

# 데이터과학을 위한 파이썬 프로그래밍



## 11. 모듈과 패키지

# 목차

1. 모듈과 패키지의 이해
2. ~~모듈 만들기(?)~~ => 모듈의 활용법(사례)
3. 패키지 만들기
4. 가상환경 사용하기

# 01 모듈과 패키지의 이해

## 01. 모듈과 패키지의 이해



(a) 파이썬



(b) 자바

[파이썬과 자바의 작성 비유]

# 01. 모듈과 패키지의 이해

## ■ 모듈의 개념

- 프로그래밍에서의 모듈은 작은 프로그램 조각을 뜻한다. 즉, 하나하나 연결해 어떤 목적을 가진 프로그램을 만드는 작은 프로그램이다. 각 모듈 역시 저마다 역할이 있고, 서로 다른 모듈과 인터페이스만 연결되면 사용할 수 있다.

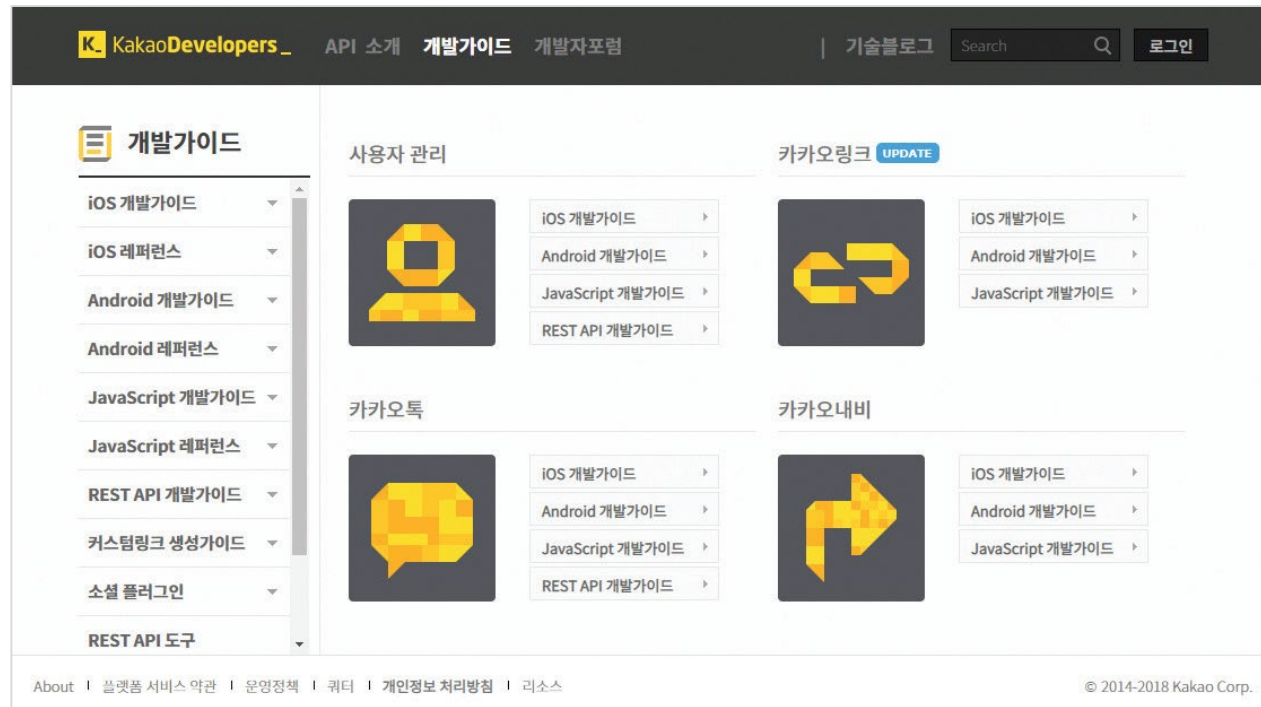


[구글의 프로젝트 Ara 콘셉트]

# 01. 모듈과 패키지의 이해

## ■ 모듈의 개념

- 모듈화된 프로그램을 사용하면, 다른 개발자가 만든 프로그램이나 자신이 만든 프로그램을 매우 쉽게 사용하거나 제공할 수 있다.



# 01. 모듈과 패키지의 이해

## ■ 모듈의 개념

- 내장 모듈이라고 하여 파이썬에서 기본적으로 제공하는 모듈 중 대표적으로 random 모듈이 있다. 이는 난수를 쉽게 생성해 주는 모듈이다. random 모듈을 호출하기 위한 코드는 다음과 같다.

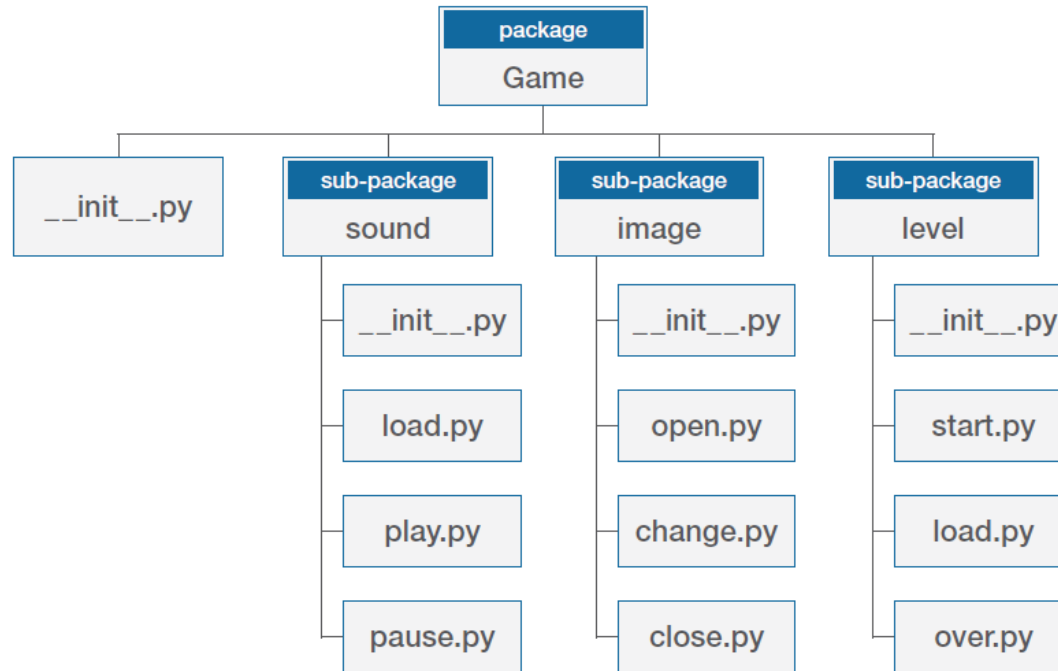
```
>>> import random
>>> random.randint(1, 1000)
198
```

- ➡ import 구문이 중요하다. import 구문은 뒤에 있는 모듈, 즉 random을 사용할 수 있도록 호출하라는 명령어이다. 다음으로 해당 모듈의 이름을 사용하여 그 모듈 안에 있는 함수, 여기서는 randint( ) 함수를 사용할 수 있다. randint( ) 함수를 사용하기 위해서는 이 randint 함수의 인터페이스, 즉 매개변수의 설정이 어떻게 되어 있는지 알아야 한다.

# 01. 모듈과 패키지의 이해

## ■ 패키지의 개념

- 패키지(packages)는 모듈의 묶음이다. 일종의 디렉터리처럼 하나의 패키지 안에 여러 개의 모듈이 있는데, 이 모듈들이 서로 포함 관계를 가지며 거대한 패키지를 만든다.



[모듈과 패키지의 관계]



## 02 모듈의 활용법(사례)

### ~~모듈 만들기~~

## 02. 모듈 만들기

### ■ 모듈 만들기 실습

코드 11-1 fah\_converter.py

```
1 def covert_c_to_f(celcius_value):  
2     return celcius_value * 9.0 / 5 + 32
```

코드 11-2 module\_ex.py

```
1 import fah_converter  
2  
3 print ("Enter a celsius value:")  
4 celsius = float(input())  
5 fahrenheit = fah_converter.covert_c_to_f(celsius)  
6 print ("That's", fahrenheit, "degrees Fahrenheit.")
```

Enter a celsius value:

10

← 사용자 입력

That's 50.0 degrees Fahrenheit.

← 결과값 출력

## 02. 모듈 만들기

### ■ 모듈 만들기 실습

- ➡ [코드 11-2]에서 가장 중요한 핵심 코드는 1행의 `import fah_converter`로, 기존에 만든 코드 파일에서 `.py`를 빼고 해당 파일의 이름만으로 파일의 함수를 불러 사용할 수 있다. 즉, `.py` 자체가 하나의 모듈이 되어 해당 모듈의 코드를 가져다 사용할 수 있다. 이 코드에서는 `fah_converter`가 모듈이고, 해당 모듈 안의 함수 `covert_c_to_f( )`를 가져다 사용하기 위해 5행에서 `fahrenheit = fah_converter.covert_c_to_f(celsius)` 코드를 작성하였다.
- ➡ 여기서 핵심은 호출받는 모듈과 호출하여 사용하는 클라이언트 프로그램이 같은 디렉터리 안에 있어야 한다는 것이다. 여기서는 `fah_converter.py`와 `module_ex.py`가 같은 디렉터리안에 있어야 문제없이 실행된다.

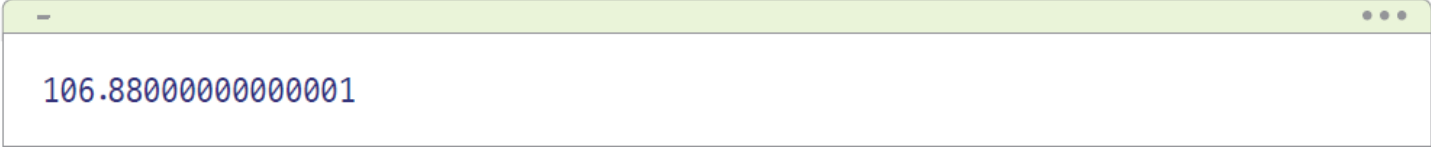
## 02. 모듈 만들기

### ■ 1) 모듈명.함수이름으로 호출

- 네임스페이스는 모듈 호출의 범위를 지정한다.
- 네임스페이스를 만드는 방법에 대해 알아보자. 첫 번째는 모듈 이름에 별칭(alias)을 생성하여 모듈 안으로 코드를 호출하는 방법이다. 별칭은 모듈의 이름을 바꿔 부를 때 사용한다.

코드 11-3 namespace1.py

```
1 import fah_converter as fah
2 print(fah.covert_c_to_f(41.6))
```



106.88000000000001

- ➡ fah\_converter 모듈을 fah로 이름을 변경하여 호출하였다. 그리고 fah.covert\_c\_to\_f(41.6) 코드로 fah\_converter 모듈 안에 covert\_c\_to\_f() 함수를 호출하였다. '모듈명.함수명(또는 클래스명/변수명)'은 해당 모듈 안에 있는 함수, 클래스, 변수를 호출할 수 있다.


## 02. 모듈 만들기

### ■ 2) from을 이용하여 특정함수/클래스만을 호출

- 두 번째 방법은 from 구문을 사용하여 모듈에서 특정 함수 또는 클래스만 호출하는 방법이다.

코드 11-4 namespace2.py

```
1 from fah_converter import covert_c_to_f
2 print(covert_c_to_f(41.6))
```



106.88000000000001

- ➡ 1행에서처럼 'from 모듈명 import 모듈 안에 있는 함수명'을 작성하여 해당 모듈 안에 있는 함수를 가져다 사용할 수 있다. 주의할 점은 from은 패키지를 호출하고, 해당 패키지 안에 있는 모듈을 호출할 때도 from 키워드를 사용할 수 있으니 참고하기 바란다. 패키지와 패키지, 패키지와 모듈 간에는 서로 중첩 구조를 가질 수 있고, 이 중첩 구조를 호출하는 것이 바로 from의 역할이다.

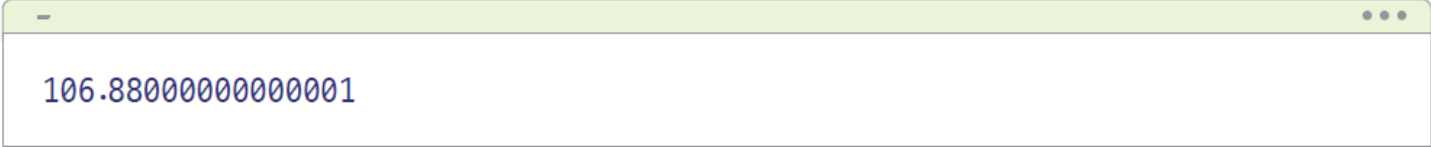
## 02. 모듈 만들기

### ■ 3) as를 이용하여 네임스페이스를 새로 만들어 호출

- 네임스페이스는 모듈 호출의 범위를 지정한다.
- 네임스페이스를 만드는 방법에 대해 알아보자. 첫 번째는 모듈 이름에 별칭(alias)을 생성하여 모듈 안으로 코드를 호출하는 방법이다. 별칭은 모듈의 이름을 바꿔 부를 때 사용한다.

코드 11-3 namespace1.py

```
1 import fah_converter as fah
2 print(fah.covert_c_to_f(41.6))
```



106.88000000000001

- ➡ fah\_converter 모듈을 fah로 이름을 변경하여 호출하였다. 그리고 fah.covert\_c\_to\_f(41.6) 코드로 fah\_converter 모듈 안에 covert\_c\_to\_f() 함수를 호출하였다. '모듈명.함수명(또는 클래스명/변수명)'은 해당 모듈 안에 있는 함수, 클래스, 변수를 호출할 수 있다.


## 02. 모듈 만들기

### ■ 4) \*를 이용하여 모든 함수/클래스/변수를 활용

- 세 번째 방법은 해당 모듈 안에 있는 모든 함수, 클래스, 변수를 가져오는 별표(\*)를 사용하는 것이다. 일반적으로 컴퓨터에서 별표는 곱셈의 의미도 있지만, 모든 것이라는 뜻도 있다. [코드 11-5]의 1행과 같이 'from 모듈명 import \*'라고 입력하면, 해당 모듈 안에 있는 모든 사용가능한 리소스를 호출한다.

**코드 11-5** namespace3.py

```
1 from fah_converter import *  
2 print(covert_c_to_f(41.6))
```



106.88000000000001

# 모듈 호출 기법 종합 예제

z\_fah\_converter.py, z\_module\_ex.py

## 모듈 z\_fah\_converter.py

```
1 def covert_c_to_f(celcius_value):  
2     return celcius_value * 9.0 / 5 + 32  
3 def more():  
4     return 'This is bonus.'
```

## 수행결과

```
1) That's 69.8 degrees Fahrenheit.  
This is bonus.  
2) That's 69.8 degrees Fahrenheit.  
This is bonus.  
3) That's 69.8 degrees Fahrenheit.  
4) That's 69.8 degrees Fahrenheit.  
This is bonus.  
5) That's 69.8 degrees Fahrenheit.
```

```
# 5. from과 as를 이용해 함수의 이름을 alias를 바꾸어 호출.  
from z_fah_converter import covert_c_to_f as cvt  
fahrenheit = cvt(celsius)  
print("5) That's", fahrenheit, "degrees Fahrenheit.")  
del cvt
```

```
z_fah_converter.py x z_module_ex.py x  
1 # 1. 모듈의 이름과 그가 지원하는 함수를 지명하여 호출한다.  
2 import z_fah_converter  
3 celsius = 21  
4 fahrenheit = z_fah_converter.covert_c_to_f(celsius)  
5 print("\n1) That's", fahrenheit, "degrees Fahrenheit.")  
6 print(z_fah_converter.more())  
7 del z_fah_converter # 모듈 제거  
8  
9 # 2. from을 이용하여 특정함수/클래스만을 호출  
10 from z_fah_converter import covert_c_to_f, more  
11 fahrenheit = covert_c_to_f(celsius)  
12 print("2) That's", fahrenheit, "degrees Fahrenheit.")  
13 #print(z_fah_converter.more()) # 오류 발생  
14 print(more())  
15 del covert_c_to_f, more  
16  
17 # 3. as를 이용해 모듈의 이름을 alias를 바꾸어 간편하게 호출한다.  
18 import z_fah_converter as fc  
19 fahrenheit = fc.covert_c_to_f(celsius)  
20 print("3) That's", fahrenheit, "degrees Fahrenheit.")  
21 del fc  
22  
23 # 4. 모듈의 이름을 지정하지 않고 모듈 안의 모든 함수를 호출한다.  
24 from z_fah_converter import *  
25 fahrenheit = covert_c_to_f(celsius)  
26 print("4) That's", fahrenheit, "degrees Fahrenheit.")  
27 print(more())  
28 #del * # 오류.  
29 del covert_c_to_f, more
```

메인 프로그램 z\_module\_ex.py



## 02. 내장 모듈의 사용 : random 모듈

### ■ random 모듈

- 난수 생성 모듈은 이미 많이 본 random 모듈을 사용하면 된다. 해당 모듈 안에는 특정 범위의 정수 임의로 생성하는 randint( ) 함수와 임의의 난수를 생성하는 random( ) 함수를 쓸 수 있다.

```
>>> import random
>>> print (random.randint (0, 100))      # 0~100 사이의 정수 난수를 생성
7
>>> print (random.random())              # 일반적인 난수 생성
0.056550421789531846
```

## 02. 내장 모듈의 사용 : time 모듈

### ■ time 모듈

- 시간과 관련된 time 모듈은 일반적으로 시간을 변경하거나 현재 시각을 출력한다. 대표적으로 프로그램이 동작하는 현재 시각을 출력할 수 있다.

### ■ 타임 모듈에서 사용되는 용어 정리

- <https://docs.python.org/3.8/library/time.html>
- The **epoch** is the point where the time starts, and is platform dependent.
  - For Unix, the epoch is January 1, 1970, 00:00:00 (UTC).
- **UTC** is Coordinated Universal Time (formerly known as Greenwich Mean Time, or GMT).
- **DST** is Daylight Saving Time, an adjustment of the timezone by (usually) one hour during part of the year.

# time 모듈(1)

z\_time.py

```
# -----  
# 실습 1: 현재의 시간 알아내기.  
# 에포크(epoch): 시간의 기준점, 1970년 1월 1일 0시 0분 0초  
# gmtime(), localtime()은 struct_time 타입의 객체를 반환한다.  
# asctime()은 이를 스트링으로 변환한다.  
# -----  
import time  
a = time.gmtime(0)    # epoch, midnight on January 1, 1970 UTC  
print('\nEpoch time:', time.asctime(a)); print(a)  
print('type(time.gmtime())=>', type(a))  
  
b = time.gmtime()      # ()안에 시점 미지정. 현재의 UTC  
print('\nCurrent UTC time:', time.asctime(b)); print(b)  
  
c = time.localtime()  # 현재의 지역시간  
print('\nCurrent local time:', time.asctime(c)); print(c)
```

Epoch time: Thu Jan 1 00:00:00 1970

time.struct\_time(tm\_year=1970, tm\_mon=1, tm\_mday=1, tm\_hour=0, tm\_min=0, tm\_sec=0, tm\_wday=3, tm\_yday=1)  
type(time.gmtime())=> <class 'time.struct\_time'>

Current UTC time: Sun Nov 17 13:38:59 2019

time.struct\_time(tm\_year=2019, tm\_mon=11, tm\_mday=17, tm\_hour=13, tm\_min=38, tm\_sec=59, tm\_wday=6, tm\_yday=321)

Current local time: Sun Nov 17 22:38:59 2019

time.struct\_time(tm\_year=2019, tm\_mon=11, tm\_mday=17, tm\_hour=22, tm\_min=38, tm\_sec=59, tm\_wday=6, tm\_yday=321)

## time 모듈(2)

z\_time.py

```
# -----  
# 실습 2: 경과 시간 알아내기, 지정시간 대기하기  
# 1) time.time(): UTC 시간을 기준으로 현재까지의 경과 시간[초, 부동소수] 반환  
# 2) time.sleep(): 지정한 시간(초, 부동소수)만큼 대기  
# -----  
  
import time  
start = time.time()      # time(1st) 모듈의 time(2nd) 함수(class).  
time.sleep(1)            # time 모듈의 sleep() 함수. 1[초] 동안 대기  
end = time.time()        # 경과시간 계산  
print('elapsed time=', end-start)      # 지연시간[초] 출력  
  
# sleep() 함수에 대해서는 모듈이름(time)을 지정할 필요가 없다.  
from time import sleep  
start = time.time()  
sleep(0.5)  
  
# 모듈 time의 모든 함수에 대해 모듈 이름을 지정할 필요가 없다.  
from time import *  
end = time()  
print('elapsed time=', end-start)      # 지연시간[초] 출력
```

```
elapsed time= 1.000382900238037  
elapsed time= 0.5006935596466064
```

## 02. 내장 모듈의 사용 : urllib 모듈

### ■ urllib 모듈

- 웹과 관련된 urllib 모듈은 웹 주소의 정보를 불러온다. 대표적으로 urllib의 request 모듈을 사용하면 특정 URL의 정보를 불러올 수 있다. urllib.request.urlopen( )의 괄호에 특정 웹주소를 입력하면 해당 주소의 HTML 정보를 가져온다.

```
>>> import urllib.request
>>> response = urllib.request.urlopen("http://theteamlab.io")
>>> print(response.read())
```

# urllib 모듈

A URL usually consists of the following components:

링크

Protocol, domain, path (or pathname), hash and query string.

- **Protocol** is the technology that will be used to transfer the data, usually http or https
- **Domain** is the the domain name, *tealium.com* for example.
- **Path** relates to the section and page on the site
- **Hash** relates to a section within the page
- **Query string** contains data that is being passed to the page

So if we look at a URL, you can see how it gets broken up:

<http://tealium.com/solutions/?example=test&example2=test2#section3>

웹으로 정보 주고 받기 링크

프로토콜://계정:패스워드@호스트:포트번호/하위경로?질의조건#색인

<https://python.bakyeono.net/data/movies.json>

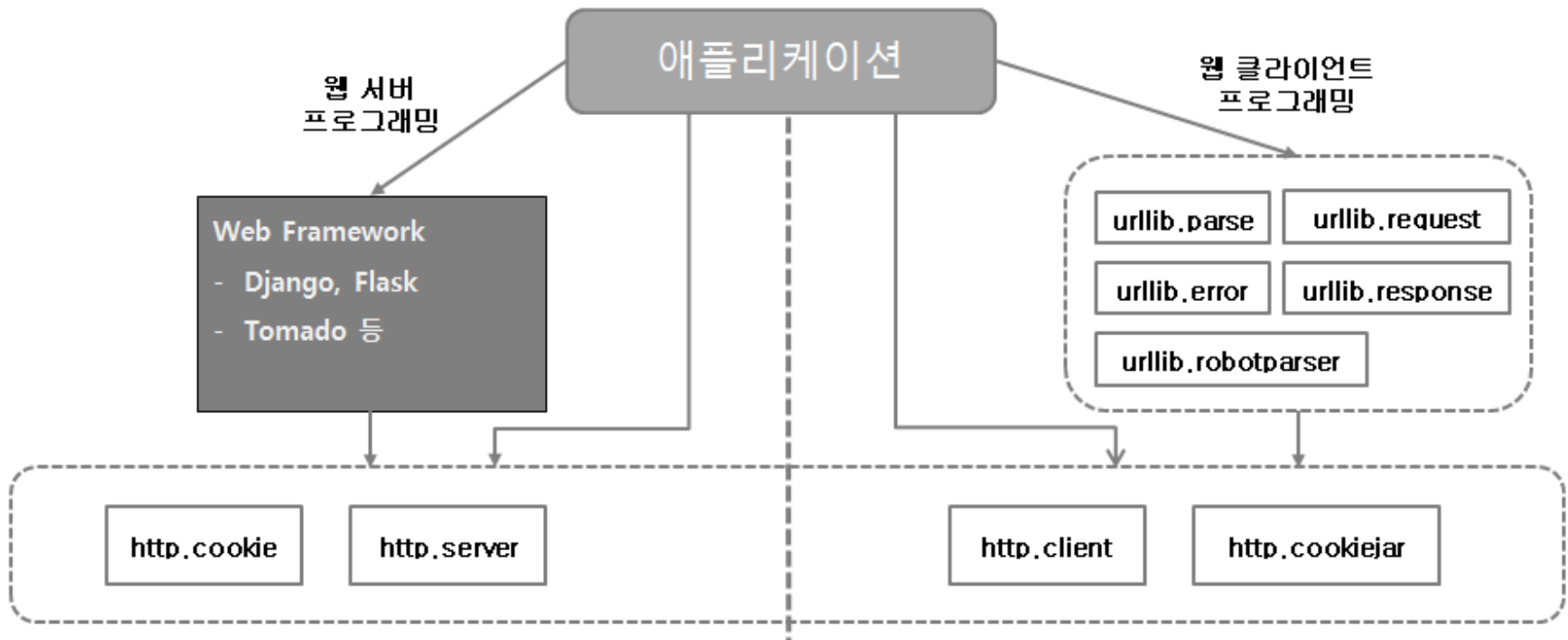
프로토콜

호스트

하위 경로

# Urllib package

- *urllib*는 URL 작업을 위한 여러 모듈을 모은 패키지입니다. [링크](#)
  - 1) URL을 열고 읽기 위한 *urllib.request*
  - 2) *urllib.request*에 의해 발생하는 예외를 포함하는 *urllib.error*
  - 3) URL 구문 분석을 위한 *urllib.parse*
  - 4) *robots.txt* 파일을 구문 분석하기 위한 *urllib.robotparser*



# Urllib package

- *urllib.request --- Extensible library for opening URLs.* [링크](#)
  - *The urllib.request module defines functions and classes which help in opening URLs (mostly HTTP) in a complex world.*
  - *--- basic and digest authentication, redirections, cookies and more.*
- *Python urllib tutorial for Accessing the Internet.* [링크](#)
  - 파이썬기반 웹페이지 접근 차단을 방지하기 위해 웹브라우저로 가장하여 접근하는 기법 예시
- **참고: Python3 requests 모듈.** [링크](#), [PyPI 링크](#)
  - Last released: May 4, 2015



# urllib 모듈 예제-사이트 저장

z\_urllib\_webpage.py

## ■ Web page source의 파일 저장

- `url = 'https://www.naver.com'`
- `f = urllib.request.urlopen(url)`
- `c = f.read().decode('utf-8') # type(c)=<class 'str'>`
  - `f.read()`의 출력 결과의 타입: `type(f.read()) = <class 'bytes'>`
- `f = open('z_source1.html', mode='wt', encoding='utf-8')`
- `f.write(c); f.close()`

# urllib 모듈 예제-사이트 영상 저장 및 출력

z\_urllib\_web\_image.py

## ■ 영상 저장

- `img = urllib.request.urlopen('https://xx.abc.com/cc.jpg').read()`
  - `type(img) = <class 'bytes'>`
- `f = open('efg.png', mode='wb')`
  - `# type(f)=<class '_io.TextIOWrapper'>`
  - 저장되는 파일의 형식은 이름에 관계없는 듯. 본 사례의 경우 실제로는 jpg 형식
- `f.write(img); f.close()`

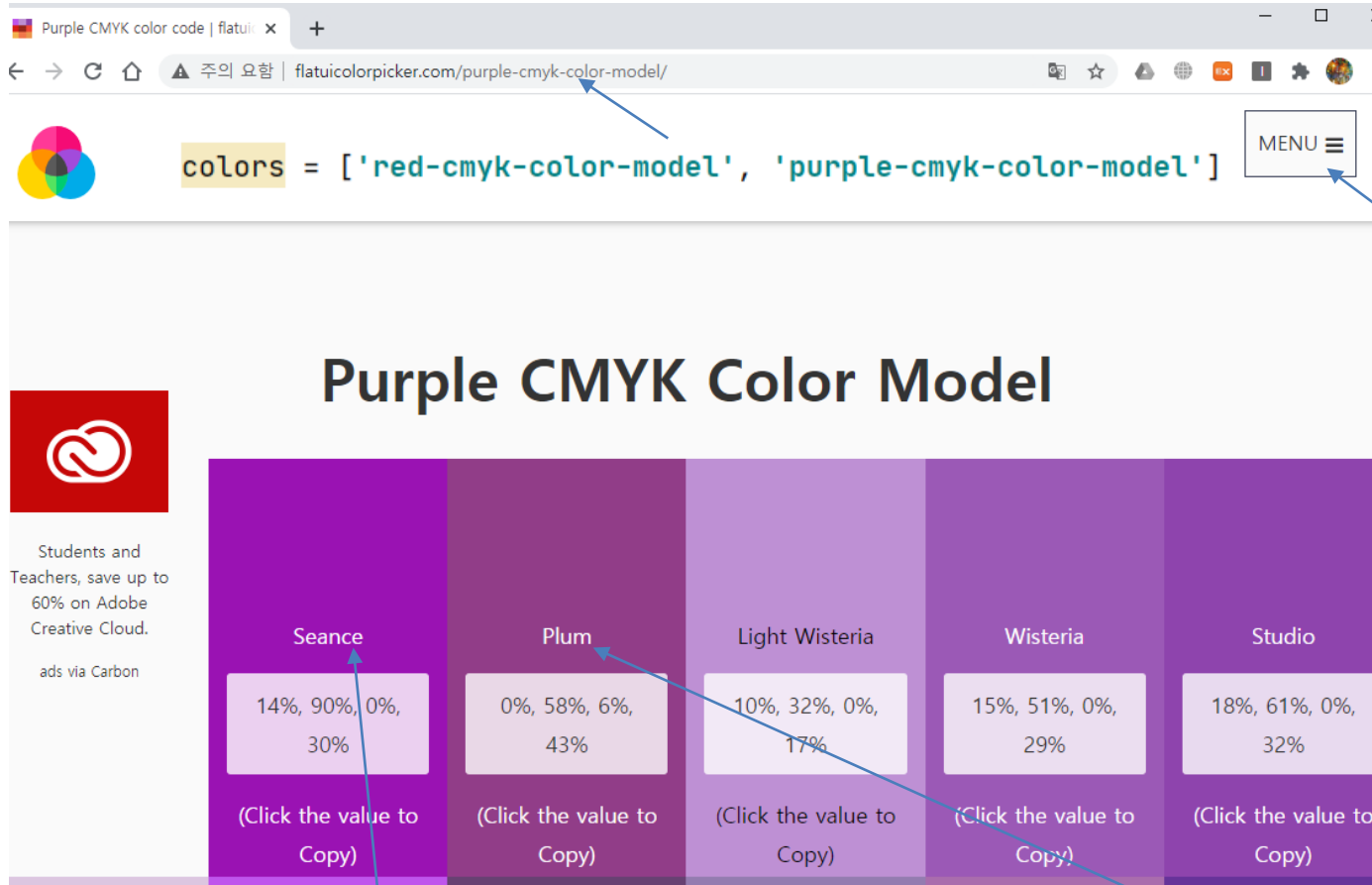
# urllib 모듈 예제(web crawling/scraping/spidering)

z\_crawling\_cmyk.py

- **Web scraping, web harvesting, or web data extraction** is [data scraping](#) used for [extracting data](#) from [websites](#).      *링크: [webscraping](#)*
- While web scraping can be done manually by a software user, the term typically refers to automated processes implemented using a [bot](#) or [web crawler](#).
- It is a form of copying, in which specific data is gathered and copied from the web, typically into a central local [database](#) or spreadsheet, for later [retrieval](#) or [analysis](#).
- **실습 1: 다음 URL에 올라있는 다양한 색상의 CMYK 칼라의 비율을 화면에 출력하고 파일에 저장한다.** <http://www.flatuicolorpicker.com>
  - 저장되는 파일의 이름: `_cmyk.txt`
  - 잉크젯/칼라 레이저 프린터의 잉크는 CMYK 4 종류의 color 토너로 구성되어 있습니다.
  - 본 예제는 다양한 색상이 만들어지기 위한 CMYK의 구성 성분을 %로 표시합니다. # CMYK는 각각 Cyan/Magenta/Yellow/Black 색상을 의미합니다.

# urllib 모듈 예제(web crawling/scraping/spidering)

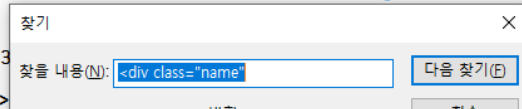
z\_crawling\_cmyk.py



여러 종류의 색 선택 가능

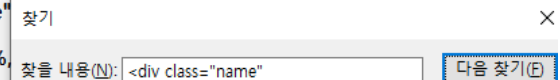
```
<div class="block-content">
<div class="name" style="color:#ffffff"> Seance</div>
<div class="value"
data-clipboard-target=".value"
data-clipboard-action="copy"
data-clipboard-text="14%, 90%, 0%, 30%"
14%, 90%, 0%, 30%
</div>
<div class="copy" style="color:#ffffff">
```

\_page.txt



```
<div class="block-content">
<div class="name" style="color:#ffffff"> Plum</div>
<div class="value"
data-clipboard-target=".value"
data-clipboard-action="copy"
data-clipboard-text="0%, 58%, 6%, 43%"
0%, 58%, 6%, 43%
</div>
```

\_page.txt



## 02. 모듈 만들기

### 여기서 잠깐! 파이썬 모듈 검색

- 이외에도 많은 파이썬 모듈이 있다. 그렇다면 이 모듈들은 어떻게 불러 사용할 수 있을까? 가장 좋은 방법은 구글에서 검색하는 것이다. 특히 영어로 검색하는 것이 좋다. 예를 들어, 프로그래밍이 걸린 시간을 쓰는 모듈을 찾고 싶다면 다음과 같은 방식으로 검색어를 입력하여 찾으면 된다.

```
python time module run time
```

- 다른 방법으로는 우리나라의 대표 사이트인 파이썬 코리아에 문의할 수 있다. 파이썬 코리아의 페이스북 페이지는 파이썬 개발자에게 많은 정보를 제공하는 대표적인 커뮤니티이다. 다음 주소에서 여러 질문을 하면 많은 개발자가 친절히 알려줄 것이다.

```
https://www.facebook.com/groups/pythonkorea
```

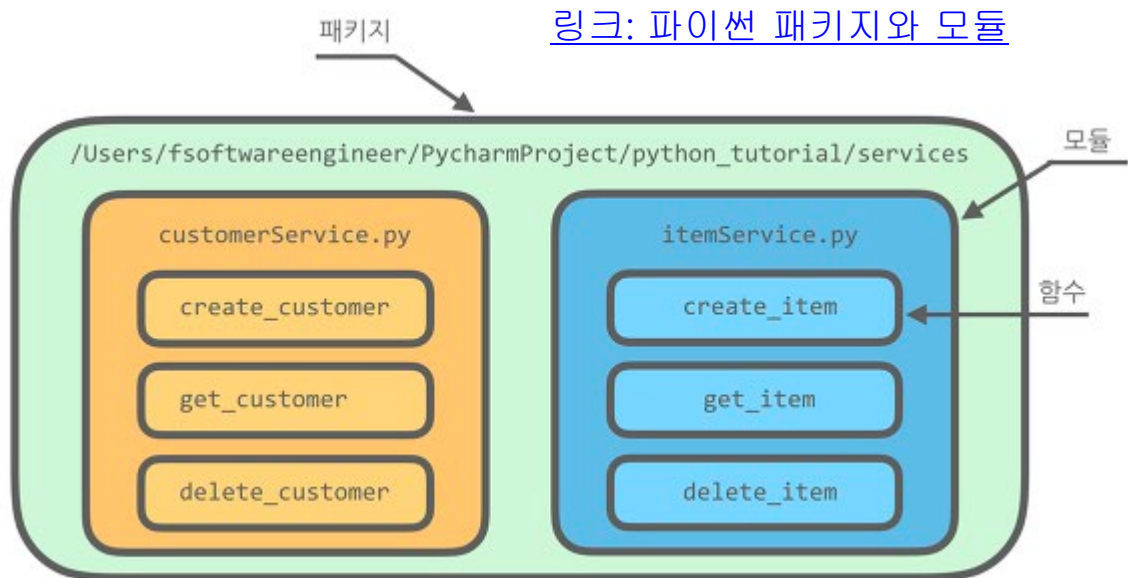
03

패키지 만들기

## 03. 패키지 만들기

### ■ 패키지의 구성

- 패키지는 하나의 대형 프로젝트를 수행하기 위한 모듈의 묶음이다. 모듈은 하나의 파일로 이루어져 있고, 패키지는 파일이 포함된 디렉터리(폴더)로 구성된다. 즉, 여러 개의 .py 파일이 하나의 디렉터리에 들어가 있는 것을 패키지라고 한다.
- 흔히 다른 사람이 만든 프로그램을 불러 사용하는 것을 라이브러리(library)라고 하는데, 파이썬에서는 패키지를 하나의 라이브러리로 이해하면 된다.
- 파이썬의 모듈을 구성할 때와 마찬가지로 패키지에도 예약어가 있다. 한 가지 주의할 점은 패키지에서는 파일명 자체가 예약어를 반드시 지켜야만 실행되는 경우가 있다. 따라서 패키지 내의 몇몇 파일에는 `__init__`, `__main__` 등의 키워드 파일명이 사용된다.



## 03. 패키지 만들기

패키지 폴더 roboadvisor

### ■ 패키지 만들기 실습 : 1단계: 디렉터리 구성하기

- 이번 실습에서 만들 패키지 이름은 'roboadvisor'이다. roboadvisor에는 세 가지 기능이 있다고 가정하자.
  - ① **crawling(크롤링)**: 주식 관련 데이터를 인터넷에서 가져오는 기능
  - ② **database(데이터베이스)**: 가져온 데이터를 데이터베이스에 저장하는 기능
  - ③ **analysis(분석)**: 해당 정보를 분석하여 의미 있는 값을 뽑는 기능

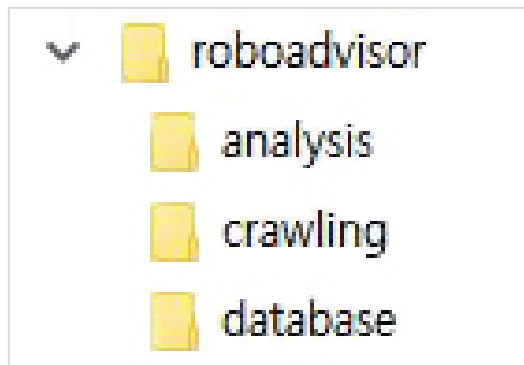


### 03. 패키지 만들기

패키지 폴더 roboadvisor

#### ■ 패키지 만들기 실습 : 1단계: 디렉터리 구성하기

- 패키지를 구성하기 위한 첫 번째 단계는 각 패키지의 세부 패키지에 맞춰 디렉터를 구성하는 것이다. 먼저 cmd 창에 다음 명령을 입력하여 디렉터를 생성한다



[roboadvisor 디렉터리 구성]

```
C:\ 명령 프롬프트

D:\workspace>mkdir roboadvisor

D:\workspace>cd roboadvisor

D:\workspace\roboadvisor>mkdir crawling

D:\workspace\roboadvisor>mkdir database

D:\workspace\roboadvisor>mkdir analysis

D:\workspace\roboadvisor>
```

[세부 패키지에 맞춰 디렉터리 생성]

### 03. 패키지 만들기

패키지 폴더 roboadvisor

#### ■ 패키지 만들기 실습 : 2단계: 디렉터리별로 필요한 모듈 만들기

- 만들어진 디렉터리에 필요한 모듈을 만든다. 하나의 패키지는 중첩된 구조로 만들 수 있으므로 패키지 안에 또 하나의 패키지가 들어갈 수 있다. 하지만 이렇게 각각의 디렉터리를 하나의 패키지로 선언하기 위해서는 예약된 파일을 만들어야 한다. 바로 `__init__.py`이다.

[패키지의 구조]



### 여기서 잠깐! 패키지의 구조 설계

- 앞의 패키지 구조는 임의로 작성한 것이다. 패키지의 구조를 만들기 위해 프로그램 개발자는 설계를 해야 한다. 하위 패키지 별로 해야 하는 일과 하위 패키지에 소속된 모듈들이 해야 할 일을 따로 정의해 각 모듈에 역할을 부여하는 것이다. 여기까지 가는 과정에는 많은 경험과 지식이 필요하다. 지금 단계에서는 어려울 수 있으니, 이 예제에서는 대략적인 구조와 역할을 임의로 작성한다.

## 03. 패키지 만들기

패키지 폴더 roboadvisor

### ■ 패키지 만들기 실습 : 2단계: 디렉터리별로 필요한 모듈 만들기

- 이제 각 하위 패키지에 포함된 모듈에 필요한 기능을 구현하기 위해 코딩을 하자.
- [코드 11-6]과 [코드 11-7]과 같은 방식으로 crawling 디렉터리 아래 parser.py와 scrap.py에, database 디렉터리 아래 connection.py와 query.py에 코드를 입력한다.

#### 코드 11-6 series.py(analysis 디렉터리)

```
1 def series_test():  
2     print("series")
```

#### 코드 11-7 statics.py(analysis 디렉터리)

```
1 def statics_test():  
2     print("statics")
```

## 03. 패키지 만들기

패키지 폴더 roboadvisor

### ■ 패키지 만들기 실습 : 2단계: 디렉터리별로 필요한 모듈 만들기

- 실제 해당 모듈을 사용하기 위해 다음과 같이 파이썬 셸에서 코드를 작성한다. 이 코드는 roboadvisor의 상위 디렉터리에서 파이썬 셸을 실행해야 정상적으로 진행된다.

```
>>> from roboadvisor.analysis import series
>>> series.series_test()
series
```

- ➡ 이 코드를 실행하면 roboadvisor 디렉터리 안에는 ' \_ \_pycache\_ \_'라는 디렉터리가 생성되는데, 이는 파이썬의 언어적 특성으로 생기는 결과이다. \_ \_pycache\_ \_ 디렉터리에는 해당 프로그램이 작동될 때 사용하기 위한 모듈들을 컴파일하고, 그 결과를 저장한다. 이렇게 한 번 \_ \_pycache\_ \_ 디렉터리가 생성되면 그 시점에서 해당 모듈을 수정해도 결과가 반영되지 않는다. 해당 프로그램 또는 파이썬 셸이 완전히 종료한 후 수정해야 해당 모듈의 결과를 반영할 수 있다. 인터프리터 언어이지만 내부적으로 컴파일도 하고, 효율적으로 사용하기 위한 여러 가지 작업이 있다는 것을 기억하기 바란다.

## 03. 패키지 만들기

패키지 폴더 roboadvisor

- 패키지 만들기 실습 : ✓ 생략 가능 3단계: 디렉터리별로 `__init__.py` 구성하기
- 디렉터리별로 `__init__.py` 파일을 구성한다. `__init__`은 해당 디렉터리가 파이썬의 패키지라고 선언하는 초기화 스크립트이다. `__init__.py` 파일은 파이썬의 거의 모든 라이브러리에 있다. 예를 들어, 대표적인 파이썬 머신러닝 라이브러리인 scikit-learn의 경우 다음과 같이 가장 상위 디렉터리부터 `__init__.py` 파일이 있는 것을 확인할 수 있다.

Branch: master ▾ scikit-learn / sklearn / `__init__.py`

 aboutcaud MNT Add sklearn show\_versions() method (#11596)

22 contributors 

97 lines (80 sloc) | 3.27 KB

```
1  """
2  Machine learning module for Python
```

[scikit-learn에서 확인할 수 있는 `__init__.py` 파일]

## 03. 패키지 만들기

패키지 폴더 roboadvisor

✓ 생략 가능

### ■ 패키지 만들기 실습 : 3단계: 디렉터리별로 `__init__.py` 구성하기

- `__init__.py` 파일은 패키지 개발자, 설치 시 확인해야 할 내용 등 메타데이터라고 할 수 있는 내용을 담고 있다. 하지만 가장 중요한 내용은 이 패키지의 구조이다. 일반적으로 `__init__.py` 파일에는 다음과 같이 해당 패키지가 포함된 모듈에 관한 정보가 있다. 다음 코드를 roboadvisor 디렉터리의 `__init__.py`에 입력한다

**코드 11-8** `__init__.py`(roboadvisor 디렉터리)

```
1 import analysis
2 import crawling
3 import database
4
5 __all__ = ['analysis', 'crawling', 'database']
```

- ➡ roboadvisor 디렉터리에는 3개의 하위 패키지, 즉 analysis, crawling, database가 있다. 이 각각의 패키지를 `__init__.py` 안에 `__all__`과 import문을 사용해 선언해야 한다. 따라서 `__all__`이라는 리스트형의 변수를 만들어 차례대로 하위 패키지의 이름을 작성하고, 같은 방법으로 각 하위 패키지를 import문으로 호출한다.

## 03. 패키지 만들기

패키지 폴더 roboadvisor

✓ 생략 가능

### ■ 패키지 만들기 실습 : 3단계: 디렉터리별로 `__init__.py` 구성하기

- 하위 패키지의 `__init__.py` 파일도 마찬가지이다. 예를 들어, `analysis` 디렉터리의 `__init__.py` 파일은 다음과 같이 각 패키지에 포함된 모듈명을 모두 작성해야 한다. 패키지로 표시하기 위해 꼭 해야 하는 작업이며 패키지별로 모두 처리해야 한다. `crawling`과 `database` 디렉터리의 `__init__.py` 파일에도 다음과 같은 방식으로 코드를 입력하고 저장한다.

코드 11-9 `__init__.py`(`analysis` 디렉터리)

```
1 from . import series          # 현재 패키지에서 series 모듈을 가져옴
2 from . import statics        # 현재 패키지에서 statics 모듈을 가져옴
3
4 __all__ = ['series', 'statics']
```



## 03. 패키지 만들기

패키지 폴더 roboadvisor

여기서  잠깐! `__init__.py` 파일을 만들지 않으면 어떤 문제가 발생할까?

- 파이썬 3.6 버전 이상에서는 `__init__.py` 파일을 만들지 않아도 큰 문제가 생기지 않는다. 하지만 파이썬 3.3 버전 이하에서는 `__init__.py`가 없을 경우 해당 디렉토리를 패키지로 인정하지 않는다. 물론 많은 사람이 상위 버전의 파이썬을 사용하기 때문에 문제 되지 않을 수도 있지만, 하위 버전의 파이썬을 사용할 수도 있으니 `__init__.py` 파일은 패키지를 만들 때 반드시 생성하도록 한다.

## 03. 패키지 만들기

### ■ 패키지 만들기 실습 : 4단계: `__main__.py` 파일 만들기

- 4단계에서는 패키지를 한 번에 사용하기 위해 `roboadvisor` 디렉터리에 `__main__.py` 파일을 만든다. => 패키지 폴더 밖에 이 패키지를 로드하는 메인 프로그램을 만드는 것이 더 일반적이다. 여기서는 편의상 패키지 동작을 점거하는 `__main__.py` 파일을 패키지 폴더 안에 만들어 보았다. 파일 이름은 `__main__.py`가 아니어도 상관없다.
- 지금까지 계속 파이썬 파일명 형태인 `.py` 파일로 실행하였다. 같은 방법으로 패키지 자체를 실행하기 위해 만들어야 하는 것이 `__main__.py` 파일이다. `__main__.py` 파일의 구성은 간단하다. 기본적으로 호출해야 하는 여러 모듈을 `from`과 `import`문으로 호출한 후, `if __name__ == '__main__':` 구문 아래에 실제 실행 코드를 작성하면 된다.

#### 코드 11-10 `__main__.py`

```
1 from analysis.series import series_test
2 from crawling.parser import parser_test
3
4 if __name__ == '__main__':
5     series_test()
6     parser_test()
```

## 03. 패키지 만들기

### ■ 패키지 만들기 실습 : 5단계: 실행하기(패키지 이름만으로 호출하기)

- 마지막 5단계에서는 해당 패키지를 실행한다. 모든 코드를 작성한 후, 해당 패키지의 최상위 디렉터리(본 예시에서는 roboadvisor의 상위 디렉터리)에서 'python 패키지명'을 입력하여 실행한다.

`python roboadvisor` ← 패키지를 실행하였다기 보다는 패키지를 load만 하였다.  
`series` ↗  
`parser` ↘    패키지를 load 하면 그 내부 모듈이 한번 수행된다. 이미 로드되었는데 또 로드하면 수행되지 않는다.

```
명령 프롬프트
D:\workspace\Ch11\roboadvisor>python roboadvisor
(null): can't open file 'roboadvisor': [Errno 2] No such file or directory

D:\workspace\Ch11\roboadvisor>cd..

D:\workspace\Ch11>python roboadvisor
series
parser
```

[패키지 실행]

## 03. 패키지 만들기

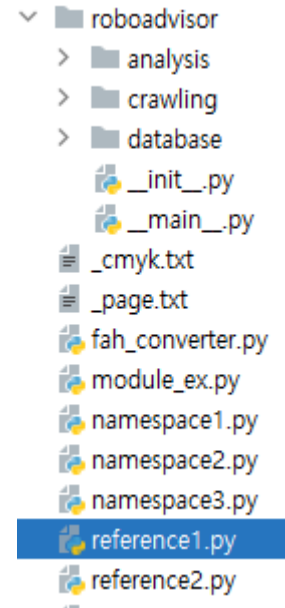
reference1.py

### ■ 패키지 네임스페이스 : 절대 참조

- 먼저 절대 참조의 예시부터 보자.

```
from roboadvisor.analysis import series
```

- ➔ 위 코드에서 from은 roboadvisor부터 시작한다. 즉, 패키지 이름부터 시작하여 series까지 모든 경로를 입력한다. 'from 전체 패키지.서브 패키지 import 모듈' 형식이다. 이렇게 전체경로를 모두 입력하는 것을 절대 참조라고 한다. \_\_init\_\_.py 파일을 만들 때도 절대 참조로 모듈을 호출하는 것이 좋다. 한 가지 주의할 점은 가장 상위에 있는 \_\_init\_\_.py 파일도 상위 디렉터리 roboadvisor를 넣는 것이 좋다.



```
series  
parser
```

```
from roboadvisor.analysis.series import series_test  
from roboadvisor.crawling.parser import parser_test  
  
if __name__ == '__main__':  
    series_test()  
    parser_test()
```

## 03. 패키지 만들기

보류: reference2.py

### ■ 패키지 네임스페이스 : 상대 참조

- 상대 참조의 핵심은 현재의 디렉터리를 기준으로 모듈을 호출하는 것이다.

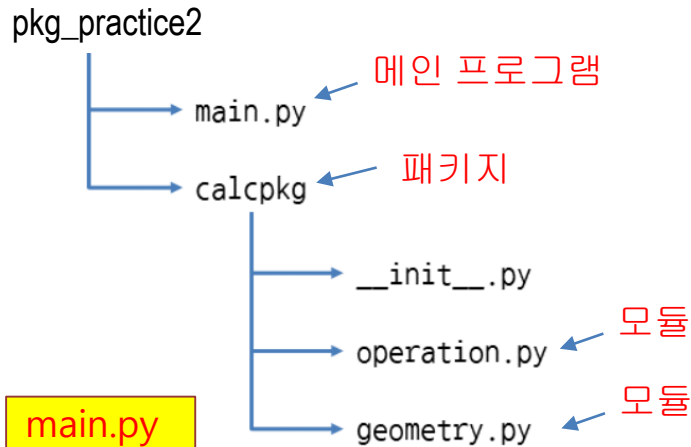
코드 11-12 reference2.py

```
1 from .series import series_test
2 from ..crawling.parser import parser_test
```

- 여기서 가장 중요한 코드는 .series와 ..crawling.parser이다. 먼저 점 1개 (.)는 현재 디렉터리를, 점 2개(..)는 부모 디렉터리를 뜻한다.

# 패키지 설치의 다른 사례(1/2) ■ 패키지 만들기

패키지 폴더 pkg\_practice2



calcpkg/operation.py

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

calcpkg/geometry.py

```
def triangle_area(base, height):  
    return base * height / 2  
  
def rectangle_area(width, height):  
    return width * height
```

```
import calcpkg.operation # calcpkg 패키지의 operation 모듈을 가져옴  
import calcpkg.geometry # calcpkg 패키지의 geometry 모듈을 가져옴  
  
print(calcpkg.operation.add(10, 20)) # operation 모듈의 add 함수 사용  
print(calcpkg.operation.mul(10, 20)) # operation 모듈의 mul 함수 사용  
  
print(calcpkg.geometry.triangle_area(30, 40)) # geometry 모듈의 triangle_area 함수 사용  
print(calcpkg.geometry.rectangle_area(30, 40)) # geometry 모듈의 rectangle_area 함수 사용  
  
from calcpkg.operation import add, mul  
print(add(11, 22))  
print(mul(11, 22))  
  
import calcpkg.operation as clc  
print(clc.add(100, 200))  
print(clc.mul(100, 200))
```

수행결과

```
30  
200  
600.0  
1200  
33  
242  
300  
20000
```

## 패키지 설치의 다른 사례(2/2) ■ 패키지 만들기

패키지 폴더 pkg\_practice2

```
def triangle_area(base, height):  
    return base * height / 2
```

geometry2.py

```
def rectangle_area(width, height):  
    return width * height
```

```
calcpkg.geometry2  
600.0  
1200  
600.0  
1200
```

수행결과

```
# 본 파일을 모듈로 활용하는 메인 프로그램은 main2.py입니다.  
# 모듈자격으로 수행하면 모듈 이름이 출력된다.  
# calcpkg.geometry2      <= __name__ = 'calcpkg.geometry2'  
print(__name__)
```

```
# 메인 루틴의 자격으로 수행하면 다음 메시지가 출력된다.  
# __main__      <= __name__ = '__main__'  
# This is being executed as main status.  
if __name__ == '__main__':  
    print('This is being executed as main status.')
```

```
import calcpkg.geometry2 as gm # 'calcpkg.geometry2' 출력됨
```

main2.py

```
print(gm.triangle_area(30, 40)) # geometry 모듈의 triangle_area 함수 사용  
print(gm.rectangle_area(30, 40)) # geometry 모듈의 rectangle_area 함수 사용
```

```
import calcpkg.geometry2 # 한번 더 로드하였으나 메시지는 출력되지 않음  
print(calcpkg.geometry2.triangle_area(30, 40)) # geometry2 모듈의 triangle_area 함수 사용  
print(calcpkg.geometry2.rectangle_area(30, 40)) # geometry2 모듈의 rectangle_area 함수 사용
```

# 모듈과 패키지의 경로

pip\_install\_path.py

- 파이썬에서는 모듈, 패키지를 찾을 때 일단 현재 폴더에서 먼저 찾는다.
- 실패하면 다음과 같이 sys 모듈의 path 변수(즉, sys.path)에 지정한 경로를 사용한다.

```
In[2]: import sys
In[3]: sys.path
Out[3]:
['C:\\Program Files\\JetBrains\\PyCharm Community Edition 2018.3.5\\helpers\\pydev',
'C:\\Program Files\\JetBrains\\PyCharm Community Edition 2018.3.5\\helpers\\third_party\\thriftpy',
'C:\\Program Files\\JetBrains\\PyCharm Community Edition 2018.3.5\\helpers\\pydev',
'C:\\Python\\python37.zip',
'C:\\Python\\DLLs',
'C:\\Python\\lib',
'C:\\Python',
'C:\\Users\\KJH\\AppData\\Roaming\\Python\\Python37\\site-packages',
'C:\\Python\\lib\\site-packages',
'C:\\Python\\lib\\site-packages\\IPython\\extensions',
'D:\\Work\\@@PythonProgramming\\LectureMaterials\\10_(ch11)_모듈과 패키지\\Ch11',
'D:/Work/@@PythonProgramming/LectureMaterials/10_(ch11)_모듈과 패키지/Ch11']
```

- 여기서 site-packages 폴더에는 pip로 설치한 패키지가 들어간다. 그리고 자기가 만든 모듈, 패키지도 site-packages 폴더에 넣으면 스크립트 파일이 어디에 있든 모듈, 패키지를 사용할 수 있다.
- 만약 가상 환경(virtual environment)를 만들어서 모듈과 패키지를 관리한다면 가상 환경/Lib/site-packages 폴더에 모듈과 패키지가 들어간다.



# sys.path의 다른 사례

pip\_install\_path.py

```
In[2]: import sys
```

```
In[3]: sys.path
```

```
Out[3]:
```

```
['Q:\\win10_Programs\\PyCharm Community Edition 2018.2.2\\helpers\\pydev',  
 'D:\\Work\\@@PythonProgramming\\LectureMaterials',  
 'Q:\\win10_Programs\\PyCharm Community Edition 2018.2.2\\helpers\\pydev',  
 'C:\\Python\\Python3.7.0\\python37.zip',  
 'C:\\Python\\Python3.7.0\\DLLs',  
 'C:\\Python\\Python3.7.0\\lib',  
 'C:\\Python\\Python3.7.0',  
 'C:\\Python\\Python3.7.0\\lib\\site-packages',  
 'C:\\Python\\Python3.7.0\\lib\\site-packages\\IPython\\extensions',  
 'D:\\Work\\@@PythonProgramming\\LectureMaterials',  
 'D:/Work/@@PythonProgramming/LectureMaterials']
```

# 모듈과 패키지의 경로

pip\_install\_path.py

- 커맨드 창에서 pip로 설치한 패키지의 파이썬 path 알아내기
- `python -m site --user-site`

```
C:\>python -m site --user-site  
C:\Users\KJH\AppData\Roaming\Python\Python37\site-packages
```

- 프로그램 상에서 pip로 설치한 패키지의 파이썬 path 알아내기
- `import site`
- `print (site.getsitepackages())`

```
['C:\\Python', 'C:\\Python\\lib\\site-packages']
```

04

가상환경 사용하기

## 04. 가상환경 사용하기

### ■ 가상환경의 개념

- 일반적으로 어떤 프로젝트를 수행할 때는 파이썬 코드를 수행할 기본 인터프리터에서 추가로 프로젝트별로 필요한 패키지를 설치한다. 이러한 패키지를 설치할 때 서로 다른 프로젝트가 영향을 받지 않도록 독립적인 프로젝트 수행 환경을 구성하는데, 이를 가상환경이라고 한다.

가상환경 도구	특징
virtualenv + pip	<ul style="list-style-type: none"><li>가장 대표적인 가상환경 관리 도구</li><li>레퍼런스와 패키지가 가장 많음</li></ul>
conda	<ul style="list-style-type: none"><li>상용 가상환경 도구인 miniconda의 기본 가상환경 도구</li><li>설치가 쉬워 윈도우에서 유용함</li></ul>

[가상환경 도구]

## 04. 가상환경 사용하기

### ■ 가상환경 설정하기 : 가상환경 만들기

- 가상환경을 만드는 명령어는 다음과 같다.

```
conda create -n my_project python=3.4
```

conda create   -n my\_project   python=3.4

↑  
가상환경 새로 만들기

↑  
가상환경 이름

↑  
파이썬 버전

[가상환경 만들기 명령어]

## 04. 가상환경 사용하기

### ■ 가상환경 설정하기 : 가상환경 만들기

```
명령 프롬프트
D:\workspace>conda create -n my_project python=3.4
Solving environment: done

## Package Plan ##

environment location: C:\Miniconda3\envs\my_project

added / updated specs:
- python=3.4

The following packages will be downloaded:

package                        | build                | size
-----|-----|-----
setuptools-27.2.0              | py34_1               | 762 KB
vc-10                           | 0                    | 702 B
vs2010_runtime-10.00.40219.1 | 2                    | 1.1 MB
python-3.4.5                   | 0                    | 22.9 MB
wheel-0.29.0                   | py34_0               | 123 KB
pip-9.0.1                      | py34_1               | 1.7 MB
-----|-----|-----
Total:                          |                      | 26.6 MB

The following NEW packages will be INSTALLED:

pip: 9.0.1-py34_1
python: 3.4.5-0
setuptools: 27.2.0-py34_1
vc: 10-0
vs2010_runtime: 10.00.40219.1-2
wheel: 0.29.0-py34_0

Proceed ([y]/n)? y
```

[가상환경 설치 명령 입력]

## 04. 가상환경 사용하기

### ■ 가상환경 설정하기 : 가상환경 실행하기

```
#  
# To activate this environment, use:  
# > activate my_project  
#  
# To deactivate an active environment, use:  
# > deactivate  
#  
# * for power-users using bash, you must source
```

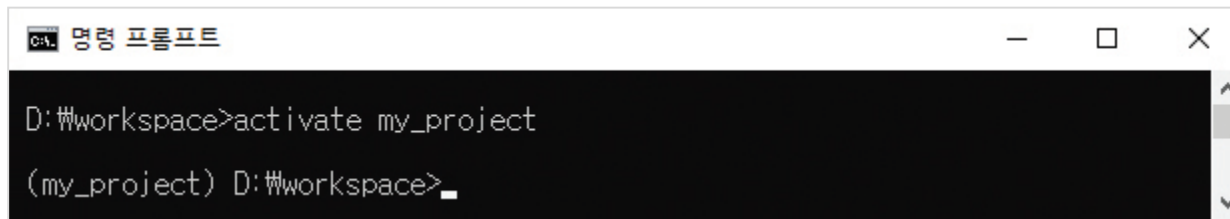
[가상환경 설치 화면]

```
activate my_project
```

- 이 코드는 my\_project라는 가상환경을 활성화(activate)하라는 뜻이다. 구성된 가상환경의 이름을 activate 다음에 넣으면 해당 가상환경이 실행되고, 프롬프트 앞에(my\_project)라는 가상환경 이름이 붙는다. 이제 이 환경에서는 가상환경의 인터프리터만 실행된다.

## 04. 가상환경 사용하기

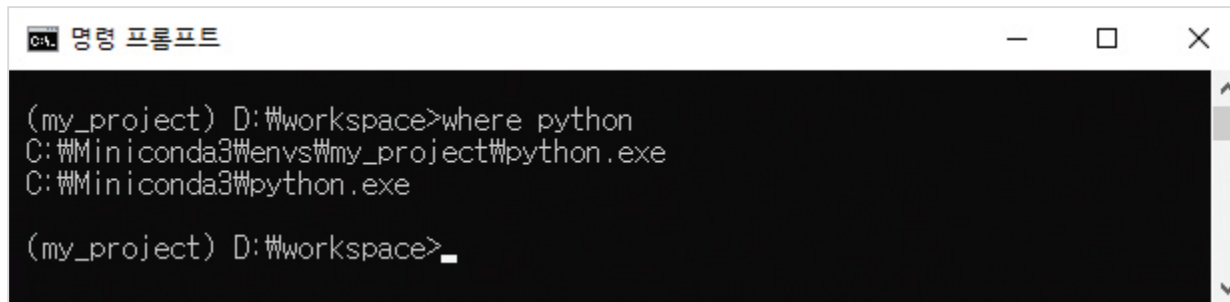
### ■ 가상환경 설정하기 : 가상환경 실행하기



```
C:\> 명령 프롬프트
D:\workspace>activate my_project
(my_project) D:\workspace>_
```

[가상환경 활성화]

- 이 상태에서 'where python'을 입력하면 현재 실행되는 파이썬의 위치가 어디인지 출력된다.



```
(my_project) D:\workspace>where python
C:\Miniconda3\envs\my_project\python.exe
C:\Miniconda3\python.exe
(my_project) D:\workspace>_
```

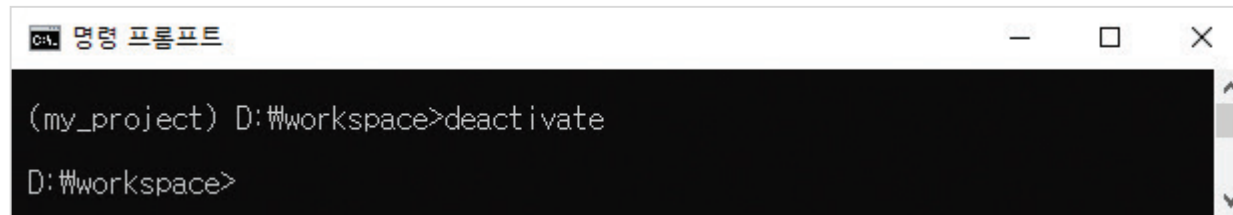
[파이썬의 위치 확인]



## 04. 가상환경 사용하기

### ■ 가상환경 설정하기 : 가상환경 실행하기

- 실행된 가상환경을 종료하기 위해서는 'deactivate'를 입력하면 된다.



```
(my_project) D:\workspace>deactivate
D:\workspace>
```

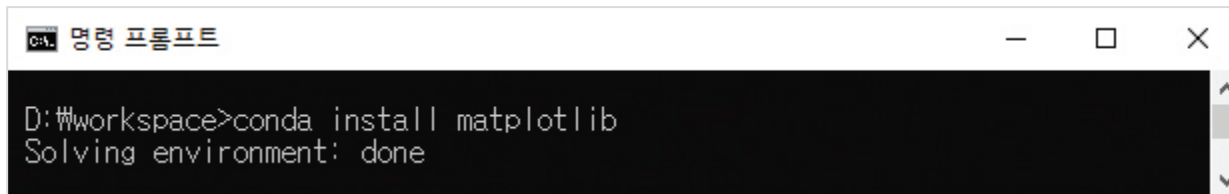
[가상환경 종료]

## 04. 가상환경 사용하기

### ■ 가상환경 설정하기 : 가상환경 패키지 설치하기

- 가상환경의 실행을 완료했으니 해당 가상환경에서 새로운 패키지를 설치해야 한다. 패키지를 설치하기 위해서는 다음과 같은 명령어를 입력한다.

```
conda install matplotlib
```



A screenshot of a Windows Command Prompt window. The title bar shows 'C:\> 명령 프롬프트' (C:\> Command Prompt). The command prompt shows the command 'D:\workspace>conda install matplotlib' and the output 'Solving environment: done'. The window has standard Windows window controls (minimize, maximize, close) in the top right corner.

```
C:\> 명령 프롬프트
D:\workspace>conda install matplotlib
Solving environment: done
```

[matplotlib 설치]

## 04. 가상환경 사용하기

### ■ 가상환경 설정하기 : 가상환경 패키지 실습하기

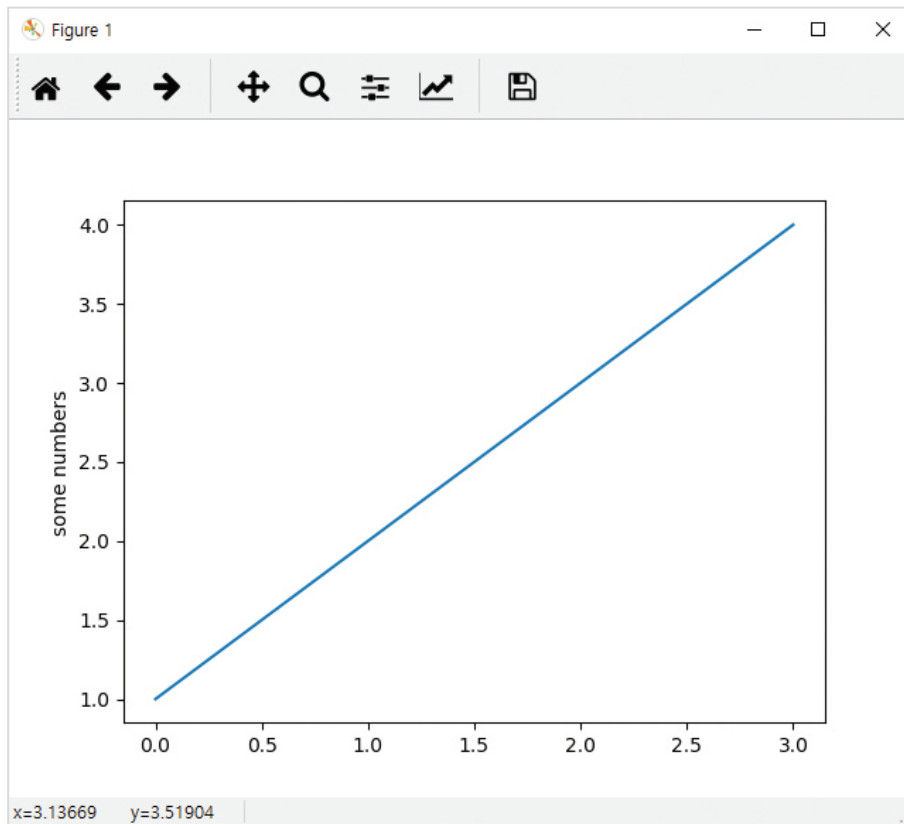
- 설치된 패키지를 실행해 보자. 앞에서 설치한 matplotlib은 대표적인 파이썬 그래프 관리 패키지로, 엑셀과 같은 그래프를 화면에 출력한다. 데이터 분석을 할 경우, 다양한 데이터 분석 도구와 함께 사용한다.

```
>>> import matplotlib.pyplot as plt
>>> plt.plot([1, 2, 3, 4])
[<matplotlib.lines.Line2D object at 0x000001E8CC52C080>]
>>> plt.ylabel('some numbers')
Text(0, 0.5, 'some numbers')
>>> plt.show()
```

## 04. 가상환경 사용하기

### ■ 가상환경 설정하기 : 가상환경 패키지 실습하기

- 코드를 실행하면 다음과 같은 깔끔한 그래프 화면을 확인할 수 있다. matplotlib은 논문을 쓰거나 여러 가지 데이터 분석 결과를 보여 줄 때 매우 유용한 모듈이다.



[matplotlib 실행 결과 화면]

## 04. 가상환경 사용하기

### 여기서 잠깐! jupyter 패키지

- 다른 패키지로 데이터를 분석할 때 매우 유용한 패키지로 jupyter가 있다. 먼저 패키지를 설치하기 위해 cmd 창에서 다음과 같은 명령어를 입력한다.

```
conda install jupyter
```

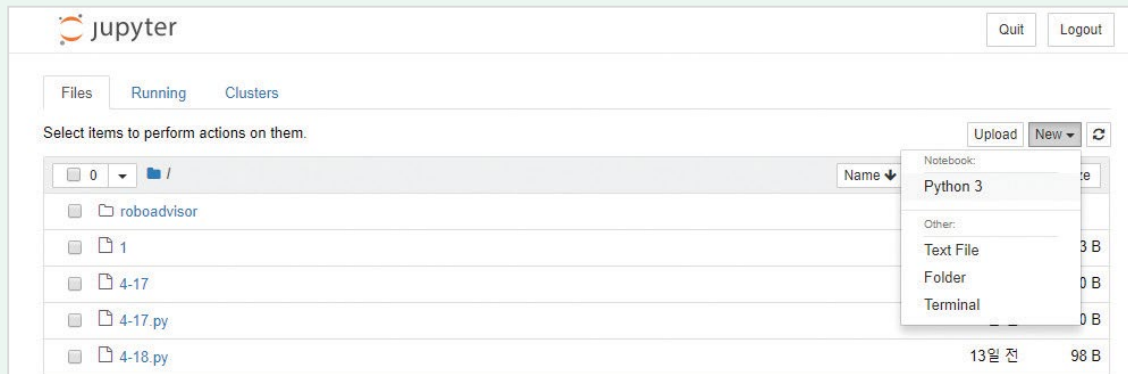
- 다음과 같이 설치한 후, 'jupyter notebook' 을 입력하여 실행하면 jupyter 환경에서 코딩할 수 있다.

```
jupyter notebook
```

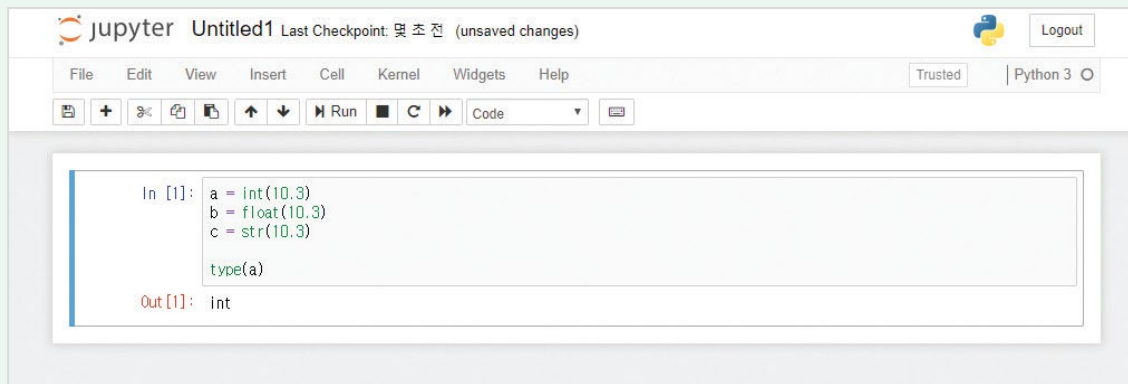
- jupyter를 실행하면 웹에서 코딩할 수 있는 환경이 나온다. 여기에서 [New] 버튼을 클릭하여 새로운 notebook을 생성한 후, 코딩하고 Ctrl + Enter 를 누르면 결과를 볼 수 있다.

## 04. 가상환경 사용하기

여기서  잠깐! jupyter 패키지



[jupyter 메인 화면]



[jupyter 웹 코딩]

# Thank You !