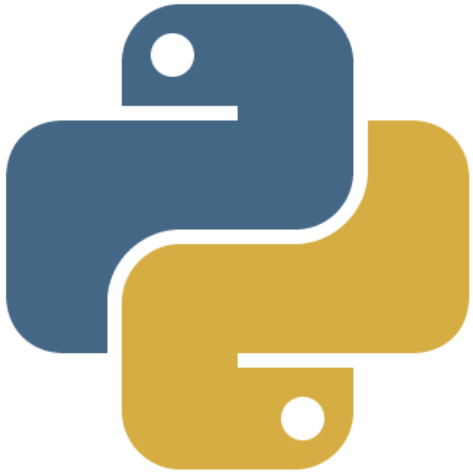
Programming in Python

12 – 모듈과 네임스페이스, 패키지



python

2016년 8월, 국민대학교 컴퓨터공학부

Python 모듈 (Modules)

Python 모듈의 기본 사항들

- 모듈의 작성
  - Python 코드를 작성하여 .py 의 파일 이름으로 저장한다.
  - 모듈 내 최상위 수준 (top-level) 에서 정의된 객체는 모두 이 모듈을 사용하는 클라이언트에 노출
  - 모듈 이름은 변수 이름 규칙에 부합해야 함
    - Python 프로그램에서 import 하면 해당 이름은 변수 (객체) 이름으로 취급되기 때문

- 모듈의 이용
  - import 문장
  - from … import 문장

모듈의 이용 - import

- import 문장을 이용

```
>>> import module1
>>> module1.printer('Hello world!')
Hello world!
```

module1.py

```
def printer(x):
    print(x)
```

```
>>> module1
<module 'module1' from 'module1.pyc'>

>>> printer
NameError: name 'printer' is not defined

>>> module1.printer
<function printer at 0x1018a1488>
```

module1.printer

module1 은 모듈 객체로서 이름이 부여되고
printer 는 이 객체의 속성 (attribute)

모듈의 이용 - from

- from 문장을 이용

```
>>> from module1 import printer
>>> printer('Hello world!')
Hello world!
```

module1.py

```
def printer(x):
    print(x)
```

```
>>> module1
NameError: name 'module1' is not defined

>>> module1.printer
NameError: name 'module1' is not defined

>>> printer
<function printer at 0x1007a1488>
```

from … import 문장에 의하여
module1.printer 함수가 객체로 생성되고
여기에 printer 라는 이름이 부여됨

모듈의 이용 - from *

- from 문장을 이용

```
>>> from module1 import printer
>>> printer('Hello world!')
Hello world!
```

module1.py

```
def printer(x):
    print(x)
```

```
>>> from module1 import *
>>> module1
NameError: name 'module1' is not defined

>>> module1.printer
NameError: name 'module1' is not defined

>>> printer
<function printer at 0x1020a1488>
```

from … import 문장에 의하여
module1.printer 함수가 객체로 생성되고
여기에 printer 라는 이름이 부여됨

(다른 객체들도 있었다면 각각 부여됨)

import 와 from … import

Python 에서 벌어지는 import 동작은 동일하다!
단, 어디에 어떤 이름을 부여하여 객체를 참조하는지만 달라질 뿐

- import 는 전체 모듈에 대하여 하나의 이름 (참조) 을 부여한다.
- from … import 는 (하나 이상의) 지정된 이름을 모듈 내의 같은 이름의 객체에 부여한다.
- from … import 는 모듈 내의 모든 객체에 대하여 동일한 이름을 부여한다.

```
from module import name1, name2, name3
```

=

```
import module
name1 = module.name1
name2 = module.name2
name3 = module.name3
del module
```

import 의 동작

import 는 한번만 실행된다.

```
>>> import simple
hello
>>> simple.spam
1

>>> simple.spam = 2
>>> simple.spam
2

>>> import simple
>>> simple.spam
2
```

simple.py

```
print('hello')
spam = 1
```

import 는 대입문과 동일하다.

```
>>> from small import x, y
>>> x = 42
>>> y[0] = 42
>>> x, y
(42, [42, 2])

>>> import small
>>> small.x
1
>>> small.y
[42, 2]
```

small.py

```
x = 1
y = [1, 2]
```

x 는 small.x 와 공유되는 참조가 아니며, (immutable)
y 는 small.y 와 공유되는 참조임 (mutable)

import 를 쓸 것인가, from … import 를 쓸 것인가

import module
vs
from module import name

module.name
vs
name

- 매번 module.name 처럼 코드를 작성하는 것은
  - 타이핑하기 귀찮은 일, 그냥 name 이라고 쓰고 싶다.
  - 프로그램이 커지고 비슷비슷한 이름들이 많이 등장하면, module.name 이라고 써야 알아보기 쉽다.
  - import 한 쪽에도 name 이라는 이름이 있다면, module.name 은 충돌을 피한다.
- from module import * 하면
  - 어떠한 이름들이 네임스페이스에 복사되는지 알기 어렵다. (특히 다른 사람의 코드에 대해서는)
  - 서로 다른 여러 개의 모듈을 import 하면 네임스페이스가 망가질 가능성은 더욱 커진다.

import 를 써야만 하는 경우

동일한 이름이 서로 다른 모듈에 있고,
이 모듈들을 import 해야 하는 경우

M.py

```
def func():
    print 'M'
```

N.py

```
def func():
    print 'N'
```

```
>>> from M import func
>>> from N import func
>>> func()                    # N.func 를 참조
N

>>> import M, N
>>> M.func()                  # 각각 M 과
M
>>> N.func()                  # N의 func 를 참조
N
```

dir 와 __dict__

- 내부적으로는 모듈의 네임스페이스가 사전으로 관리됨
  - module.__dict__ 객체
- dir (built-in 함수) 을 이용하여 이 사전의 키들을 열람할 수 있음

small.py

```
x = 1
y = [1, 2]
```

```
>>> dir(small)
['__builtins__', '__doc__', '__file__', '__name__', '__package__', 'x', 'y']

>>> list(small.__dict__.keys())
['__builtins__', '__file__', '__package__', 'x', 'y', '__name__', '__doc__']

>>> small.__dict__
???
```

Python 패키지 (Packages)

Python 에서의 패키지 (package)
= Python 코드가 들어있는 디렉토리 (directory)
= Python 모듈의 집합

import dir1.dir2.mod
from dir1.dir2.mod import x

하위 디렉토리의 구분자는 period (.) 를 이용
mod.py 는 dir1/dir2/mod.py 로 존재

패키지 경로 탐색 규칙

- 경로에 포함된 모든 디렉토리에는 __init__.py 파일이 포함되어 있어야 함
  - 파일 내용은 비어 있어도 됨
- 최상위 경로는 모듈 탐색 경로에 포함되어 있어야 함

```
dir0/
    dir1/
        __init__.py
        dir2/
            __init__.py
            mod.py
```

import dir1.dir2.mod

- dir1 과 dir2 에는 __init__.py 가 존재해야 함
- dir0 (최상위 디렉토리) 는 프로그램의 현재 디렉토리이거나 PYTHONPATH 등에 의하여 모듈 탐색 경로로 지정되어 있어야 함
  - __init__.py 는 있어도 되나, 무시됨

패키지 import 예제

dir1/__init__.py

```
print('dir1 init')
x = 1
```

dir1/dir2/__init__.py

```
print('dir2 init')
y = 2
```

dir1/dir2/mod.py

```
print('in mod.py')
z = 3
```

```
>>> import dir1.dir2.mod
dir1 init
dir2 init
in mod.py

>>> import dir1.dir2.mod

>>> from imp import reload
>>> reload(dir1)
dir1 init
<module 'dir1' from 'dir1/__init__.pyc'>

>>> reload(dir1.dir2)
dir2 init
<module 'dir1.dir2' from 'dir1/dir2/__init__.pyc'>
```

```
>>> dir1
<module 'dir1' from 'dir1/__init__.pyc'>
>>> dir1.dir2
<module 'dir1.dir2' from 'dir1/dir2/__init__.pyc'>
>>> dir1.dir2.mod
<module 'dir1.dir2.mod' from 'dir1/dir2/mod.py'>
```

```
>>> dir1.x
1
>>> dir1.dir2.y
2
>>> dir1.dir2.mod.z
3
```

패키지 import 예제

dir1/__init__.py

```
print('dir1 init')
x = 1
```

dir1/dir2/__init__.py

```
print('dir2 init')
y = 2
```

dir1/dir2/mod.py

```
print('in mod.py')
z = 3
```

```
>>> import dir1.dir2.mod
dir1 init
dir2 init
in mod.py

>>> dir2.mod
NameError: name 'dir2' is not defined

>>> mod.z
NameError: name 'mod' is not defined
```

```
>>> from dir1.dir2 import mod
dir1 init
dir2 init
in mod.py

>>> mod.z
3

>>> dir2
NameError: name 'dir2' is not defined
>>> dir1
NameError: name 'dir1' is not defined
```

모듈에 부여되는 이름을 변경: as

```
import modulename as name
```

=

```
import modulename
name = modulename
del modulename
```

```
from modulename import attrname as name
```

from 에도 적용할 수 있음

```
import reallylongmodulename as name
name.func()
```

짧은 별명을 이용함으로써 편리

```
from module1 import utility as util1
from module2 import utility as util2
```

utility 라는 같은 이름으로
서로 다른 두 모듈을 import 할 수 없음

```
import dir1.dir2.mod as mod
mod.func()
```

=? from dir1.dir2 import mod

__name__ 과 __main__

__name__ 은 모듈의 속성 (attribute) 으로서, 해당 모듈의 이름을 가리킴

- Python 파일 (.py) 이 최상위 수준 (top-level) 프로그램으로 실행될 때에는
  - 프로그램 시작 시점에 __name__ 은 "__main__" 으로 설정됨
- 그렇지 않은 경우에는 (import 되는 경우에는)
  - import 되는 시점에 __name__ 은 자신이 클라이언트에게 알려지는 이름으로 설정됨

__name__ 을 "__main__" 과 비교함으로써,
자신이 최상위 프로그램으로 실행되고 있는지 여부를 알아낼 수 있음

__name__ 과 __main__ 예제

runme.py

```
def tester():
        print("It's Christmas in Heaven...")

if __name__ == '__main__':
    tester()
```

이 부분은 runme.py 가 최상위 스크립트로 실행될 때에만 실행되고 import 에 의하여 실행될 때에는 실행되지 않음

```
% python
>>> import runme
>>> runme.tester()
It's Christmas in Heaven...

% python runme.py
It's Christmas in Heaven...
```

단위 테스트 (Unit Test) 에 응용

min.py

```
print('I am: %s' % __name__)

def min(*args):
        res = args[0]
        for arg in args[1:]:
            if res > arg:
                    res = arg
        return res

if __name__ == '__main__':
        print(min(4, 2, 1, 5, 6, 3))
        print(min('s', 'p', 'a', 'm'))
```

```
% python min.py
I am: __main__
1
a

% python
>>> import min
I am: min
>>> min.min(4, 2, 3)
2
>>> min.min('s', 'p', 'a', 'm')
'a'
```

__main__ 의 명령어 라인 인자 (Command-line Arguments)

test.py

```
"""
This is a test script in Python
"""

def test1(x):
    """
    This function is test1 - add 1.
    """
    print("test1: %s" % (int(x) + 1))

def test2(x):
    """
    This function is test2 - multiply by 2.
    """
    print("test2: %s" % (int(x) * 2))

if __name__ == '__main__':
    import sys
    if len(sys.argv) == 3:
        if sys.argv[1] == 't1':
            test1(sys.argv[2])
        if sys.argv[1] == 't2':
            test2(sys.argv[2])
    else:
        print("Hello")
```

```
%python test.py
Hello

% python test.py t1 3
test1: 4

% python test.py t2 8
test2: 16

% python
>>> import test
>>> test.test1(5)
test1: 6

>>> test.test1(7)
test1: 8
```

인자의 개수: len(sys.argv)
argv[0] 는 스크립트 파일 이름
argv 는 모두 문자열로 전달됨

```
>>> help(test)
Help on module test:

NAME
    test - This is a test script in Python

FILE
    /…/…/test.py

FUNCTIONS
    test1(x)
        This function is test1 - add 1.

    test2(x)
        This function is test2 - multiply by 2.
```

Quiz

- 모듈은 어떻게 만들어지는가?
- 특정 모듈을 이용하기 위하여 import 하는 세 가지 방식은?
  - 그리고 그 각각의 장단점은?
- __init__.py 파일의 목적은 무엇인가?
  - 이 파일은 어느 디렉토리에 존재해야 하는가?
- __name__ 이란 무엇인가? 그리고, 어디에 이용할 수 있는가?
  - __name__ 이 "__main__" 인 것은 무엇을 의미하는가?
    - 이것을 어디에 응용할 수 있는가?

Q & A