프로그래밍

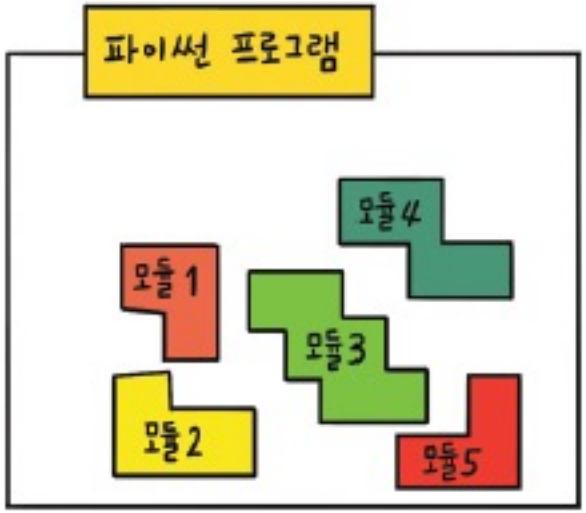
이은섭
(Eunseop Lee)

Computer Science and Engineering,
Pohang University of Science and Technology
eseop90@gmail.ac.kr

모듈

모듈이란?

- 모듈이란 함수나 변수 또는 클래스를 모아 놓은 파이썬 파일로 다른 파이썬 프로그램에서 불러와 사용할 목적으로 만듬



모듈 만들기

- add와 sub 함수만 있는 파일 'mod1.py'를 만들고 특정 디렉토리에 저장

```
# mod1.py
def add(a, b):
    return a + b

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

모듈 불러오기

- mod1.py 모듈을 불러오기 위해 'import mod1'이라고 입력

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

- mod1.add, mod1.sub처럼 쓰지 않고 add, sub처럼 쓰고 싶은 경우

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

모듈 불러오기

- import의 사용 방법

```
import 모듈_이름

# 예
import mod1
```

- 모듈 이름 없이 함수 이름만 쓰고 싶은 경우

```
from 모듈_이름 import 모듈_함수

# 예
from mod1 import add, sub
```

- 모듈의 모든 함수를 불러와 사용하고 싶은 경우

```
from 모듈_이름 import *

# 예
from mod1 import *
```

클래스, 함수, 변수를 포함한 모듈

- 모듈은 클래스, 함수, 변수를 포함할 수 있음

```python
# mod2.py
PI = 3.141592

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

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

클래스, 함수, 변수를 포함한 모듈

- 변수

```
>>> import mod2
>>> print(mod2.PI)
3.141592
```

- 클래스

```
>>> a = mod2.Math()
>>> print(a.solv(2))
12.566368
```

- 함수

```
>>> print(mod2.add(mod2.PI, 4.4))
7.541592
```

if __name__ == "__main__":의 의미

- mod1.py 파일을 다음과 같이 수정 후 실행

```python
# mod1.py
def add(a, b):
    return a+b

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

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

```
# terminal
> python mod1.py
5
2
```

if __name__ == "__main__":의 의미

- mod1.py 파일의 add와 sub 함수를 사용하기 위해 mod1 모듈을 import

```
>>> import mod1
5
2
```

  - import mod1을 수행하는 순간 mod1.py 파일이 실행되어 결과값을 출력

if __name__ == "__main__":의 의미

- 이러한 문제를 방지하려면 mod1.py 파일을 다음처럼 수정

```python
# mod1.py
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))
```

  - 직접 이 파일을 실행했을 때(python mod1.py) __name__ == "__main__"이 참이 되어 if 문 다음 문장이 수행
  - 인터프리터나 다른 파일에서 이 모듈을 불러 사용할 때는 __name__ == "__main__"이 거짓이 되어 if 문 다음 문장이 수행되지 않음

다른 디렉토리에 있는 모듈을 불러오는 방법

- 특정 모듈을 사용하기 위해서는, 동일한 디렉토리에 소스 파일이 존재해야함
- sys.path.append 사용하여 다른 디렉토리에 있는 모듈을 사용할 수 있음

```
>>> import sys
>>> sys.path
['', '/Users/eseop/opt/anaconda3/lib/python39.zip', '/Users/eseop/opt/anaconda3/lib/python3.9
', '/Users/eseop/opt/anaconda3/lib/python3.9/lib-dynload', '/Users/eseop/opt/anaconda3/lib/py
thon3.9/site-packages', '/Users/eseop/opt/anaconda3/lib/python3.9/site-packages/aeosa', '/Use
rs/eseop/opt/anaconda3/lib/python3.9/site-packages/locket-0.2.1-py3.9.egg']
>>> import mod1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'mod1'
```

- sys.path는 파이썬 라이브러리가 설치되어 있는 디렉터리 목록을 보여줌
- 이 디렉터리 안에 저장된 파이썬 모듈은 모듈이 저장된 디렉터리로 이동할 필요 없이 바로 불러 사용할 수 있음
- mod1은 '/Users/eseop/Documents/Workspace/test2'에 존재하기에 현재 디렉토리(/Users/eseop)에서 import하면 오류가 발생

다른 디렉토리에 있는 모듈을 불러오는 방법

- 다른 디렉터리에 있는 모듈을 사용하기 위해서는 'sys.path.append'를 사용

```
>>> sys.path.append('/Users/eseop/Documents/Workspace/test2')
>>> sys.path
['', '/Users/eseop/opt/anaconda3/lib/python39.zip', '/Users/eseop/opt/anaconda3/lib/python3.9
', '/Users/eseop/opt/anaconda3/lib/python3.9/lib-dynload', '/Users/eseop/opt/anaconda3/lib/py
thon3.9/site-packages', '/Users/eseop/opt/anaconda3/lib/python3.9/site-packages/aeosa', '/Use
rs/eseop/opt/anaconda3/lib/python3.9/site-packages/locket-0.2.1-py3.9.egg', '/Users/eseop/Doc
uments/Workspace/test2'
>>> import mod1
>>> mod1.add(3, 1)
4
>>> mod1.sub(4, 2)
2
```

패키지

패키지란?

- 패키지(packages)란 관련 있는 모듈의 집합
- 패키지는 파이썬 모듈을 계층적(디렉터리 구조)으로 관리할 수 있게 해 준다.

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

패키지 만들기

- 패키지인 game과 패키지를 실행시킬 main 디렉터리를 각각 생성하고, 다음과 같이 서브 디렉터리 및 파이썬 파일들을 생성

```
game/__init__.py
game/sound/__init__.py
game/sound/echo.py
game/graphic/__init__.py
game/graphic/render.py
```

- 각 디렉터리에 빈 __init__.py 파일을 만듬

패키지 만들기

- echo.py 파일

```python
# echo.py
def echo_test():
    print("echo")
```

- render.py 파일

```python
# render.py
def render_test():
    print("render")
```

패키지 만들기

- main.py에서 패키지를 참조할 수 있도록 'sys.path.append'를 이용하여 game 패지키를 추가

```
import sys
sys.path.append('../')
```

패키지 안의 함수 실행하기

- 패키지 안의 함수를 실행하는 방법에는 3가지가 존재

1. 모듈을 'import'하여 실행

```
import sys
sys.path.append('../')

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

패키지 안의 함수 실행하기

- 패키지 안의 함수를 실행하는 방법에는 3가지가 존재

2. 모듈이 있는 디렉터리까지를 'from ... import'하여 실행하는 방법

```
import sys
sys.path.append('../')

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

패키지 안의 함수 실행하기

- 패키지 안의 함수를 실행하는 방법에는 3가지가 존재

3. 모듈의 함수를 직접 import하여 실행하는 방법

```
import sys
sys.path.append('../')

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

__init__.py의 용도

- __init__.py 파일은 해당 디렉터리가 패키지의 일부임을 알려 주는 역할을 함
  - 만약 game, sound, graphic 등 패키지에 포함된 디렉터리에 __init__.py 파일이 없다면 패키지로 인식되지 않음
  - python 3.3 버전부터는 __init__.py 파일이 없어도 패키지로 인식함.
  - 하지만 하위 버전 호환을 위해 __init__.py 파일을 생성하는 것이 안전한 방법
- 또한 __init__.py 파일은 패키지와 관련된 설정이나 초기화 코드를 포함할 수 있음

패키지 변수 및 함수 정의

- 패키지 수준에서 변수와 함수를 정의할 수 있음
  - 예를 들어, game 패키지의 __init__.py 파일에 공통 변수나 함수를 정의할 수 있음

```
# game/__init__.py
VERSION = 3.5

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

  - 패키지의 __init__.py 파일에 정의된 변수와 함수는 다음과 같이 사용할 수 있음

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

패키지 내 모듈을 미리 import

- __init__.py 파일에 패키지 내의 다른 모듈을 미리 import하여 패키지를 사용하는 코드에서 간편하게 접근할 수 있음

```
# game/__init__.py
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
```

패키지 초기화

- __ini__.py 파일에 패키지를 처음 불러올 때 실행되어야 하는 코드를 작성할 수 있음
  - 예) 데이터베이스 연결, 설정 파일 로드 등

```
# game/__init__.py
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 ...
>>>
```

패키지 초기화

- 패키지의 초기화 코드는 패키지의 하위 모듈의 함수를 import할 경우에도 실행됨

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

- 단, 초기화 코드는 한 번 실행된 후에는 다시 import를 수행하더라도 실행되지 않음
  - game 패키지를 import한 후에 하위 모듈을 다시 import 하더라도 초기화 코드는 처음 한 번만 실행됨.

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

패키지 초기화

- 패키지의 초기화 코드는 패키지의 하위 모듈의 함수를 import할 경우에도 실행됨

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

- 단, 초기화 코드는 한 번 실행된 후에는 다시 import를 수행하더라도 실행되지 않음
  - game 패키지를 import한 후에 하위 모듈을 다시 import 하더라도 초기화 코드는 처음 한 번만 실행됨.

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

패키지 초기화

```
# __init__.py
print('Initializing game package')
```

```
# graphic/__init__.py
print('Initializing game/graphic package')
```

- '>>> from game.graphic.render import render_test'를 실행시키면?

패키지 초기화

```
# __init__.py
print('Initializing game package')
```

```
# graphic/__init__.py
print('Initializing game/graphic package')
```

- '>>> from game.graphic.render import render_test'를 실행시키면?

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

  - game의 __init__.py를 먼저 확인 후, game/graphic의 __init__.py를 확인함

__all__

- 다음을 실행해보자

```
>>> 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
```

- game.sound 패키지에서 모든 것(*)을 import했으므로 echo 모듈을 사용할 수 있어야 할 것 같은데, echo라는 이름이 정의되지 않았다는 오류가 발생

__all__

- 특정 디렉터리의 모듈을 *를 사용하여 import할 때는 해당 디렉터리의 __init__.py 파일에 __all__ 변수를 설정하고 import할 수 있는 모듈을 정의해 주어야 함

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

  - 여기에서 __all__이 의미하는 것은 sound 디렉터리에서 *를 사용하여 import할 경우, 이곳에 정의된 echo 모듈만 import된다는 의미

- __init__.py 파일을 변경한 후 예제를 수행하면 원하는 결과가 출력

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

relative 패키지

- 만약 graphic 디렉터리의 render.py 모듈에서 sound 디렉터리의 echo.py 모듈을 사용하고 싶다면?

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

- 다음과 같이 relative하게 import하는 것도 가능

```
# render.py
from ..sound.echo import echo_test

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

  - '..'은 render.py 파일의 부모 디렉터리(game)를 의미

relative 패키지

- 만약 graphic 디렉터리의 render.py 모듈에서 동일 디렉터리의 render2.py 모듈을 사용하고 싶다면?

```
# game/graphic/render2.py
def render_test2():
    print("render2")
```

```
# game/graphic/render.py
from ..sound.echo import echo_test
from .render2 import render_test2

def render_test():
    print("render")
    echo_test()
    render_test2()
```

  - '..'은 render.py 파일의 부모 디렉터리(game)를 의미
  - '.'은 render.py 파일의 현재 디렉터리(graphic)를 의미

Q&A