UNIT 45

모듈과 패키지 만들기

45 모듈과 패키지 만들기

모듈과 패키지 만들기

- 모듈(module)은 변수, 함수, 클래스 등을 모아 놓은 스크립트 파일이고, 패키지(package)는 여러 모듈을 묶은 것임
- 모듈은 간단한 기능을 담을 때 사용하며, 패키지는 코드가 많고 복잡할 때 사용함
- 패키지는 기능들이 모듈 여러 개로 잘게 나누어져 있고, 관련된 모듈끼리 폴더에 모여 있는 형태임

45.1 모듈 만들기

» 모듈 만들기

- 그럼 간단하게 2의 거듭제곱을 구하는 모듈을 만들어보자
- 다음 내용을 프로젝트 폴더(C:\project) 안에 square2.py 파일로 저장함
- IDLE에서 새 소스 파일을 만들고 저장하는 방법은 '3.2 IDLE에서 소스 파일 실행하기'를 참조하자

square2.py

```
base = 2          # 변수

def square(n):    # 함수
    return base ** n
```

- 모듈을 만들었을 때 모듈 이름은 square2임
- 스크립트 파일에서 확장자 .py를 제외하면 모듈 이름이 됨

45.1 모듈 만들기

» 모듈 사용하기

- 다음 내용을 프로젝트 폴더(C:\project) 안에 main.py 파일로 저장한 뒤 실행해보자
- 이때 square2.py 파일과 main.py 파일은 반드시 같은 폴더에 있어야 함

import 모듈

- 모듈.변수
- 모듈.함수()

main.py

```
import square2              # import로 square2 모듈을 가져옴

print(square2.base)         # 모듈.변수 형식으로 모듈의 변수 사용
print(square2.square(10))   # 모듈.함수() 형식으로 모듈의 함수 사용
```

실행 결과

```
2
1024
```

45.1 모듈 만들기

from import로 변수, 함수 가져오기

- 모듈에서 from import로 변수와 함수를 가져온 뒤 모듈 이름을 붙이지 않고 사용할 수도 있음

• from 모듈 import 변수, 함수

```
>>> from square2 import base, square
>>> print(base)
2
>>> square(10)
1024
```

45.1 모듈 만들기

모듈에 클래스 작성하기

- 다음 내용을 프로젝트 폴더(C:\project) 안에 person.py 파일로 저장하자

person.py

```
class Person:    # 클래스
    def __init__(self, name, age, address):
        self.name = name
        self.age = age
        self.address = address

    def greeting(self):
        print('안녕하세요. 저는 {0}입니다.'.format(self.name))
```

45.1 모듈 만들기

모듈에 클래스 작성하기

- 이제 main.py 파일을 다음과 같이 고쳐서 실행해보자

```
import 모듈
  • 모듈.클래스()
```

main.py

```
import person    # import로 person 모듈을 가져옴

# 모듈.클래스()로 person 모듈의 클래스 사용
maria = person.Person('마리아', 20, '서울시 서초구 반포동')
maria.greeting()
```

실행 결과

```
안녕하세요. 저는 마리아입니다.
```

45.1 모듈 만들기

from import로 클래스 가져오기

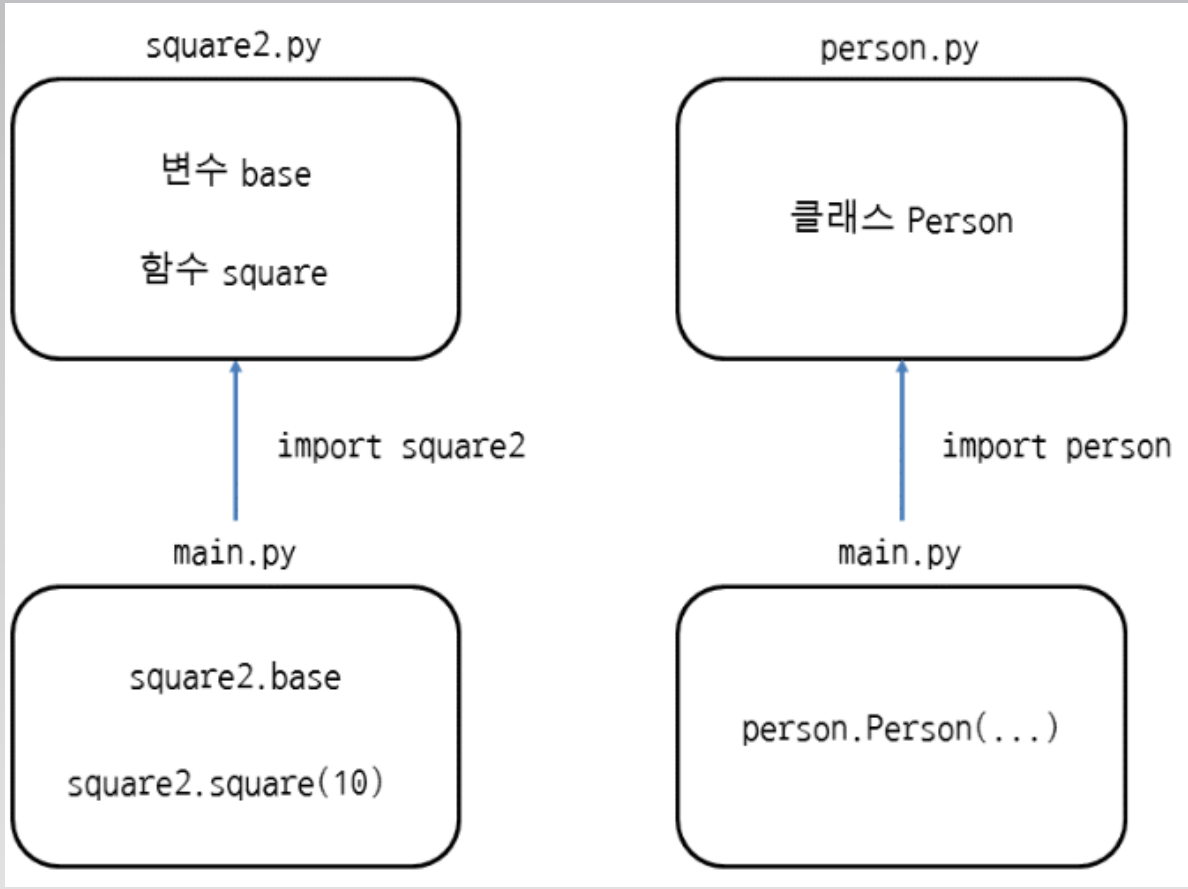
- 모듈에서 from import로 클래스를 가져온 뒤 모듈 이름을 붙이지 않고 사용할 수도 있음

• from 모듈 import 클래스

```
>>> from person import Person
>>> maria = Person('마리아', 20, '서울시 서초구 반포동')
>>> maria.greeting()
안녕하세요. 저는 마리아입니다.
```

45.1 모듈 만들기

▼ 그림 모듈 사용하기



45.2 모듈과 시작점 알아보기

» 모듈과 시작점 알아보기

- 인터넷에 있는 파이썬 코드를 보다 보면 if __name__ == '__main__':으로 시작하는 부분을 자주 만나게 됨

```
if __name__ == '__main__':
    코드
```

- 이 코드는 현재 스크립트 파일이 실행되는 상태를 파악하기 위해 사용함
- 다음 내용을 프로젝트 폴더(C:\project) 안에 hello.py 파일로 저장하자

hello.py

```
print('hello 모듈 시작')
print('hello.py __name__:', __name__)    # __name__ 변수 출력
print('hello 모듈 끝')
```

45.2 모듈과 시작점 알아보기

» 모듈과 시작점 알아보기

- 다음 내용을 프로젝트 폴더(C:\project) 안에 main.py 파일로 저장한 뒤 실행해보자

main.py

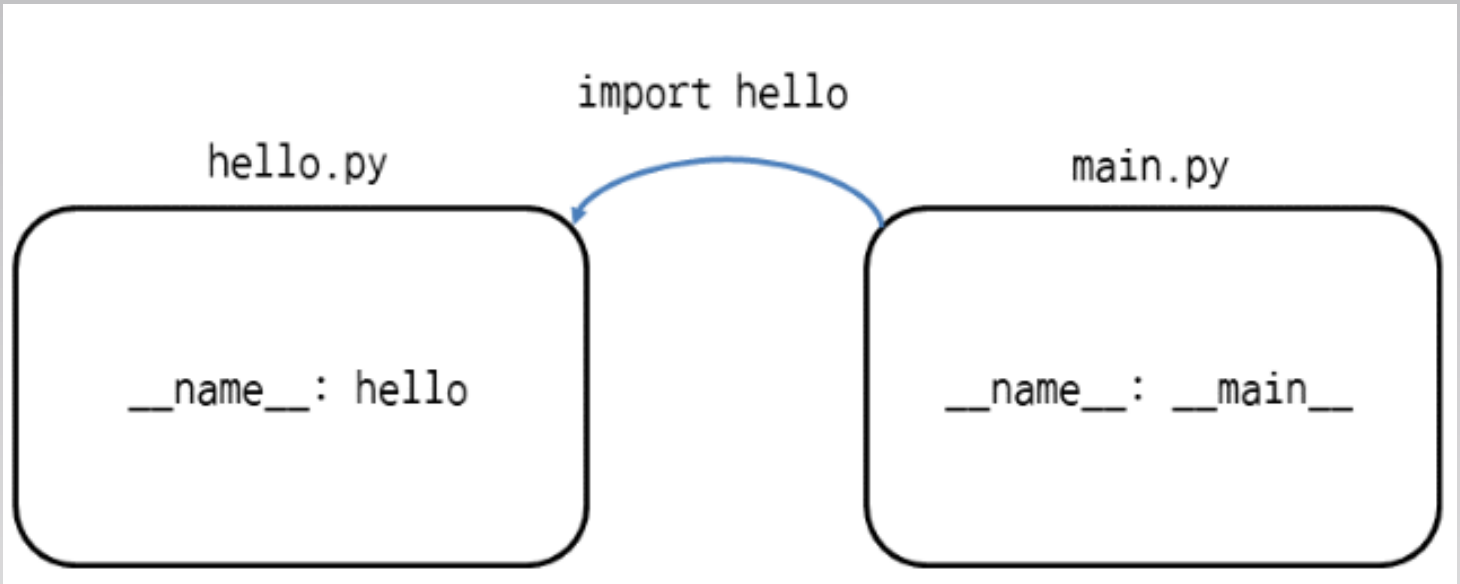
```
import hello    # hello 모듈을 가져옴

print('main.py __name__:', __name__)    # __name__ 변수 출력
```

실행 결과

```
hello 모듈 시작
hello.py __name__: hello
hello 모듈 끝
main.py __name__: __main__
```

45.2 모듈과 시작점 알아보기

▼ 그림 hello.py를 모듈로 가져왔을 때



45.2 모듈과 시작점 알아보기

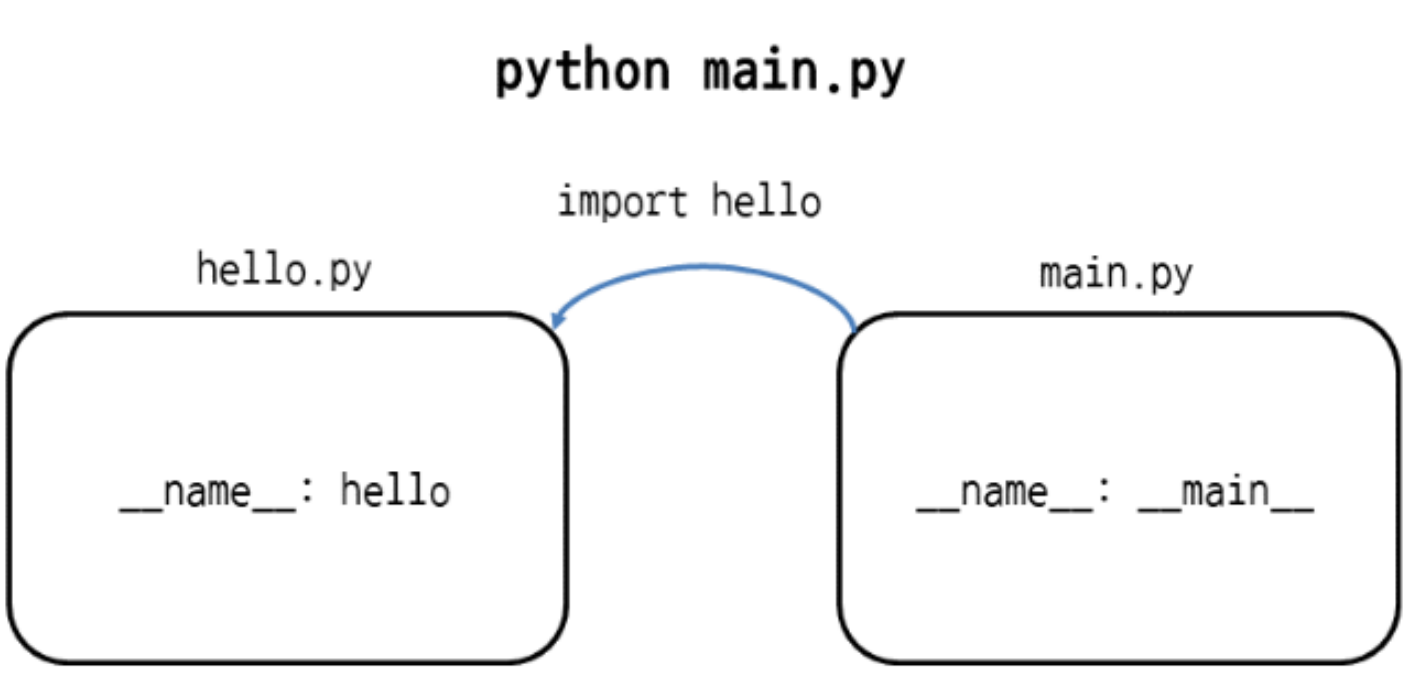
모듈과 시작점 알아보기

- __name__은 모듈의 이름이 저장되는 변수이며 import로 모듈을 가져왔을 때 모듈의 이름이 들어감
- 파이썬 인터프리터로 스크립트 파일을 직접 실행했을 때는 모듈의 이름이 아니라 '__main__'이 들어감(참고로 __name__과 __main__을 헷갈리지 마세요. 같은 네 글자에 알파벳 모양이 비슷해서 헷갈리기 쉬움)
- 콘솔(터미널, 명령 프롬프트)에서 python으로 main.py 파일을 실행해보자(리눅스, macOS에서는 python3 사용)

```
C:\project>python main.py
hello 모듈 시작
hello.py __name__: hello
hello 모듈 끝
main.py __name__: __main__
```

45.2 모듈과 시작점 알아보기

▼ 그림 hello.py를 모듈로 가져왔을 때



45.2 모듈과 시작점 알아보기

» 모듈과 시작점 알아보기

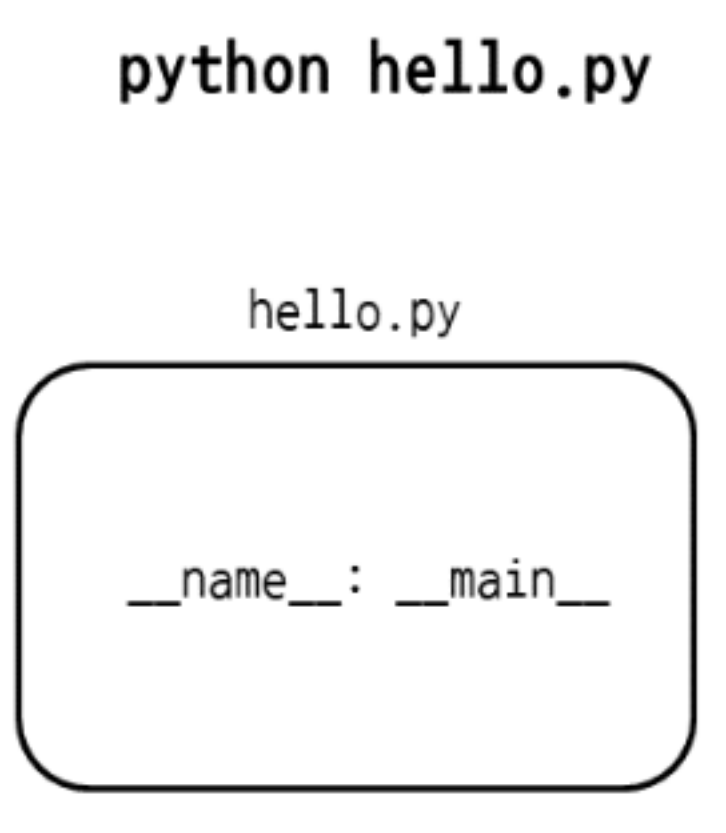
- 다음과 같이 python으로 hello.py 파일을 실행해보면 결과가 조금 달라짐

```
C:\project>python hello.py
hello 모듈 시작
hello.py __name__: __main__
hello 모듈 끝
```

- 어떤 스크립트 파일이든 파이썬 인터프리터가 최초로 실행한 스크립트 파일의 __name__에는 '__main__'이 들어감
- 프로그램의 시작점(entry point)이라는 뜻임

45.2 모듈과 시작점 알아보기

▼ 그림 hello.py를 단독으로 실행했을 때



45.2 모듈과 시작점 알아보기

모듈과 시작점 알아보기

- 파이썬은 최초로 시작하는 스크립트 파일과 모듈의 차이가 없음
- 스크립트 파일이든 시작점도 될 수 있고, 모듈도 될 수 있음
- name__ 변수를 통해 현재 스크립트 파일이 시작점인지 모듈인지 판단함
- if __name__ == '__main__':처럼 __name__ 변수의 값이 '__main__'인지 확인하는 코드는 현재 스크립트 파일이 프로그램의 시작점이 맞는지 판단하는 작업임
- 스크립트 파일이 메인 프로그램으로 사용될 때와 모듈로 사용될 때를 구분하기 위한 용도임

45.2 모듈과 시작점 알아보기

» 스크립트 파일로 실행하거나 모듈로 사용하는 코드 만들기

- 다음 내용을 프로젝트 폴더(C:\project) 안에 calc.py 파일로 저장한 뒤 실행해보자

calc.py

```
def add(a, b):
    return a + b

def mul(a, b):
    return a * b

if __name__ == '__main__':    # 프로그램의 시작점일 때만 아래 코드 실행
    print(add(10, 20))
    print(mul(10, 20))
```

실행 결과

```
30
200
```

```
C:\project>python calc.py
30
200
```

45.2 모듈과 시작점 알아보기

» 스크립트 파일로 실행하거나 모듈로 사용하는 코드 만들기

- 그럼 calc.py를 모듈로 사용하고 import로 calc를 가져와보자

```
>>> import calc
>>>
```

- 모듈로 가져왔을 때는 아무것도 출력되지 않은것은__name__ 변수의 값이 '__main__'일 때만 10, 20의 합과 곱을 출력하도록 만들었기 때문임
- 스크립트 파일을 모듈로 사용할 때는 calc.add, calc.mul처럼 함수만 사용하는 것이 목적이므로 10, 20의 합과 곱을 출력하는 코드는 필요가 없음
- 다음과 같이 calc.add와 calc.mul 함수에 원하는 값을 넣어서 사용하면 됨

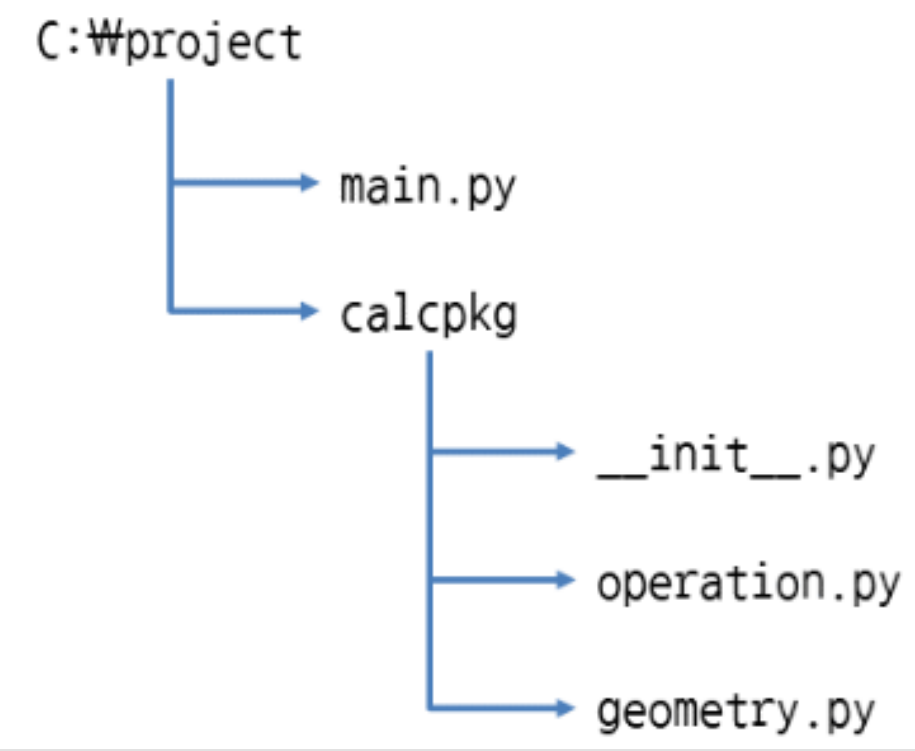
```
>>> calc.add(50, 60)
110
>>> calc.mul(50, 60)
3000
```

45.3 패키지 만들기

패키지 만들기

- 모듈은 스크립트 파일이 한 개지만 패키지는 폴더(디렉터리)로 구성되어 있음

▼ 그림 패키지 폴더 구성



45.3 패키지 만들기

패키지 만들기

- 먼저 프로젝트 폴더(C:\project) 안에 calcpkg 폴더를 만듬

calcpkg/__init__.py

```
# __init__.py 파일은 내용을 비워 둘 수 있음
```

- 폴더(디렉터리) 안에 __init__.py 파일이 있으면 해당 폴더는 패키지로 인식됨

45.3 패키지 만들기

패키지에 모듈 만들기

- 첫 번째 모듈은 덧셈, 곱셈 함수가 들어있는 operation 모듈이고, 두 번째 모듈은 삼각형, 사각형의 넓이 계산 함수가 들어있는 geometry 모듈임

calcpkg/operation.py

```
def add(a, b):
    return a + b

def mul(a, b):
    return a * b
```

calcpkg/geometry.py

```
def triangle_area(base, height):
    return base * height / 2

def rectangle_area(width, height):
    return width * height
```

45.3 패키지 만들기

패키지 사용하기

- 다음 내용을 프로젝트 폴더(C:\project) 안에 main.py 파일로 저장한 뒤 실행해보자(main.py 파일을 calcpkg 패키지 폴더 안에 넣으면 안 됨)

```
import 패키지.모듈
패키지.모듈.변수
패키지.모듈.함수()
  • 패키지.모듈.클래스()
```

main.py

```
import calcpkg.operation    # calcpkg 패키지의 operation 모듈을 가져옴
import calcpkg.geometry     # calcpkg 패키지의 geometry 모듈을 가져옴

print(calcpkg.operation.add(10, 20))    # operation 모듈의 add 함수 사용
print(calcpkg.operation.mul(10, 20))    # operation 모듈의 mul 함수 사용

print(calcpkg.geometry.triangle_area(30, 40))    # geometry 모듈의 triangle_area 함수 사용
print(calcpkg.geometry.rectangle_area(30, 40))   # geometry 모듈의 rectangle_area 함수 사용
```

실행 결과

```
30
200
600.0
1200
```

45.3 패키지 만들기

from import로 패키지의 모듈에서 변수, 함수, 클래스 가져오기

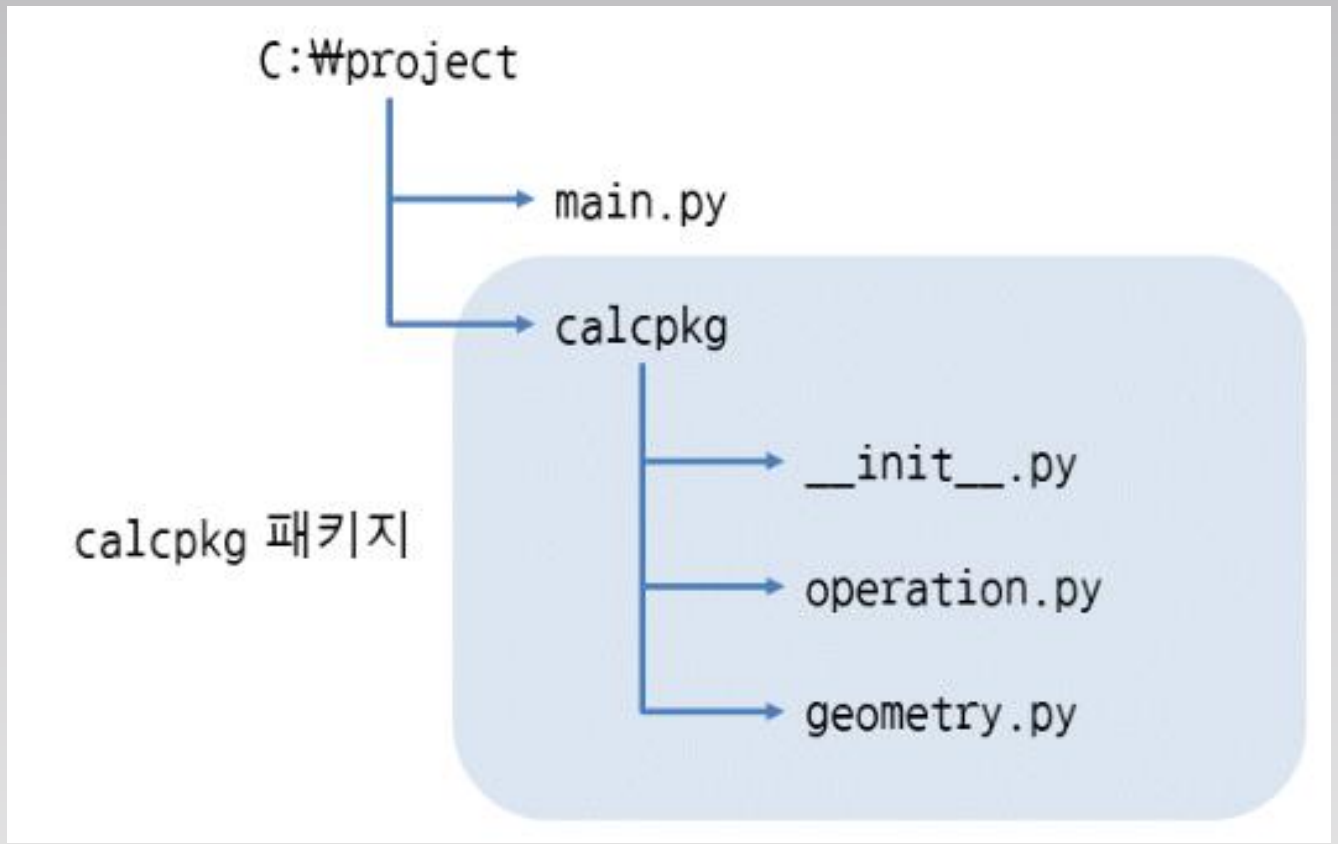
- 패키지의 모듈에서 from import로 함수(변수, 클래스)를 가져온 뒤 패키지와 모듈 이름을 붙이지 않고 사용할 수도 있음

- from 패키지.모듈 import 변수
- from 패키지.모듈 import 함수
- from 패키지.모듈 import 클래스

```
>>> from calcpkg.operation import add, mul
>>> add(10, 20)
30
>>> mul(10, 20)
200
```

45.3 패키지 만들기

▼ 그림 calcpkg 패키지의 계층



45.4 패키지에서 from import 응용하기

패키지에서 from import 응용하기

- import calcpkg처럼 import 패키지 형식으로 패키지만 가져와서 모듈을 사용할때는 calcpkg 패키지의 __init__.py 파일을 다음과 같이 수정함

• from . import 모듈

calcpkg/__init__.py

```
from . import operation    # 현재 패키지에서 operation 모듈을 가져옴
from . import geometry     # 현재 패키지에서 geometry 모듈을 가져옴
```

- 파이썬에서 __init__.py 파일은 폴더(디렉터리)가 패키지로 인식되도록 하는 역할도 하고, 이름 그대로 패키지를 초기화하는 역할도 함
- .(점)은 현재 패키지라는 뜻임

45.4 패키지에서 from import 응용하기

패키지에서 from import 응용하기

main.py

```
import calcpkg    # calcpkg 패키지만 가져옴

print(calcpkg.operation.add(10, 20))    # operation 모듈의 add 함수 사용
print(calcpkg.operation.mul(10, 20))    # operation 모듈의 mul 함수 사용

print(calcpkg.geometry.triangle_area(30, 40))    # geometry 모듈의 triangle_area 함수 사용
print(calcpkg.geometry.rectangle_area(30, 40))   # geometry 모듈의 rectangle_area 함수 사용
```

실행 결과

```
30
200
600.0
1200
```

- calcpkg의 __init__.py에서 하위 모듈을 함께 가져오게 만들었으므로 import calcpkg로 패키지만 가져와도 calcpkg.operation.add(10, 20)처럼 사용할 수 있음

45.4 패키지에서 from import 응용하기

from import로 패키지에 속한 모든 변수, 함수, 클래스 가져오기

- main.py에서 import calcpkg를 from calcpkg import *와 같이 수정하고, 각 함수들도 앞에 붙은 calcpkg.operation, calcpkg.geometry를 삭제한 뒤 실행해보자

• from 패키지 import *

main.py

```
from calcpkg import *    # calcpkg 패키지의 모든 변수, 함수, 클래스를 가져옴

print(add(10, 20))    # operation 모듈의 add 함수 사용
print(mul(10, 20))    # operation 모듈의 mul 함수 사용

print(triangle_area(30, 40))    # geometry 모듈의 triangle_area 함수 사용
print(rectangle_area(30, 40))   # geometry 모듈의 rectangle_area 함수 사용
```

실행 결과

```
Traceback (most recent call last):
  File "C:\project\main.py", line 3, in <module>
    print(add(10, 20))    # operation 모듈의 add 함수 사용
NameError: name 'add' is not defined
```

45.4 패키지에서 from import 응용하기

from import로 패키지에 속한 모든 변수, 함수, 클래스 가져오기

- 실행을 해보면 add가 정의되지 않았다면서 에러가 발생하는것은 __init__.py에서 모듈만 가져왔을 뿐 모듈 안의 함수는 가져오지 않았기 때문임
- IDLE의 파이썬 프롬프트에서 dir 함수를 호출하여 현재 네임스페이스(namespace, 이름공간)를 확인해보자(main.py 안에서 print(dir())을 호출하고 main.py를 실행해도 됨)

```
>>> dir()
['__annotations__', '__builtins__', '__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec_
_', 'geometry', 'operation']
```

- 현재 네임스페이스에는 operation, geometry만 들어있어서 add, mul처럼 함수 이름만으로는 호출할 수가 없음

45.4 패키지에서 from import 응용하기

from import로 패키지에 속한 모든 변수, 함수, 클래스 가져오기

- 현재 패키지(calcpkg)라는 것을 명확하게 나타내기 위해 모듈 앞에 .(점)을 붙임

- from .모듈 import 변수, 함수, 클래스

calcpkg/__init__.py

```
# 현재 패키지의 operation, geometry 모듈에서 각 함수를 가져옴
from .operation import add, mul
from .geometry import triangle_area, rectangle_area
```

실행 결과

```
30
200
600.0
1200
```

45.4 패키지에서 from import 응용하기

from import로 패키지에 속한 모든 변수, 함수, 클래스 가져오기

- __init__.py 파일에서 특정 함수(변수, 클래스)를 지정하지 않고 *을 사용해서 모든 함수(변수, 클래스)를 가져와도 상관없음

```
• from .모듈 import *
```

calcpkg/__init__.py

```
from .operation import *    # 현재 패키지의 operation 모듈에서 모든 변수, 함수, 클래스를 가져옴
from .geometry import *     # 현재 패키지의 geometry 모듈에서 모든 변수, 함수, 클래스를 가져옴
```

- 패키지의 __init__.py에서 from .모듈 import 변수, 함수, 클래스 또는 from .모듈 import * 형식으로 작성했다면 패키지를 가져오는 스크립트에서는 패키지.함수() 형식으로 사용할 수 있음(변수, 클래스도 같은 형식)

45.4 패키지에서 from import 응용하기

from import로 패키지에 속한 모든 변수, 함수, 클래스 가져오기

- import calcpkg와 같이 패키지만 가져오면 됨

- import 패키지
- 패키지.변수
- 패키지.함수()
- 패키지.클래스()

main.py

```
import calcpkg    # calcpkg 패키지만 가져옴

print(calcpkg.add(10, 20))   # 패키지.함수 형식으로 operation 모듈의 add 함수 사용
print(calcpkg.mul(10, 20))   # 패키지.함수 형식으로 operation 모듈의 mul 함수 사용

print(calcpkg.triangle_area(30, 40)) # 패키지.함수 형식으로 geometry 모듈의 triangle_area 함수 사용
print(calcpkg.rectangle_area(30, 40))# 패키지.함수 형식으로 geometry 모듈의 rectangle_area 함수 사용
```

실행 결과

```
30
200
600.0
1200
```

45.4 패키지에서 from import 응용하기

from import로 패키지에 속한 모든 변수, 함수, 클래스 가져오기

- __init__.py에서 from .모듈 import 변수, 함수, 클래스 또는 from .모듈 import * 형식으로 모듈을 가져오면 calcpkg 패키지의 네임스페이스에는 add, mul, triangle_area, rectangle_area가 들어감
- 모듈을 거치지 않고 calcpkg.add처럼 패키지에서 함수를 바로 사용할 수 있음

45.4 패키지에서 from import 응용하기

▼ 그림 패키지 안의 하위 패키지 계층