세상의 속도를
따라잡고 싶다면

점프 투 파이썬

박응용 지음(위키독스 운영자)

이지스퍼블리싱

05

파이썬 날개 달기

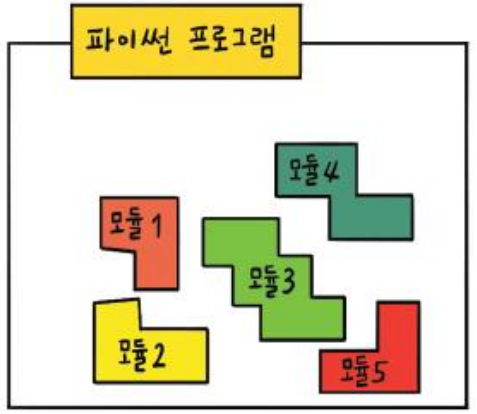
05-2 모듈

05-3 패키지

05-2 모듈

모듈 만들기

- 모듈
  - 함수나 변수 또는 클래스를 모아 놓은 파일
  - 다른 파이썬 프로그램에서 불러와 사용할 수 있도록 만든 파이썬 파일



- add와 sub 함수만 있는 파일(모듈) mod1.py를 만들어 특정 디렉터리에 저장

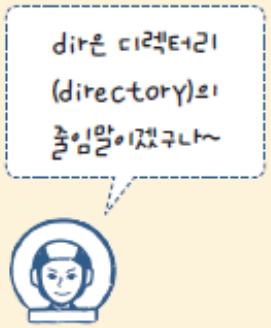
```
def add(a, b):
    return a + b

def sub(a, b):
    return a - b
```

05-2 모듈

▪ 모듈 불러오기

- mod1.py를 저장한 디렉터리로 이동한 다음 대화형 인터프리터 실행
  - 예) 저장한 디렉터리: C:₩doit

```
C:\Users\pahkey>cd C:\doit  <-- 모듈을 저장한 디렉터리로 이동
C:\doit>dir  <-- 디렉터리 안에 mod1.py 파일이 있는지 확인
...
2014-09-23 오후 01:53 49 mod1.py
...
C:\doit>python  <-- 파이썬 인터프리터 실행
>>>
```



05-2 모듈

▪ 모듈 불러오기

- 대화형 인터프리터에서 mod1.py 불러오기

```
>>> import mod1
>>> print(mod1.add(3, 4))
7
>>> print(mod1.sub(4, 2))
2
```

  - 'import mod1'이라고 입력하여, mod1.py 불러오기
  - mod1.py 파일에 있는 add 함수를 사용하려면 모듈 이름(mod1) 뒤에 도트 연산자(.)를 붙이고 함수 이름을 입력

05-2 모듈

모듈 불러오기

import

- 이미 만들어 놓은 파이썬 모듈을 사용할 수 있게 해 주는 명령어
- mod1.py에서 확장자 .py를 제거한 mod1만이 모듈 이름

```
import 모듈_이름
```

- mod1.add, mod1.sub처럼 쓰지 않고 모듈 이름 없이 함수 이름만 쓰고 싶은 경우

```
from 모듈_이름 import 모듈_함수
```

```
>>> from mod1 import add
>>> add(3, 4)
7
```

05-2 모듈

▪ 모듈 불러오기

▪ import

- 모듈의 함수를 여러 개 불러오고 싶은 경우

1) 쉼표(,)로 구분하여 필요한 함수 불러오기

```
from mod1 import add, sub
```

2) * 문자 사용하기

```
from mod1 import *
```

- * 문자는 '모든 것'이라는 뜻으로, 모듈의 모든 함수를 불러와 사용하겠다는 의미

05-2 모듈

if __name__ == "__main__":의 의미

- mod1.py 파일을 다음과 같이 수정

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

def sub(a, b):
    return a - b

print(add(1, 4))
print(sub(4, 2))
```

- 출력한 결과값 확인

```
C:\doit>python mod1.py
5 <- 1+4
2 <- 4-2
```

- 그러나 mod1 모듈을 import할 때 mod1.py 파일이 실행되어 결과값을 출력하는 문제

```
C:\Users\pahkey> cd C:\doit
C:\doit> python
>>> import mod1
5
2
```

05-2 모듈

if __name__ == "__main__":의 의미

- mod1.py 파일을 if __name__ == "__main__":을 사용하여 다음과 같이 수정

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

def sub(a, b):
    return a - b

if __name__ == "__main__":
    print(add(1, 4))
    print(sub(4, 2))
```

  - C:₩doit>python mod1.py처럼 직접 파일을 실행했을 경우
    - __name__ 변수에 __main__ 값이 저장
    - __name__ == "__main__"이 참이 되어 if 문 다음 문장이 수행됨

  - 대화형 인터프리터나 다른 파일에서 이 모듈을 불러와 사용할 경우
    - __name__ 변수에 모듈 이름 mod1이 저장
    - __name__ == "__main__"이 거짓이 되어 if 문 다음 문장이 수행되지 않음

```
>>> import mod1
>>>
```

05-2 모듈

▪ 클래스나 변수 등을 포함한 모듈

- mod2.py 파일 만들어서 저장하기
  - 원의 넓이를 계산하는 Math 클래스
  - 두 값을 더하는 add 함수
  - 원주율 값에 대항되는 PI

```
PI = 3.141592

class Math:
    def solv(self, r):
        return PI * (r ** 2)

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

- 대화형 인터프리터를 열어 다음과 같이 확인

```
C:\Users\pahkey> cd C:\doit
C:\doit> python
>>> import mod2
>>> print(mod2.PI) <— PI 변수 사용
3.141592
```

```
>>> a = mod2.Math() <— Math 클래스 사용
>>> print(a.solv(2))
12.566368
```

```
>>> print(mod2.add(mod2.PI, 4.4)) <— add 함수 사용
7.541592
```

05-2 모듈

다른 파일에서 모듈 불러오기

- mod2.py 파일을 다른 파이썬 파일 modtest.py에서 불러와 사용하기
  - 대화형 인터프리터에서 한 것과 마찬가지로,
    mod2.py와 동일한 경로에 modtest.py를 만들고 import mod2로 mod2 모듈을 불러올 수 있음

```
import mod2
result = mod2.add(3, 4)
print(result)
```

05-2 모듈

▪ 다른 디렉터리에 있는 모듈을 불러오는 방법

- 이전에 만든 mod2.py 파일을 C:₩doit₩mymod로 이동

```
C:\Users\pahkey>cd C:\doit
C:\doit>mkdir mymod  <-- mymod 디렉터리 생성
C:\doit>move mod2.py mymod  <-- 지정한 디렉터리로 파일 이동
        1개 파일을 이동했습니다.
```

- sys.path.append 사용하기
  - sys 모듈은 파이썬을 설치할 때 함께 설치되는 라이브러리 모듈
  - sys.path로 파이썬 라이브러리가 설치되어 있는 디렉터리 목록 확인

```
C:\doit>python
>>> import sys
```

```
>>> sys.path
['', 'C:\\Windows\\SYSTEM32\\python311.zip', 'c:\\Python311\\DLLs',
'c:\\Python311\\lib', 'c:\\Python311', 'c:\\Python311\\lib\\site-packages']
```

05-2 모듈

▪ 다른 디렉터리에 있는 모듈을 불러오는 방법

- 이전에 만든 mod2.py 파일을 C:₩doit₩mymod로 이동

```
C:\Users\pahkey>cd C:\doit
C:\doit>mkdir mymod  <— mymod 디렉터리 생성
C:\doit>move mod2.py mymod  <— 지정한 디렉터리로 파일 이동
        1개 파일을  이동했습니다.
```

- sys.path.append 사용하기
  - sys.path에 C:₩doit₩mymod 디렉터리 추가

```
>>> sys.path.append("C:/doit/mymod")
>>> sys.path
['', 'C:\\Windows\\SYSTEM32\\python311.zip', 'c:\\Python311\\DLLs',
'c:\\Python311\\lib', 'c:\\Python311', 'c:\\Python311\\lib\\site-packages',
'C:/doit/mymod']
```

  - sys.path 출력 결과 확인

```
>>> import mod2
>>> print(mod2.add(3, 4))
7
```

05-2 모듈

▪ 다른 디렉터리에 있는 모듈을 불러오는 방법

- 이전에 만든 mod2.py 파일을 C:₩doit₩mymod로 이동

```
C:\Users\pahkey>cd C:\doit
C:\doit>mkdir mymod  <-- mymod 디렉터리 생성
C:\doit>move mod2.py mymod  <-- 지정한 디렉터리로 파일 이동
        1개 파일을 이동했습니다.
```

- PYTHONPATH 환경 변수 사용하기
  - set 명령어를 사용해 PYTHONPATH 환경 변수에 mod2.py 파일이 있는 C:₩doit₩mymod 디렉터리를 설정

```
C:\doit>set PYTHONPATH=C:\doit\mymod
C:\doit>python
>>> import mod2
>>> print(mod2.add(3, 4))
7
```

05-3 패키지

패키지(packages)란?

- 관련 있는 모듈의 집합
- 파이썬 모듈을 계층적(디렉터리 구조)로 관리할 수 있게 해 줌

- 파이썬 패키지는 디렉터리와 파이썬 모듈로 이루어짐

- 패키지 사용의 장점
  1) 간단한 파이썬 프로그램이 아니라면 패키지 구조로 파이썬 프로그램을 만드는 것이 공동 작업이나 유지 보수 등 여러 면에서 유리함
  2) 패키지 구조로 모듈을 만들면 다른 모듈과 이름이 겹치더라도 더 안전하게 사용 가능

05-3 패키지

패키지(packages)란?

- 가상의 game 패키지 예
  - game, sound, graphic, play는 디렉터리 이름
  - 확장자가 .py인 파일은 파이썬 모듈

  - game 디렉터리가 이 패키지의 루트 디렉터리
  - sound, graphic, play는 서브 디렉터리

```
game/
    __init__.py
    sound/
        __init__.py
        echo.py
        wav.py
    graphic/
        __init__.py
        screen.py
        render.py
    play/
        __init__.py
        run.py
        test.py
```

05-3 패키지

▪ 패키지 만들기

1) C:₩doit 디렉터리 밑에 game 및 기타 서브 디렉터리 및 .py 파일 생성

```
C:/doit/game/__init__.py
C:/doit/game/sound/__init__.py
C:/doit/game/sound/echo.py
C:/doit/game/graphic/__init__.py
C:/doit/game/graphic/render.py
```

2) 각 디렉터리에 __init__.py 파일 생성
   - 내용은 비워둠

3) echo.py 파일 내용 작성

```
def echo_test():
    print("echo")
```

05-3 패키지

▪ 패키지 만들기

4) render.py 파일 내용 작성

```
def render_test():
    print("render")
```

5) 생성한 game 패키지를 참조할 수 있도록
명령 프롬프트 창에서 set 명령어로 PYTHONPATH 환경 변수에 C:₩doit 디렉터리 추가

```
C:\>set PYTHONPATH=C:/doit
C:\>python
>>>
```

05-3 패키지

▪ 패키지 안의 함수 실행하기

1) echo 모듈을 import하여 실행하는 방법

```
>>> import game.sound.echo
>>> game.sound.echo.echo_test()
echo
```

2) echo 모듈이 있는 디렉터리를 from ... import하여 실행하는 방법

```
>>> exit()  <-- 인터프리터 종료
C:\>python  <-- 인터프리터 재시작
>>> from game.sound import echo
>>> echo.echo_test()
echo
```

3) echo 모듈의 echo_test 함수를 직접 import하여 실행하는 방법

```
>>> from game.sound.echo import echo_test
>>> echo_test()
echo
```

05-3 패키지

▪ 패키지 안의 함수 실행하기

- echo_test 함수를 사용하는 것이 불가능한 경우
  - import game을 수행하면 game 디렉터리의 __init__.py에 정의한 것만 참조 가능

```
>>> import game
>>> game.sound.echo.echo_test()
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
AttributeError: 'module' object has no attribute 'sound'
```

  - 도트 연산자(.)를 사용해서 import할 때 가장 마지막 항목은 반드시 모듈 또는 패키지여야만 함

```
>>> import game.sound.echo.echo_test
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
 ModuleNotFoundError: No module named 'game.sound.echo.echo_test'; 'game.sound.
echo' is not a package
```

05-3 패키지

__init__.py의 용도

- 해당 디렉터리가 패키지의 일부임을 알려 주는 역할
- 패키지와 관련된 설정이나 초기화 코드를 포함할 수 있음

- 패키지 변수 및 함수 정의

```
VERSION = 3.5

def print_version_info():
    print(f"The version of this game is {VERSION}.")
```

```
>>> import game
>>> print(game.VERSION)
3.5
>>> game.print_version_info()
The version of this game is 3.5.
```

05-3 패키지

__init__.py의 용도

- 패키지 내 모듈을 미리 import

```
from .graphic.render import render_test

VERSION = 3.5

def print_version_info():
    print(f"The version of this game is {VERSION}.")
```

```
>>> import game
>>> game.render_test()
render
```

05-3 패키지

init.py의 용도

패키지 초기화

```
from .graphic.render import render_test

VERSION = 3.5

def print_version_info():
    print(f"The version of this game is {VERSION}.")

# 여기에 패키지 초기화 코드를 작성한다.
print("Initializing game ...")
```

```
>>> import game
Initializing game ...
>>>
```

```
>>> from game.graphic.render import render_test
Initializing game ...
>>>
```

```
>>> import game
Initializing game ...
>>> from game.graphic.render import render_test
>>>
```

05-3 패키지

- __init__.py의 용도
  - __all__

```
>>> from game.sound import *
Initializing game ...
>>> echo.echo_test()
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
NameError: name 'echo' is not defined
```

    - echo라는 이름이 정의되지 않았다는 오류

  - 해당 디렉터리의 __init__.py 파일에 __all__ 변수 설정

```
__all__ = ['echo']
```

    - __all__은 sound 디렉터리에서 *를 사용하여 import할 경우, 이곳에 정의된 echo 모듈만 import된다는 의미
  - 다시 실행 후 예상 결과 출력 확인

```
>>> from game.sound import *
Initializing game ...
>>> echo.echo_test()
echo
```

05-3 패키지

relative 패키지

- graphic 디렉터리의 render.py 모듈에서 sound 디렉터리의 echo.py 모듈을 사용하고 싶을 때

```
from game.sound.echo import echo_test
def render_test():
    print("render")
    echo_test()
```

  - from game.sound.echo import echo_test 문장을 추가하여 echo_test 함수를 사용할 수 있도록 수정

```
>>> from game.graphic.render import render_test
Initializing game ...
>>> render_test()
render
echo
```

05-3 패키지

relative 패키지

- graphic 디렉터리의 render.py 모듈에서 sound 디렉터리의 echo.py 모듈을 사용하고 싶을 때

```
from game.sound.echo import echo_test
def render_test():
    print("render")
    echo_test()
```

- from ..sound.echo import echo_test로 수정

```
from ..sound.echo import echo_test
def render_test():
    print("render")
    echo_test()
```

- ..은 render.py 파일의 부모 디렉터리를 의미

- relative한 접근자의 종류

접근자	설명
..	부모 디렉터리를 의미한다.
.	현재 디렉터리를 의미한다.

감사합니다.