07장 모듈

- 모듈이란 함수나 변수 또는 클래스들을 모아 놓은 파일이다.
- 모듈은 다른 파이썬 프로그램에서 불러와 사용할 수 있게끔 만들어진 파이썬 파일이라고도 할 수 있다.

7.1 모듈 만들기

- 파이썬에선 많은 내장 모듈을 지원하지만 필요에 따라 사용자가 직접 모듈을 생성할 수도 있다.
- 모듈은 함수뿐만 아니라 클래스나 변수 등을 포함할 수도 있다.
- 모듈은 <모듈이름>.py이라는 이름을 갖는 파일로 만든다.

[ch07_module/mod1.py]

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

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

print(add(1, 0))  # 1
print(sub(3, 1))  # 2

'''
if __name__ == "__main__":
    print(add(1, 0))
    print(sub(3, 1))
'''
```

7.2 모듈 불러오기

- 모듈 함수를 사용하는 방법
  - import 모듈
  - import 모듈 as 별칭
  - from 모듈 import 함수
  - from 모듈 import *

[ch07_module/ex02_import.py]

```
# import 모듈
import mod1
print("="*10)
print(mod1.add(3,4))  # 7
print(mod1.sub(4,2))  # 2

# from 모듈 이름 import 모듈 함수
from mod1 import add
print(add(4,1)) # 5
# print(sub(4,2))
# NameError: name 'sub' is not defined

from mod1 import *
print(sub(7,1)) # 6
```

7.3 if __name__ == '__main__':의 의미

- 파이썬 파일을 윈도우 프롬프트에서 직접 실행할 때와 임포트됐을 때가 각기 다르게 동작한다.
- __name__ 변수는 파이썬이 내부적으로 읽을 수만 있는 특별한 변수 이름으로 자기 모듈의 이름을 나타낸다. 하지만 직접 실행할 때는 모듈의 이름이 아니라 __main__이라는 값이 저장된다. 즉 모듈의 코드가 메인으로 실행됐다는 뜻이다.
- __name__을 이용해 구분하는 방법은 모듈을 개발할 때 많이 사용한다. 모듈을 수정하거나 코드를 추가할 때마다 임포트해서 테스트 코드를 수행하기보다 직접 실행하여 바로 테스트 결과를 확인할 수 있기 때문에 개발 효율이 높아진다.

[ch07_module/mod1.py]

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

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

'''
print(add(1, 0))  # 1
print(sub(3, 1))  # 2
'''

if __name__ == "__main__":
    print(add(1, 0))
    print(sub(3, 1))
```

[ch07_module/ex03_main.py]

```
from mod1 import add
print(add(4, 3))
# 7
```

7.4 클래스나 변수 등을 포함한 모듈

- 지금까지 살펴 본 모듈은 함수만 포함했지만 클래스나 변수 등을 포함할 수도 있다.

[ch07_module/mod2.py]

```
# 모듈 구성요소: 변수, 함수, 클래스
PI = 3.141592  # 변수

class Math:    # 클래스
    def solve(self, r):  # 클래스 메서드(함수)
        return PI * (r**2)

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

[ch07_module/ex04_import.py]

```
import mod2

print(mod2.PI)
# 3.141592

a = mod2.Math()
print(a.solve(2))
# 12.566368

print(mod2.add(mod2.PI, 4.4))
# 7.5415920000000005
```

[꿀팁] 모듈을 불러오는 방법(모듈 경로 설정하기)

```
# 1. 먼저 이전에 만든 mod2.py 파일을 c:\dev\mymod\로 복사시킨다.


# 2. 임포트를 하면 인터프리터는 현재 작업 디렉토리에서 해당 모듈을 찾는다.
C:\Users\won>python
Python 3.7.1 (default, Dec 10 2018, 22:54:23) [MSC v.1915 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import mod2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'mod2'
>>> quit()

C:\Users\won>cd c:\dev\mymod

c:\dev\mymod>python
Python 3.7.1 (default, Dec 10 2018, 22:54:23) [MSC v.1915 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import mod2
>>> print(mod2.add(3,4))
7
>>>


# 3. sys.path.append(모듈을 저장한 디렉토리) 사용하기
# 표준 라이브러리 디렉토리
C:\Users\won>python
Python 3.7.1 (default, Dec 10 2018, 22:54:23) [MSC v.1915 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import sys
>>> sys.path
['', 'D:\\prod\\Anaconda3\\python37.zip', 'D:\\prod\\Anaconda3\\DLLs', 'D:\\prod\\Anaconda3\\lib',
'D:\\prod\\Anaconda3'
, 'D:\\prod\\Anaconda3\\lib\\site-packages', 'D:\\prod\\Anaconda3\\lib\\site-packages\\win32',
'D:\\prod\\Anaconda3\\lib
\\site-packages\\win32\\lib', 'D:\\prod\\Anaconda3\\lib\\site-packages\\Pythonwin']
>>> sys.path.append('c:\\dev\mymod')
>>> sys.path
['', 'D:\\prod\\Anaconda3\\python37.zip', 'D:\\prod\\Anaconda3\\DLLs', 'D:\\prod\\Anaconda3\\lib',
'D:\\prod\\Anaconda3'
, 'D:\\prod\\Anaconda3\\lib\\site-packages', 'D:\\prod\\Anaconda3\\lib\\site-packages\\win32',
'D:\\prod\\Anaconda3\\lib
\\site-packages\\win32\\lib', 'D:\\prod\\Anaconda3\\lib\\site-packages\\Pythonwin', 'c:\\dev\\mymod']
>>> import mod2
>>> print(mod2.add(3, 4))
```

```
7
>>>


# 4. PYTHONPATH 환경변수에 등록된 위치 (옵션)
# Linux에서 다음과 같은 구문을 셸 설정파일(bash셸의 경우는 .bash_profile)에 추가하면 된다.
export PYTHONPATH=$PYTHONPATH:/home/user/dev/mymod

# Windows에서 PYTHONPATH 환경변수를 사용한다.
C:\Users\won>set PYTHONPATH=c:\dev\mymod

C:\Users\won>python
Python 3.7.1 (default, Dec 10 2018, 22:54:23) [MSC v.1915 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import mod2
>>> print(mod2.add(3, 4))
7
```

[과제] 모듈에 있는 클래스 함수를 불러오기

- mod2.py 모듈을 사용해 반지름이 5인 원의 넓이를 계산해 보자.

[ch07_module/verify/ve01_circle.py]

```
import mod2

# 아래에 코드를 작성한다.



print(result)
# 78.5398
```

7.5 패키지

- 패키지(Packages)는 도트(.)를 이용하여 파이썬 모듈을 계층적(디렉터리 구조)으로 관리할 수 있게 해 준다. 예를 들어 모듈명이 A.B인 경우 A는 패키지명이 되고 B는 A패키지의 B 모듈이 된다.
- 파이썬 패키지는 디렉터리와 파이썬 모듈로 이루어지고, 그 구조는 아래와 같이 구성될 수 있다.

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

7.5.1 패키지 만들기

- game, graphic, play(패키지)들을 생성하고 .py 파일(모듈)들을 아래와 같이 만든다.

```
# game/sound/echo.py
def echo_test():
    print("echo")

# game/graphic/render.py
def render_test():
    print("render")
```

[ch07_module/ex05_package.py]

```
# 패키지 안의 함수 실행하기
import game.sound.echo
game.sound.echo.echo_test()
# echo

from game.sound import echo
echo.echo_test()
# echo

from game.sound.echo import echo_test
echo_test()
# echo

from game.graphic.render import render_test
render_test()
'''
render
echo
'''
```

~~7.5.2 __init__.py의 용도~~

- 해당 디렉터리가 패키지의 일부임을 알려주는 역할을 한다. 만약 game, sound, graphic 등 패키지에 포함된 디렉토리에 __init__.py 파일이 없다면 패키지로 인식되지 않는다.
- python3.3 버전부터는 __init__.py 파일 없이도 패키지로 인식이 된다. 하지만 하위 버전 호환을 위해 __init__.py 파일을 생성하는 것이 안전한 방법이다.

```
# __init__.py의 용도
# 시험 삼아 sound 디렉터리의 __init__.py를 제거하고 다음을 수행해 보자.
# python3.3 이전 버전은 __init__.py 파일이 없으면 임포트 오류(ImportError)가 발생하게 된다.
import game.sound.echo
```

~~7.5.3 __all__의 용도~~

- 특정 디렉터리의 모듈을 *를 이용하여 import할 때에는 다음과 같이 해당 디렉터리의 __init__.py 파일에 __all__이라는 변수를 설정하고 import할 수 있는 모듈을 정의해 주어야

한다. 그렇지 않으면 "NameError: name 'echo' is not defined"라는 오류가 발생한다.

- python3.3 버전부터는 정의하지 않아도 상관없다.

```
# sound/__init__.py 파일
__all__=['echo']


# chap06_5.py 파일의 일부
# __all__의 용도
from game.sound import *
echo.echo_test()
```

7.5.4 relative 패키지

- from game.sound.echo import echo_test와 같이 전체 경로를 이용하여 import할 수도 있지만 ..(상위 디렉터리) 혹은 .(현재 디렉터리)과 같이 relative하게 import하는 것도 가능하다.
- ..과 같은 relative한 접근자는 render.py와 같이 모듈 안에서만 사용해야 한다. 파이썬 인터프리터에서 relative한 접근자를 사용하면 "SystemError: cannot perform relative import'와 같은 오류가 발생한다.

[ch07_module/game/graphic/render.py]

```
#-*- coding:utf-8 -*-

# from game.sound.echo import echo_test
from ..sound.echo import echo_test

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