleCalc as cc

if __name__=='__main__':
  fc.add(3,4)
  cc.area(5)
```

# 패키지 사용하기



**from 패키지.모듈명 import 함수**
**from 패키지.모듈명 import 변수**
**from 패키지.모듈명 import 클래스**
함수
변수

파일명: main.py

```
from calculator.fourCalc import add
from calculator.circleCalc import area


if __name__=='__main__':
  add(4,5)
  area(5)
```

파일명: main.py

```
from calculator.fourCalc import *
from calculator.circleCalc import *


if __name__=='__main__':
  add(4,5)
  area(5)
```

# 패키지 사용하기



**import 패키지명**

패키지.모듈명.함수
패키지.모듈명.변수

파일명: __init__.py

```
from . import fourCalc
from . import circleCalc
```

"."은 현재 패키지를 의미한다.

파일명: main.py

```
import calculator

if __name__=='__main__':
  calculator.fourCalc.add(4,5)
  calculator.circleCalc.area(5)
  calculator.circleCalc.circumference(5)
```

# 패키지 사용하기



**from 패키지명 import ***

함수

변수

파일명: __init__.py

```
from .fourCalc import add, sub
from .circleCalc import area
```

파일명: main.py

```
from calculator import *

if __name__=='__main__':
  area(5)
  sub(3,4)
```

모듈명없이 바로 사용

# 패키지 사용하기



**import 패키지명**

패키지명.함수
패키지명.변수

파일명: __init__.py

```
from .fourCalc import *
from .circleCalc import *
```

파일명: main.py

```
import calculator

if __name__=='__main__':
  calculator.area(5)
  calculator.div(3,4)
```

모듈명없이 바로 사용

# 모듈의 위치에 관계없이 어디서든 import하기

# 파이썬 인터프리터가 모듈을 찾는 순서



1. 현재 작업 디렉토리
2. PYTHONPATH 환경 변수에 등록된 디렉토리
3. 표준 라이브러리 디렉토리

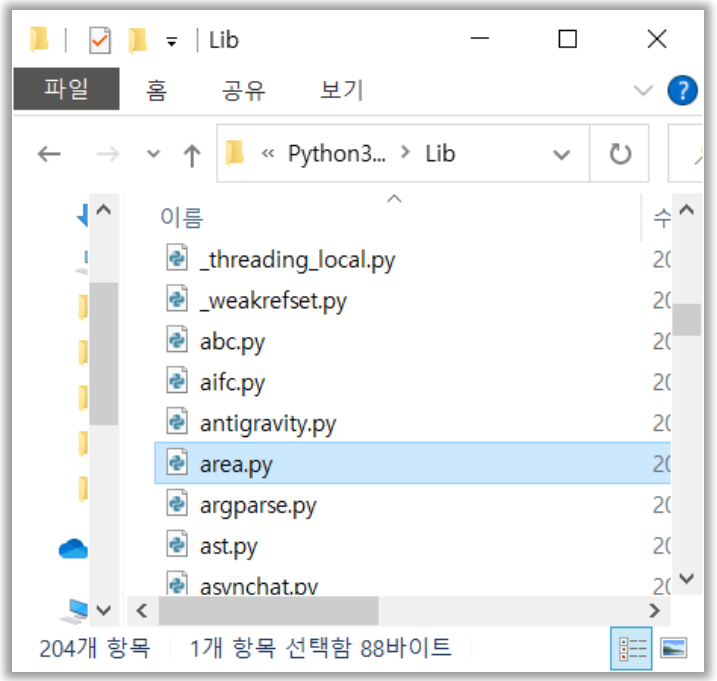
예: C:\Python312\Lib (파이썬이 설치된 경로의 Lib 디렉토리)

# 모듈의 위치에 관계없이 어디서든 import 하는 방법1



- 파이썬 인터프리터가 설치된 Lib디렉토리(폴더)에 넣기
  내가 만든 모듈(.py)을 파이썬이 설치된 Lib폴더에 저장하면 된다.

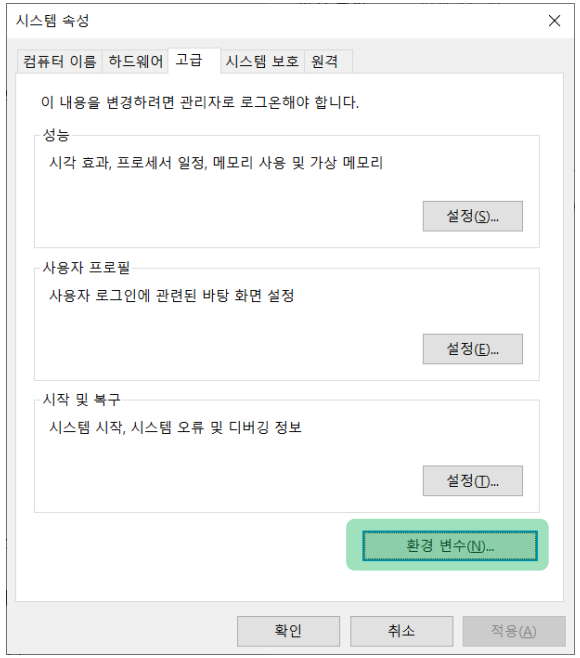
**C:₩Program Files₩Python3xx₩Lib**

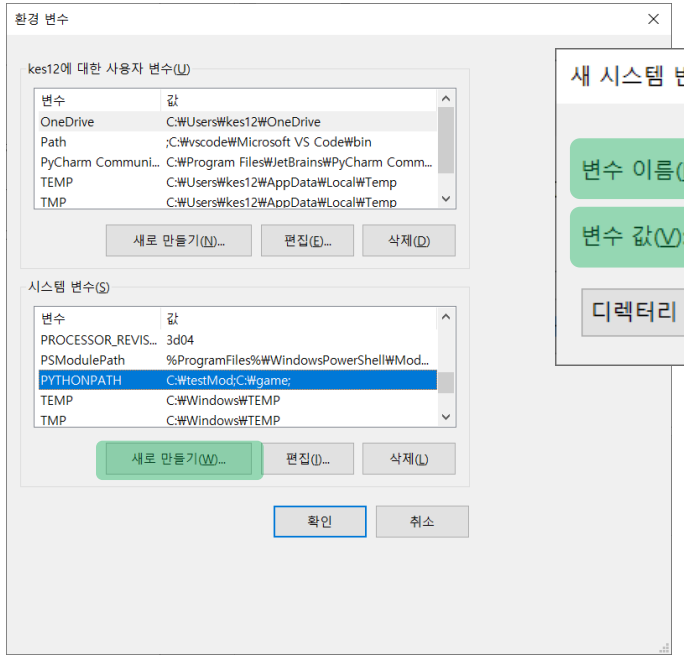


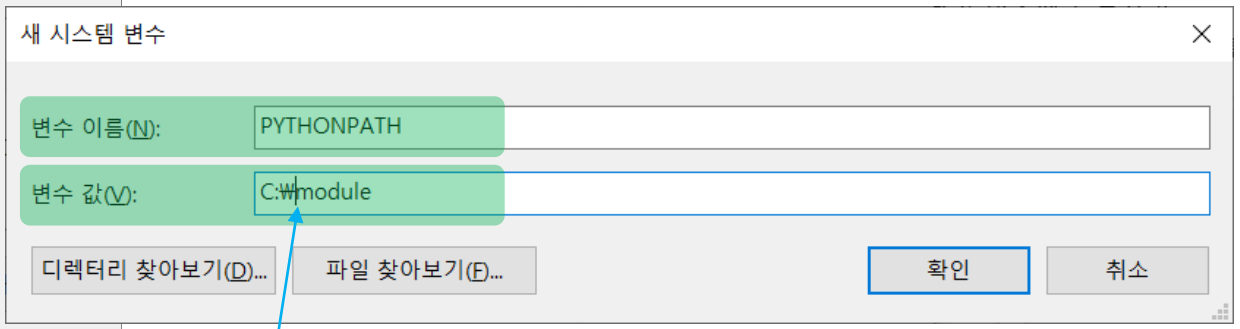
# 모듈의 위치에 관계없이 어디서든 import 하는 방법2



- **PYTHONPATH 환경 변수에 등록하기**
  제어판->시스템->고급 시스템 설정->환경 변수




추가할 경로 입력

환경 변수 등록 후 시스템을 재시작해야 적용됨.

sys 모듈을 사용해서 파이썬 라이브러리가 설치된 디렉토리를 확인할 수 있다.

```
import sys
print(sys.path)
```

# Built in module 목록 확인



- math모듈처럼 별도의 설치 없이 import해서 쓸 수 있는 모듈
- built in module 목록 확인

```
import sys
print(sys.builtin_module_names)
```

```
('_abc', '_ast', '_bisect', '_blake2', '_codecs', '_codecs_cn', '_codecs_hk', '_codecs_iso2022', '_codecs_jp',
'_codecs_kr', '_codecs_tw', '_collections', '_contextvars', '_csv', '_datetime', '_functools', '_heapq', '_imp',
'_io', '_json', '_locale', '_lsprof', '_md5', '_multibytecodec', '_opcode', '_operator', '_pickle', '_random',
'_sha1', '_sha256', '_sha3', '_sha512', '_signal', '_sre', '_stat', '_statistics', '_string', '_struct', '_symtable',
'_thread', '_tokenize', '_tracemalloc', '_typing', '_warnings', '_weakref', '_winapi', '_xxsubinterpreters',
'array', 'atexit', 'audioop', 'binascii', 'builtins', 'cmath', 'errno', 'faulthandler', 'gc', 'itertools', 'marshal',
'math', 'mmap', 'msvcrt', 'nt', 'sys', 'time', 'winreg', 'xxsubtype', 'zlib')
```