

# Python Module 만들고 불러보기 [1/2]

모듈에 대해서 자세히 살펴보기 전에 간단한 모듈을 한번 만들어 보자.

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

위와 같이 sum 함수만 있는 파일 mod1.py를 만들고 C:\Python 디렉터리에 저장하자. 이 파일이 바로 모듈이다. 지금까지 에디터로 만들어 왔던 파일과 다르지 않다.

반드시 mod1.py를 저장한 위치로 이동한 다음 이후 예제를 진행해야 한다. 그래야만 대화형 인터프리터에서 mod1.py를 읽을 수 있다. 이제 아래와 같이 따라 해보자.

```
>>> import mod1
>>> print(mod1.sum(3,4))
7
```

# Python Module 만들고 불러보기 [2/2]

이번에는 mod1.py 파일에 다음 함수를 추가해 보자.

```
def safe_sum(a, b):  
    if type(a) != type(b):  
        print("더할수 있는 것이 아닙니다.")  
        return  
    else:  
        result = sum(a, b)  
    return result
```

```
from mod1 import sum, safe_sum
```

```
from mod1 import *
```

## `if __name__ == "__main__":` 의 의미

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

def safe_sum(a, b):
    if type(a) != type(b):
        print("더할수 있는 것이 아닙니다.")
        return
    else:
        result = sum(a, b)
        return result

if __name__ == "__main__":
    print(safe_sum('a', 1))
    print(safe_sum(1, 4))
    print(sum(10, 10.4))
```

```
C:\Python>python mod1.py
더할 수 있는 것이 아닙니다.
None
5
20.4
```

`if __name__ == "__main__":` 을 사용하면 `C:\Python>python mod1.py` 처럼 직접 이 파일을 실행시켰을 때는 `__name__ == "__main__"` 이 참이 되어 if문 다음 문장들이 수행된다. 반대로 대화형 인터프리터나 다른 파일에서 이 모듈을 불러서 사용할 때는 `__name__ == "__main__"` 이 거짓이 되어 if문 다음 문장들이 수행되지 않는다.

# 클래스나 변수 등을 포함한 모듈

지금까지 살펴본 모듈은 함수만 포함했지만 클래스나 변수 등을 포함할 수도 있다. 다음의 프로그램을 작성해 보자.

```
# mod2.py
PI = 3.141592

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

def sum(a, b):
    return a+b

if __name__ == "__main__":
    print(PI)
    a = Math()
    print(a.solv(2))
    print(sum(PI , 4.4))
```

이 파일은 반지름을 계산하는 Math 클래스와 두 값을 더하는 sum 함수 그리고 원주율 값에 해당되는 PI 변수처럼 클래스, 함수, 변수 등을 모두 포함하고 있다. 파일 이름을 mod2.py로 하고

C:\Python 디렉터리에 저장하자. mod2.py 파일은 다음과 같이 실행할 수 있다.

```
C:\Python>python mod2.py
3.141592
12.566368
7.541592
```

```
# modtest.py
import mod2
result = mod2.sum(3, 4)
print(result)
```

mod2.py와 modtest.py가 같은 directory에 있어야 한다

```
C:\Python>python
>>> import mod2
>>>

__name__ == "__main__" 이 거짓이 되므로 아무런 값도 출력되지 않는다.
```

## [모듈을 불러오는 또 다른 방법]

우리는 지금껏 도스 창을 열고 모듈이 있는 디렉터리로 이동한 다음에나 모듈을 사용할 수 있었다.  
이번에는 모듈을 저장한 디렉터리로 이동하지 않고 모듈을 불러와서 사용하는 방법

### 1. sys.path.append(모듈을 저장한 디렉터리) 사용하기

먼저 sys 모듈을 불러온다.

```
>>> import sys
```

```
>>> sys.path
['', 'C:\\Windows\\SYSTEM32\\python35.zip', 'c:\\Python35\\DLLs',
'c:\\Python35\\lib', 'c:\\Python35', 'c:\\Python35\\lib\\site-packages']
```

```
>>> sys.path.append("C:/Python/Mymodules")
>>> sys.path
['', 'C:\\Windows\\SYSTEM32\\python35.zip', 'c:\\Python35\\DLLs',
'c:\\Python35\\lib', 'c:\\Python35', 'c:\\Python35\\lib\\site-packages',
'C:/Python/Mymodules']
```

### 2. PYTHONPATH 환경 변수 사용하기

모듈을 불러와서 사용하는 또 다른 방법으로는 PYTHONPATH 환경 변수를 사용하는 방법이 있다.

```
C:\Users\home>set PYTHONPATH=C:\Python\Mymodules
C:\Users\home>python
Python 3.5.1 (v3.5.1:37a07cee5969, Dec 6 2015, 01:54:25) [MSC v.1900 64 bit (AM...
Type "help", "copyright", "credits" or "license" for more information.
>>> import mod2
>>> print(mod2.sum(3,4))
7
```

# Python Package

패키지(Packages)는 도트(.)를 이용하여 파이썬 모듈을 계층적(디렉터리 구조)으로 관리할 수 있게 해준다. 예를 들어 모듈명이 A.B인 경우 A는 패키지명이 되고 B는 A 패키지의 B 모듈이 된다.

파이썬 패키지는 디렉터리와 파이썬 모듈로 이루어지며 구조는 다음과 같다.

*가상의 game 패키지 예*

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

game, sound, graphic, play는 디렉터리명이고 .py 확장자를 가지는 파일은 파이썬 모듈이다. game 디렉터리가 이 패키지의 루트 디렉터리이고 sound, graphic, play는 서브 디렉터리이다.

# Package 기본요소 준비하기 [1/2]

1. `C:/Python` 이라는 디렉터리 밑에 `game` 및 기타 서브 디렉터리들을 생성하고 `.py` 파일들을 다음과 같이 만들어 보자(만약 `C:/Python` 이라는 디렉터리가 없다면 먼저 생성하고 진행하자).

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

2. 각 디렉터리에 `__init__.py` 파일을 만들어 놓기만 하고 내용은 일단 비워 둔다.
3. `echo.py` 파일은 다음과 같이 만든다.

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

# Package 기본요소 준비하기 [2/2]

4. render.py 파일은 다음과 같이 만든다.

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

5. 다음 예제들을 수행하기 전에 우리가 만든 game 패키지를 참조할 수 있도록 다음과 같이 도스 창에서 set 명령을 이용하여 PYTHONPATH 환경 변수에 C:/Python 디렉토리를 추가한다. 그리고 파이썬 인터프리터(Interactive shell)를 실행하자.

```
C:\> set PYTHONPATH=C:/Python
C:\> python
Python 3.5.1 (v3.5.1:37a07cee5969, Dec 6 2015, 01:54:25) [MSC v.1900 64 bit (AM...
Type "help", "copyright", "credits" or "license" for more information.
>>>
```



# 패키지 안의 함수 실행하기

첫 번째는 echo 모듈을 import하여 실행하는 방법으로, 다음과 같이 실행한다.

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

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

두 번째는 echo 모듈이 있는 디렉터리까지를 from ... import하여 실행하는 방법이다.

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

세 번째는 echo 모듈의 echo\_test 함수를 직접 import하여 실행하는 방법이다.

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

## `__init__.py` 의 용도

`__init__.py` 파일은 해당 디렉터리가 패키지의 일부임을 알려주는 역할을 한다. 만약 game, sound, graphic 등 패키지에 포함된 디렉터리에 `__init__.py` 파일이 없다면 패키지로 인식되지 않는다.

(※ python3.3 버전부터는 `__init__.py` 파일 없이도 패키지로 인식이 된다(PEP 420 (<https://www.python.org/dev/peps/pep-0420/>)). 하지만 하위 버전 호환을 위해 `__init__.py` 파일을 생성하는 것이 안전한 방법이다.)

시험 삼아 sound 디렉터리의 `init.py`를 제거하고 다음을 수행해 보자.

```
>>> import game.sound.echo
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: No module named sound.echo
```

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

sound 디렉터리에 `__init__.py` 파일이 없어서 임포트 오류(ImportError)가 발생하게 된다.

# \_\_all\_\_ variable의 용도

외부의 python 프로그램에서 import문장을 불렀을때 →

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

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

이렇게 특정 디렉터리의 모듈을 \* 를 이용하여 import할 때에는 다음과 같이 해당 디렉터리의 \_\_init\_\_.py 파일에 \_\_all\_\_ 이라는 변수를 설정하고 import할 수 있는 모듈을 정의해 주어야

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

여기서 \_\_all\_\_ 이 의미하는 것은 sound 디렉터리에서 \* 기호를 이용하여 import할 경우 이곳에 정의된 echo 모듈만 import된다는 의미이다.

위와 같이 \_\_init\_\_.py 파일을 변경한 후 위 예제를 수행하면 원하던 결과가 출력되는 것을 확인할 수 있다.

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

## relative 패키지

만약 graphic 디렉터리의 render.py 모듈이 sound 디렉터리의 echo.py 모듈을 사용하고 싶다면

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

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

from game.sound.echo import echo\_test라는 문장을 추가하여 echo\_test() 함수를 사용할 수 있도록 수정했다.

이렇게 수정한 후 다음과 같이 수행해 보자.

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

# Installing Modules: Using Package Manager



- **PyPI(the Python Package Index)**는 Python SW들이 모여있는 저장소
  - 파이썬 개발자들은 자신들의 개발한 파이썬 모듈들을 PyPI에 upload
  - PyPI에 저장된 모듈들은 누구에게나 공개
  - PyPI 홈페이지에 접속하지 않고 pip을 통해서 손쉽게 원하는 모듈을 다운로드
- 먼저 “pip” sw를 pc에 install 해야 한다
- “pip” fetches package meta-data and source codes from an official third-party repository called “**PyPI**”
- Windows cmd창 or Linux shell 에서:
  - `pip install <package name>`
  - Now you can use the `<package name>` library using `import`

# PyPI: the Python Package Index

- A repository of open software for Python
- You can download extrinsic libraries and packages from PyPI by using [pip](#)

## PyPI - the Python Package Index

The Python Package Index is a repository of software for the Python programming language. There are currently **74662** packages here.  
To contact the PyPI admins, please use the [Support](#) or [Bug reports](#) links.

**Not Logged In**  
[Login](#)  
[Register](#)  
[Lost Login?](#)  
Use [OpenID](#)   
[Login with Google](#) 

**Get Packages**

To use a package from this index either "[pip](#) install *package*" ([get pip](#)) or download, unpack and "python setup.py install" it.

**Package Authors**

Submit packages with "[python setup.py upload](#)". The index [hosts package docs](#). You may also use the [web form](#). You must [register](#). Testing? Use [testpypi](#).

**Infrastructure**

To interoperate with the index use the [JSON](#), [OAuth](#), [XML-RPC](#) or [HTTP](#) interfaces. Use [local mirroring](#) or [caching](#) to make installation more robust.

Updated	Package	Description
2016-02-16	<a href="#">django-drynk 0.1.1</a>	django-drynk gives you DRY natural keys.
2016-02-16	<a href="#">gbdx-tools 0.1.2</a>	Python tools to order imagery and launch workflows on DigitalGlobe GBDX platform.
2016-02-16	<a href="#">pyextend 0.1.9</a>	the python extend lib
2016-02-16	<a href="#">marketorestpython 0.1.10</a>	Python Client for the Marketo REST API

# Pip – Package Manager

- Pip Installs Package / Pip Installs Python
- a package management system used to install and manage software packages written in Python
  - Python 2.7.9 and later, and Python 3.4 and later include pip by default

```
pip install <some-package-name>
```

```
C:\Users\Taeuk>pip install Django
Collecting Django
  Downloading Django-1.9.2-py2.py3-none-any.whl (6.6MB)
    100% |#####| 6.6MB 97kB/s
Installing collected packages: Django
Successfully installed Django-1.9.2
```

# Popular Python Libraries (Packages)

- Database Libraries
  - pandas
  - SQLAlchemy
- Scientific Libraries
  - SciPy
  - NumPy
  - matplotlib
- Web Development Libraries
  - Django
  - Flask
- Natural Language Libraries
  - NLTK



# NumPy

```
>>> import numpy as np
>>> a = np.arange(15).reshape(3, 5)
>>> a
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
>>> a.shape
(3, 5)
>>> a.ndim
2
>>> a.dtype.name
'int64'
>>> a.size
15
>>> type(a)
<type 'numpy.ndarray'>
```

```
>>> b = np.array([6, 7, 8])
>>> b
array([6, 7, 8])
>>> type(b)
<type 'numpy.ndarray'>
```

# matplotlib

```
import numpy as np
import matplotlib.pyplot as plt
```

```
# Build a vector of 10000 normal deviates with variance  $0.5^2$  and mean 2
```

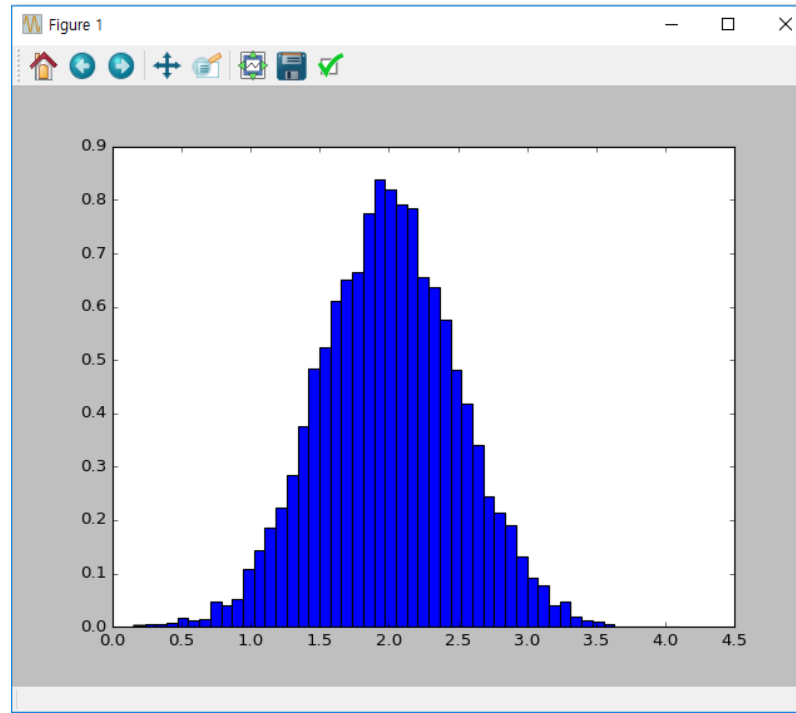
```
mu, sigma = 2, 0.5
```

```
v = np.random.normal(mu, sigma, 10000)
```

```
# Plot a normalized histogram with 50 bins
```

```
plt.hist(v, bins=50, normed=1)
```

```
plt.show()
```



# NLTK(Natural Language Toolkit)

- <http://www.nltk.org>

## NLTK 3.0 documentation

[NEXT](#) | [MODULES](#) | [INDEX](#)

### Natural Language Toolkit

NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to [over 50 corpora and lexical resources](#) such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active [discussion forum](#).

Thanks to a hands-on guide introducing programming fundamentals alongside topics in computational linguistics, plus comprehensive API documentation, NLTK is suitable for linguists, engineers, students, educators, researchers, and industry users alike. NLTK is available for Windows, Mac OS X, and Linux. Best of all, NLTK is a free, open source, community-driven project.

NLTK has been called “a wonderful tool for teaching, and working in, computational linguistics using Python,” and “an amazing library to play with natural language.”

[Natural Language Processing with Python](#) provides a practical introduction to programming for language processing. Written by the creators of NLTK, it guides the reader through the fundamentals of writing Python programs, working with corpora, categorizing text, analyzing linguistic structure, and more. The book is being updated for Python 3 and NLTK 3. (The original Python 2 version is still available at [http://nltk.org/book\\_1ed](http://nltk.org/book_1ed).)

#### TABLE OF CONTENTS

[NLTK News](#)

[Installing NLTK](#)

[Installing NLTK Data](#)

[Contribute to NLTK](#)

[FAQ](#)

[Wiki](#)

[API](#)

[HOWTO](#)

#### SEARCH

Enter search terms or a module, class or function name.

# NLTK(Natural Language Toolkit)

```
import nltk

sentence = "At eight o'clock on Thursday morning Arthur didn't  
feel very good."
tokens = nltk.word_tokenize(sentence)
tagged = nltk.pos_tag(tokens)
print(tokens)
for tag in tagged:
    print(tag)

['At', 'eight', "o'clock", 'on', 'Thursday', 'morning', 'Arthur', 'did', "n't", 'feel', 'very', 'good', '.']
('At', 'IN')
('eight', 'CD')
("o'clock", 'NN')
('on', 'IN')
('Thursday', 'NNP')
('morning', 'NN')
('Arthur', 'NNP')
('did', 'VBD')
("n't", 'RB')
('feel', 'VB')
('very', 'RB')
('good', 'JJ')
('.', '.')
```

# Installing Modules: *The Manual Way*

- If you happen to have downloaded Python modules from the internet, or you simply want to use one that you made yourself,
  - Copy the module file into **C/Python34/Lib/site-packages/**
- Note that the file name is case-sensitive, (although it is not in Windows file explorer)
- For example, `zelle_graphics.py` may be installed in Python
  - [http://mcsp.wartburg.edu/zelle/python/zelle\\_graphics.py](http://mcsp.wartburg.edu/zelle/python/zelle_graphics.py)
- Now you can use the graphics library using `import`
  - `import zelle_graphics`