2018.12

**How to use the Nginx, Flask with uWSGI**

Revision History

|  |  |
| --- | --- |
| 문 서 번 호 |  |
| 문 서 명 |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 개정번호 | 개정일자 | 개정 내용 요약 | 작성자 | 확인자 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Table of Contents

[1. Nginx의 개요 1](#_Toc532992328)

[1.1. Nginx(웹서버)의 역할 1](#_Toc532992329)

[1.1.1. 정적 파일을 처리하는 HTTP 서버로서의 역할 1](#_Toc532992330)

[1.1.2. 응용프로그램 서버에 요청을 보내는 리버스 프록시로서의 역할 2](#_Toc532992331)

[1.2. AWS에 nginx 설치하기 3](#_Toc532992332)

[1.2.1. 설치 및 제거 3](#_Toc532992333)

[1.2.2. Nginx 경로 4](#_Toc532992334)

[1.3. Nginx 디렉토리 구조 살펴보기 4](#_Toc532992335)

[1.4. nginx.conf 기본 환경설정 튜닝하기 5](#_Toc532992336)

[1.4.1. 최상단 (Core 모듈) 6](#_Toc532992337)

[1.4.2. events 블락 7](#_Toc532992338)

[1.4.3. http 블락 7](#_Toc532992339)

[1.4.4. 기타 8](#_Toc532992340)

[1.4.5. 설정 파일의 반영 8](#_Toc532992341)

[1.4.6. 참고 사이트 8](#_Toc532992342)

List of Tables

List of Figures

# Nginx의 개요

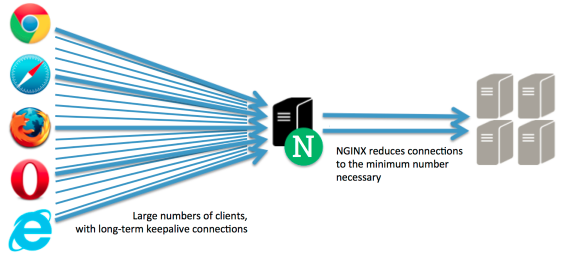
엔진엑스(Nginx)는 lgor Sysoev라는 러시아 개발자가 동시접속 처리에 특화된 웹서버 프로그램이다. Apache보다 동작이 단순하고, 전달자 역할만 하기 때문에 동시접속 처리에 특화되어 있다.

동시접속자(약 700명) 이상이라면 서버를 증설하거나 Nginx 환경을 권장한다고 한다. 지금은 아파치가 시장 점유율이 압도적이지만, 아마존웹서비스(AWS)상에서는 시장 점유율 44%에 달할정도로 가볍고, 성능이 좋은 엔진이라고 한다.

본 문서에는 AWS 인스턴스 상에서 Nginx를 설치하고, 기본적인 설정파일들을 알아보는 시간을 가질 것이다. .

참조 URL : http://whatisthenext.tistory.com/123

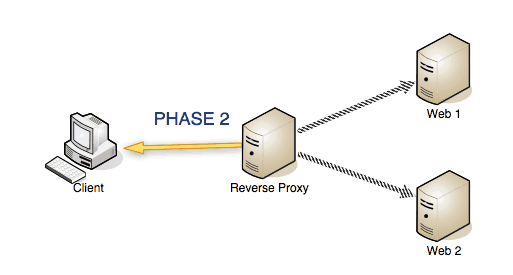
## Nginx(웹서버)의 역할



### 정적 파일을 처리하는 HTTP 서버로서의 역할

웹서버의 역할은 HTML, CSS, Javascript, 이미지와 같은 정보를 웹 브라우저(Chrome, lexplore, Opera, Firefox 등)에 전송하는 역할을 한다. (HTTP 프로토콜 준수)

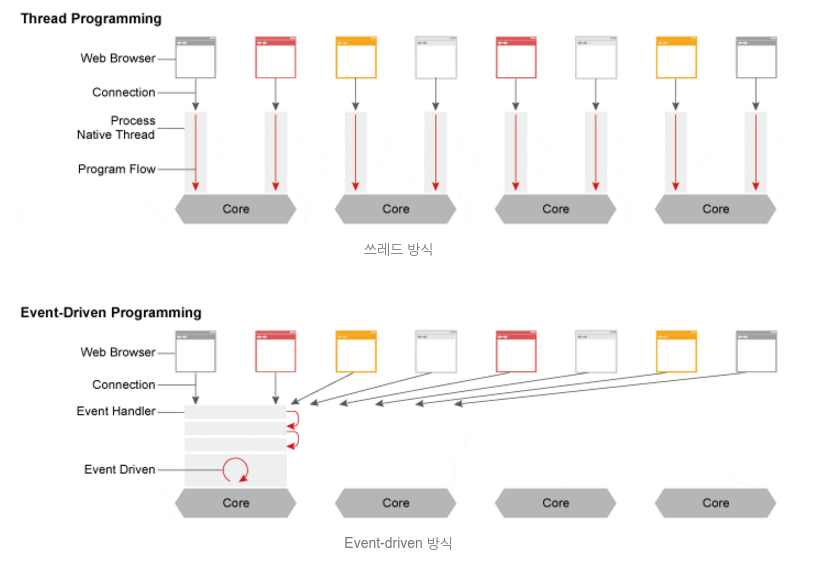
### 응용프로그램 서버에 요청을 보내는 리버스 프록시로서의 역할



두번째 역할은 리버스 프록시(reverse proxy) 인테, 한마디로 말하면 클라이언트는 가짜 서버에 요청(request)하면, 프록시 서버가 배후 서버(reverse server)로부터 데이터를 가져오는 역할을 한다. 여기서 프록시 서버가 Nginx, 리버스 서버가 응용프로그램 서버를 의미한다.

웹 응용프로그램 서버에 리버스 프록시(Nginx)를 두는 이유는 요청(request)에 대한 버퍼링이 있기 때문이다. 클라이언트가 직접 App 서버에 직접 요청하는 경우, 프로세스 1개가 응답 대기 상태가 되어야만 한다. 따라서 프록시 서버를 둠으로써 요청을 배분하는 역할을 한다.

Nginx.conf파일에서 location 지시어를 사용하여 요청을 배분한다.



Nginx는 비동기 처리방식(Event-Drive) 방식을 채택하고 있다.

## AWS에 nginx 설치하기

OS : Ubuntu 16.04 LTS

본 문서는 AWS 인스턴스(Ubuntu 16.04 LTS)에서 설치를 진행한다.

### 설치 및 제거

|  |
| --- |
| sudo apt-get install nginx # 설치  sudo apt-get remove nginx # 제거 |

|  |
| --- |
| $> nginx –v  nginx version : nginx/1.13.2 |

nginx version : nginx/1.13.2

nginx –v 했을 때 위와 같은 메시지가 출력할 것이다.

### Nginx 경로

* apt-get을 이용한 패키지 설치 방법을 이용하게 되면 기본적으로 /etc/nginx 폴더에 설치된다. 본 문서는 이 방식을 사용한다.
* 직접 compile한 경우에는 /usr/local/nginx/conf 또는 /usr/local/etc/nginx 에 설치된다.

|  |
| --- |
| sudo find / -name nginx.conf |

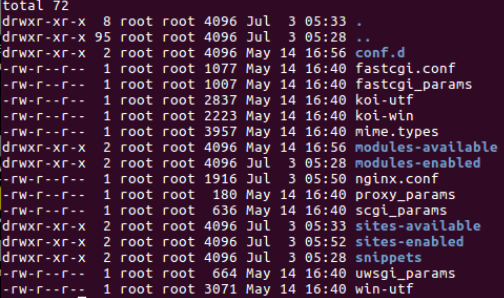
위 명령어를 이용해 nginx가 설치된 경로를 찾아본다.

## Nginx 디렉토리 구조 살펴보기

sudo find / -name nginx.conf

명령어를 통해 nginx.conf 가 위치한 경로를 이동한다.

|  |
| --- |
| cd /etc/nginx  ls –al |



|  |
| --- |
| |-- conf.d # (디렉토리) nginx.conf에서 불러들일 수 있는 파일을 저장  |-- fastcgi.conf # (파일) FASTCGI 환경설정 파일  |-- fastcgi\_params  |-- koi-utf  |-- koi-win  |-- mime.types  |-- nginx.conf # 접속자 수, 동작 프로세스 수 등 퍼포먼스에 관한 설정들  |-- proxy\_params  |-- scgi-params  |-- sites-available # 비활성화된 사이트들의 설정 파일들이 위치한다.  |-- default  |-- sites-enabled # 활성화된 사이트들의 설정 파일들이 위치한다. 존재하지 않은 경우에는 디렉토리를 직접 만들 수도 있다.  |-- default -> /etc/nginx/sites-avilable/default  |-- snippets  |-- fastcgi-php.conf  |-- snakeoil.conf  |-- uwsgin\_params  |-- win-utf |

트리명령을 사용하려면 sudo apt-get install tree를 실시한다. 설치가 완료되면 tree-d 또는 tree .로 확인 가능하다.

## nginx.conf 기본 환경설정 튜닝하기

가장 핵심이 되는 nginx.conf 파일을 먼저 살펴보자. nginx.conf 파일은 Nginx가 동작해야 할 방식을 설정 값을 통해 지정한다.

이 파일은 root 계정만 수정이 가능하기 때문에, 수정이 필요하다면 sudo vim nginx.conf를 통해 파일을 열어야 한다. 명령어를 통해 nginx.conf 파일을 열어보면 아래와 같은 형식을 갖는다. 얼핏보면 json 같기도 하다.



### 최상단 (Core 모듈)

|  |
| --- |
| user nginx; # (디폴트값 : www-data)  worker\_processes 1;  error\_log /var/log/nginx/error.log warn;  pid /var/run/nginx.pid |

1. user : NGINX 프로세스가 실행되는 권한

* nginx는 마스터(master)와 워커(worker) 프로세스로 나눈다.
* 워커 프로세스가 실질적인 웹서버 역할을 수행한다.
* user 지시어는 워커 프로세스의 권한을 지정한다.
* 워커프로세스를 악의적 사용자가 제어하면 해당 머신을 최고 사용자의 권한으로 원격제어하는 것이기 때문에 위험하다.
* 본 문서는 nginx로 변경하였다.

1. work\_processes : NGINX 프로세스 실행 가능 수

* 위에서 언급한 워커프로세스이다. 실질적인 웹서버 역할을 한다.
* auto도 무방하지만, 명시적으로 서버에 장착되어 있는 코어 수 만큼 할당하는 것이 보통이며, 더 높게도 설정 가능하다.

1. pid : NGINX 마스터 프로세스 ID 정보가 저장된다.

### events 블락

|  |
| --- |
| events {  worker\_connections 1024;  # multi\_accept on; (디폴트값 : off)  } |

* NGINX의 특징인 비동기 이벤트 처리 방식에 대한 옵션을 설정한다.
* Worker\_connections는 하나의 프로세스가 처리할 수 있는 커넥션의 수를 의미한다.
* 최대 접속자수 = worker\_processes X worket\_connections가 된다.

### http 블락

|  |
| --- |
| http {  include /etc/nginx/mime.types;  default\_type application/octet-stream;  log\_format main ‘$remote\_addr - $remote\_user [$time\_local] “$request” ‘  ‘$status $body\_bytes\_sent “$http\_referer” ‘  ‘”$http\_user\_agent” “$http\_x\_forwarded\_for”’;  access\_log /var/log/nginx/access.log main;  send\_file on;  #tcp\_nopush on;  Keepalive\_timeout 65;  #gzip on;  Include /etc/nginx/conf.d/\*.conf;  } |

* keepalive\_timeout : 접속시 커넥션을 몇 초동안 유지할지에 대한 설정값. 이 값이 높으면 불필요한 커넥션(접속)을 유지하기 때문에 낮은값 또는 0을 권장한다. (default = 10)
* servers token : NGINX의 버전을 숨길것인가에 대한 옵션이다. 보안상 주석을 제거하여 설정하는 것이 좋다.
* Types\_hash\_max\_size, server\_names\_hash\_bucket\_size 호스트의 도메인 이름에 대한 공간을 설정하는 것으로 이 값이 낮을 경우 많은 가상 호스트 도메인을 등록한다거나, 도메인 이름이 길 경우 bucket 공간이 모자라 에러가 생길 수 있으므로 넉넉하게 설정한다.

### 기타

include옵션 : 가상 호스트 설정이나 반복되는 옵션 항목을 include를 통해 불러올 수 있다. Ex) 리버스 프록시를 각 도메인에 설정한다고 했을 때 헤더 처리 옵션등을 conf.d 디렉토리에 넣어 두고 include 명령을 통해 불러 올 수 있다.

### 설정 파일의 반영

설정 파일의 내용을 변경한 후에는 NGINX에 반영해야 하는데, reload 명령을 이용한다.

|  |
| --- |
| sudo service nginx reload; |

### 참고 사이트

#### 생활 코딩 <https://opentutorials.org/module/384/3462> (2013.05.06)

* 초심자 수준에서 가장 쉽게 설명한 사이트이다. NGINX 뿐만 아니라, 네트워크 환경(웹, 웹서버, 서버와 클라이언트)에 대해 자세히 설명하였다.
* 훌륭한 강의의지만, 4년이 지났기 때문에 초반부터 살펴보고 좀더 최신의 포스팅을 볼 수 있다.

#### NGINX 기본 환경 설정 튜닝 및 설명 <https://extrememanual.net/9976> (2017.03.08)

* nginx 디렉토리 및 nginx.conf 파일 설정값에 대한 도움을 많이 받을 수 있음
* 현재로서 가장 최신 글이므로 참고할 만함.

# Python Flask, Nginx with uWSGI

기존의 웹서비스가 있는 상태에서 Python으로 작은 프로그램을 개발했는데 생각해보니 이를 웹서비스와 연동해서 사용해야 했고, PHP 기반의 웹서버에 해당 프로그램을 동시에 돌리면서 처리하기에는 부하가 생길 것 같다. 검색해보니 Rest API로 쓸 수 있는 파이썬 Flask라는 프레임워크를 찾을 수 있다.

참조 URL : https://blog.geuneul.com/archives/573

## Python Flask 서버 기본 환경

현재 웹서비스가 돌아가고 있는 환경은 네이버 클라우드 플랫폼(Naver Cloud Platform)으로, OS로는 Ubuntu 16.04 LTS 버전을 초기 생성했다는 가정하에 아래의 내용을 진행한다. 그리고, root 계정을 제공하므로 sudo 명령어는 제외하였다.

|  |
| --- |
| root@flask:~# |

## Package Installation

초기 상태에서 패키지 목록을 업데이트하고 설치하는 과정을 진행한다.

|  |
| --- |
| apt-get update  apt-get –y upgrade |

이렇게 하고 나서 명령어로 python을 입력해보면 python3가 설치되어 있는 것을 확인할 수 있다.

|  |
| --- |
| root@flask:~# python  The program ‘python’ can be found in the following packages:  \* python-minimal  \* python3 |

python3을 그대로 사용할 것이기에 아래의 명령어를 입력해 pip를 설치해준다.

|  |
| --- |
| apt-get install python3-pip |

Flast를 사용할 때 파이썬 패키지는 virtualenv에서 설치할 것이므로 아래의 명령어를 입력하여 설치를 진행한다.

|  |
| --- |
| pip3 install virtuallenv |

이제 아래의 명령어를 입력하여 Nginx를 설치한다.

|  |
| --- |
| apt-get install nginx |

## Nginx 폴더와 Flask 기본 설정하기

우선 Nginx의 기본 경로인 /var/www/html을 놔두고 아래의 명령어를 입력하여 새로운 폴더를 생성한다. 그리도 새로 만든 폴더로 이동한다.

|  |
| --- |
| mkdir /var/www/mysite  cd /var/www/mysite |

Flask를 설치할 공간을 만들기 위해 아래의 명령어를 입력하여 가상 개발 환경을 구축하고, Nginx와의 연결을 위해 uWSGI를 설치한다.

|  |
| --- |
| virtualenv .env  source .env/bin/activate  pip install flask  pip install uwsgi |

영문사이트의 많은 예제들이 apt-get install uwsgi로 설정을 진행하는데 이게 생각보다 잘 안되어 국내 블로그(<http://bablabs.tistory.com/25>)를 참고한다.

그리고 여기까지 진행했을 때, 현재 경로는 /var/www/mysite인데, 여기에 run.py, uwsgi.ini 2개의 실행 파일을 작성하여 웹서비스가 구동될 수 있도록 해준다.

|  |
| --- |
| **run.py**  from flask import Flask  app = Flask(\_\_name\_\_)  @app.route(‘/’)  def hello\_world():  Return ‘Hello World!’  If \_\_name\_\_ == ‘\_\_main\_\_’:  app.run() |

|  |
| --- |
| **uwsgi.ini**  [uwsgi]  chdir = /var/www/mysite  module = run  virtuallenv = /var/www/mysite/.env  callable = app  chmod-socket = 666  socket = /tmp/mysite.sock |

(.env)가 활성화된 상태로 uwsgi uwsgi.ini를 실행하게 되면 정상적으로 실행되는 것을 확인할 수 있는데, 이후 작업으로 /etc/init/ 폴더에 아래와 같이 conf 파일을 입력 후 백그라운드에서 동작할 수 있게 설정해주어야 한다.

|  |
| --- |
| mysite.conf  description “uWSGI server instance configured to serve mysite”  start on runlevel [2345]  stop on runlevel [!2345]  setuid root  setgid root  env PATH=/var/www/mysite/.env/bin  chkdir /var/www/mysite  exec uwsgi –ini uwsgi.ini |

그리고 나서 해당 사항을 적용하기 위해 아래와 같이 입력한다.

|  |
| --- |
| start mysite.conf |

아마 아래와 같은 내용이 뜨면서 적용이 안 될 텐데, 문장에 설명이 있으나 그대로 진행하면 안되고 아래의 문구에 있는 명령어를 입력하여 진행해야 한다.

|  |
| --- |
| The program ‘start’ is currently not installed. You can install it by typing: apt install upstart |

|  |
| --- |
| apt-get install upstart-sysv  update-initramfs –u  reboot |

위의 코드에서 2번째 줄을 실행한 뒤에는 reboot 명령어를 통해 재부팅해야 적용이 완료된다. 다음은 Nginx 설정을 위해 폴더를 이동하고 기존의 default 파일을 제거한다. 그리고 mysite 파일을 만든 후 활성화시키기 위해 심볼릭 파일을 생성한다.

|  |
| --- |
| cd /etc/nginx/sites-available  rm –rf default  vi mysite |

|  |
| --- |
| **myiste**  server {  listem 80;  server\_name IP OR ServerDomain;  location / { try\_files $uri @app; }  location @app {  include uwsgi\_params;  uwsgi\_pass unix:/tmp/mysite.sock;  }  } |

|  |
| --- |
| ln –s /etc/nginx/sites-available/mysite /etc/nginx/sites-enabled/mysite |

많은 예제들이 location = / { ~~~ } 으로 설정을 하는데 이렇게 할 경우 @app.route(‘/’)에 대해서만 작동하니 유의해서 적용해야 한다.

이제 적용을 위해 아래의 명령어를 입력하면 파이썬 Flask의 사용 준비가 끝났다. 그 이후 서버의 아이피 주소를 입력해서 ‘Hello World’가 나오는지 확인하면 된다.

|  |
| --- |
| service nginx restart |

## 참고 사이트

<http://bablabs.tistory.com/25> - Flask AWS에 프로덕션용으로 Deploy하기(nginx + uwsgi)

<http://markjberger.com/flask-with-virtualenv-uwsgi-nginx/> - Running flask with virtualenv, uwsgi, and nginx

# Anaconda – 파이썬 배포판

Download Anaconda <https://www.continuum.io/downloads>

Python 3.6 version의 64-BIT INSTALLER 를 선택하여 설치한다.

설치 시 이전 Python이 설치되어 있다면 경로를 잡지 않는다. 경로를 잡지 않더라도 Anaconda Prompt 를 통해 사용할 수 있다.

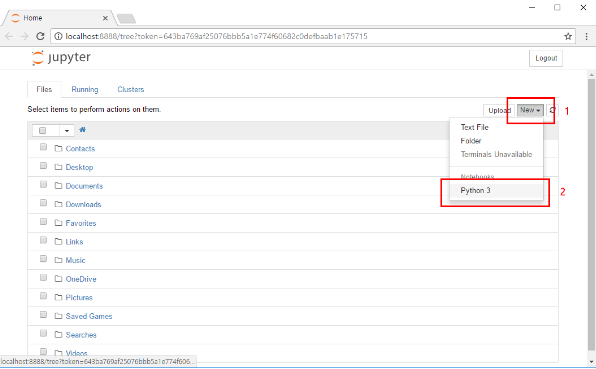
# Jupyter Notebook

Jupyter Nodebook은 웹 브라우저에서 파이썬 코드를 작성하고 실행까지 해볼 수 있는 툴이다. Anaconda를 설치하면 Jupyter Notebook이 설치된다.

## Jupyter Notebook 실행 및 사용하기

|  |
| --- |
| C:\>C:\Users/Anaconda3\python.exe –m nodebook |

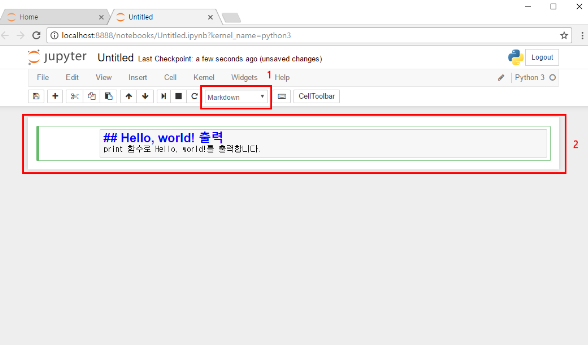
명령을 실행하면 웹 브라우저에 주피터 노트북이 표시된다. 이 화면에서 파이썬 노트북을 만들어본다. 오른쪽 New 버튼을 클릭한 뒤 Python 3을 클릭한다.



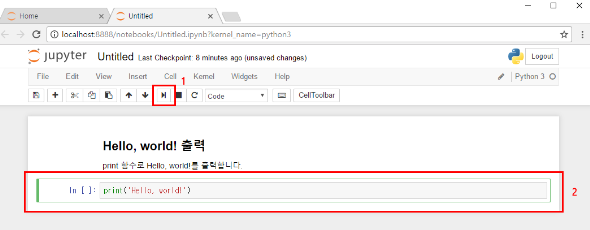
이제 새 노트북 화면이 나온다. 주피터 노트북은 노트북이라는 말 그대로 공책을 사용하듯이 코드를 작성하면서 설명도 함께 넣을 수 있다.

그럼 코드를 작성하기 전에 먼저 설명부터 넣는다. 메뉴의 드롭다운 목록에서 Markdown을 선택하고 아래 빈칸에 다음 내용을 입력한다. #은 제목이라는 뜻이며 #이 하나씩 늘어날수록 하위 제목이 된다.

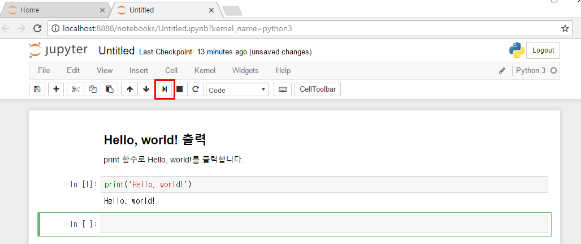
|  |
| --- |
| ## Hello, world! 출력  Print 함수로 Hello, world!를 출력합니다. |



이제 설명을 적용한 뒤 파이썬 코드를 입력해본다. 메뉴에서 ▶| 버튼을 클릭하면 설명이 적용되고 아래에 셀(Cell)이 생긴다. In[]: 오른쪽에 print (‘Hello, world!’)를 입력한다.



파이썬 코드를 입력했으면 실행을 해본다. 메뉴에서 다시 ▶| 버튼을 클릭하면 코드가 실행되고 결과가 출력된다. 이때 In[ ]: 이 In [1]: 로 바뀌는데 1은 첫 번째로 실행된 코드라는 뜻이다. 노트북에서는 코드 입력 셀이 추가될수록 숫자가 높아진다.



이런 방식으로 셀을 추가하면서 코드와 설명을 작성한다. 주피터 노트북은 코드, 코드의 실행 결과, 코드에 대한 설명을 한 번에 작성할 수 있어서 체계적인 기록이 가능하다.

특히 한 노트북 안에 들어있는 코드 셀은 모두 실행 상태가 연결된다. 즉, 코드셀에서 변수를 만들면 다른 코드 셀에서 해당 변수를 사용할 수 있다. 이때 코드 셀의 실행 순서는 In [ ]: 에 표시된 숫자 순서를 따르며 메뉴의 ↑↓ 버튼을 이용하여 코드 셀의 위치를 바꿀 수 있다. (코드 셀의 위치를 바꾸더라도 실행순서는 In [ ]: 의 숫자를 따름)

참고로 노트북의 제목을 바꾸려면 맨 위 Jupyter로고 옆의 Untitled를 클릭하면 된다. 그리고 메뉴의 File > Save and Checkpoint를 클릭하면 노트북이 파일로 저장된다. 이번 실습에서는 노트북의 제목을 바꾸지 않았으므로 C:\Users\<사용자계정> 폴더에 Untitled.ipynb 파일로 저장된다. (노트북이름.ipynb 형식)

|  |
| --- |
| 참고 | 노트북 파일 저장 경로 바꾸기  주피터 노트북은 기본적으로 C:\Users\<사용자계정> 폴더에 노트북 파일을 저장한다. 노트북 파일 저장 경로를 바꾸려면 –notebook-dir 옵션을 폴더를 지정해 주면 된다.  C:\User\JMS>C:\User\JMS\Anaconda3\python.exe –m notebook –notebook-dir C:\project  시작 > Anaconda3 (64-bit) > Jupyter Notebook의 노트북 파일 저장 경로를 바꾸는 방법은 다음과 같다.  Jupyter Notebook 메뉴에서 마우스 오른쪽 버튼을 클릭하고 자세히 > 파일 위치 열기를 클릭.  탐색기 창에서 Jupyter Notebook 바로가기를 선택한 뒤 마우스 오른쪽 버튼을 클릭하고 속성® 클릭.  속성 창의 대상(T)에서 %USERPROFILE%을 지우고 C:\project 입력 |

**참고 | 아나콘다를 설치하지 않고 주피터 노트북을 사용하려면?**

|  |
| --- |
| 아나콘다를 설치하지 않고 주피터 노트북을 사용하려면 pip로 notebook 패키지를 설치한 뒤 jupyter notebook으로 실행하면 된다. 단, Windows에서는 패키지 설치 중에 에러가 나는 경우가 많다. 이때는 ‘부록 48.10 Windows에서 패키지 설치 에러 해결하기’를 참조한다. |

|  |
| --- |
| C:\Users\JMS>pip install notebook  C:\Users\JMS>jupyter notebook |

리눅스나 macOS에서는 notebook 패키지를 설치한 뒤 jupyter notebook으로 실행한다.

[리눅스, macOS]

|  |
| --- |
| $ sudo pip install notebook  $ jupyter notebook |

노트북 파일 저장 경로는 jupyter notebook –notebook-dir 디렉토리 형식으로 지정하면 된다.

## 웹 페이지의 HTML을 가져와서 파일로 저장하기[Sample]

참조 사이트 : <https://dojang.io/mod/page/view.php?id=1158>

기상청 웹 사이트에서 도시별 현재날씨 페이지의 HTML을 가져와본다. 웹 브라우저를 실행하고 다음 주소로 이동한다.

* 도시별 현재날씨 > 지상관측자료 > 관측자료 > 날씨 > 기상청

http://www.kma.go.kr/weather/observation/currentweather.jsp

# PIP와 Virtualenv

https://medium.com/@dan\_kim/파이썬-초심자를-위한-pip-그리고-virtualenv-소개-a53512fab3c2

## PIP

Pip는 Python Package Index(PyPI) 저장소로부터 파이썬 패키지를 받아 설치하는 패키지 관리 도구이다. PyPI(가끔 The Cheeseshop 이라고도 불리는)는 third-party 파이썬 오픈소스 패키지들을 위한 저장소이다. Ruby에서의 RubyGems 혹은 PHP의 Packagist, Perl의 CPAN 그리고 Node.js의 NPM과 비슷하다고 생각하면 된다.

사실 Python은 훨씬 더 기초적인 패키지 매니저인 easy\_install을 가지고 있다. 여러분이 처음 Python을 설치할 때 easy\_install은 자동으로 설치된다. 하지만, 여러 측면에서 easy\_install보다 훨씬 더 우월한 pip를 사용하는 것이 일반적이다. 먼저, easy\_install을 통해 pip를 다음과 같이 설치할 수 있다.

|  |
| --- |
| $ sudo easy\_install pip |

이제 pip를 통해 패키지들을 설치할 수 있다. (아래 예제에서는 Django를 설치해본다)

|  |
| --- |
| $ sudo pip install Django |

여기서 우리는 Django를 시스템 전체에 (global) 설치하였다. 하지만, 대부분의 경우 패키지를 글로벌하게 설치하지 않아야 한다.

## virtualenv

**Virtualenv**는 아주 구체적인 문제를 해결한다. 보통 여러 개의 파이썬 프로젝트가 하나의 컴퓨터에서 충동을 일으키지 않고 존재할 수 있도록 도와준다.

### 어떤 문제를 해결하나?

virtualenv가 해결하는 문제를 설명하기 위해, 먼저 virtualenv가 존재하지 않는다고 가정해보자. 여러분은 외부 웹 서버에 HTTP 요청을 보내는 파이썬 프로그램을 작성해야 하는 상황이다. 이 일에 뛰어난 Requests 라이브러리를 사용할 예정이다. 위에서 보여준것처럼 pip를 이용해 Requests 라이브러리를 설치해본다.

그런데, pip는 여러분이 컴퓨터 어느곳에 패키지를 설치할까? 여러분이 pip install requests 명령어를 실행시키면 다음과 같은 일이 발생한다.

|  |
| --- |
| $ pip install requests  Downloading/unpacking requests  Downloading requests-1.0.0.tar.gz (337kb) : 337kb downloaded  Running setup.py egg\_info for package requests  Installing collected packages: requests  Running setup.py install for requests  Error: coluld not create ‘/Library/Python/2.7/site-package/requests’: Permission denied |

pip가 /Library/Python/2.7/site-packages/requests 안에 패키지를 설치하려고 했던 것으로 보인다. 이 폴더는 파이썬이 알고 있는 특별한 폴더이다. Site-packages 내부에 패키지가 설치되면, 파이썬 프로그램에서 이 패키지를 임포트해서 사용할 수 있게 된다.

하지만, 위에서는 에러가 발생했다. 일반적으로 맥에서 “일반 사용자”는 /Library 폴더에 쓰기 권한이 없기 때문에 위와 같은 에러가 발생한 것이다. 이 에러를 고치기 위해서는 sudo pip install requests 명령어를 실행하면 된다. 여기에서 sudo는 “슈퍼 유저”로 명령어를 실행하라는 의미이다.

|  |
| --- |
| $ sudo pip install requests  Password:  Downloading/unpacking requests  Running setup.py egg\_info for package requests  Installing collected packages: requests  Running setup.py install for requests  Successfully installed requests  Cleaning up … |

이제 정상적으로 동작한다. Python 명령어를 실행해서 Requests 라이브러리를 임포트해보자.

|  |
| --- |
| >>> import requests  >>> requests.get(‘http://dabapps.com’)  <Response [200]> |

지금까지는 우리는 pip를 통해 라이브러리를 설치하고, 이 패키지를 import requests 구문을 통해 파이썬 프로그램 내에서 사용해 보았다. 시간이 지나서, PyPI에서 다른 라이브러리들을 가져와서 우리의 앱을 계속 발전시켰다고 가정해보다. 시간이지나 우리가 발전시킨 앱은 뛰어난 성능을 보여주면서 많은 돈을 벌 수 있었고, 사용자들은 약간의 차이가 있는 새로운 프로그램을 작성해달라는 요청을 보내고 있다.

사용자들의 요청에 보답하여 새로운 앱을 만들어보고자 새로운 프로젝트를 시작했다. 이번 앱에도 requests 라이브러리가 필요한데, 첫번째 앱을 만들었던 이후로 requests 라이브러리에는 새로운 기능이 추가된 것을 확인했다. 이 추가된 기능이 새로운 앱에서 필수적인 상황이다. 새로운 기능을 사용하기 위해서 다음과 같은 pip 명령어로 requests 라이브러리를 업데이트했다고 가정하자.

|  |
| --- |
| sudo pip install –upgrade requests |

모든 것이 좋아보이지만, 우리가 알지 못하는 사이에 재앙은 발생하고 있다. 업데이트 이후, 우리에게 돈을 많이 벌어다주던 기존의 프로그램을 돌려보자. 프로그램이 에러가 발생한다. 왜일까? 바로 requests 라이브러리가 변경되었기 때문이다. 아주 작은 변화였지만, 우리의 코드가 더 이상 라이브러리를 못쓰는 상황이 벌어진 것이다. 모든 것이 망가져 버렸다.

물론, 새로운 requests API를 사용하도록 기존의 프로그램을 변경해 문제를 해결할 수 있다. 하지만, 시간이 소요되고 새로운 프로젝트에서 집중도를 잃게 된다. 보통 숙련된 파이썬 개발자는 단지 2개의 프로젝트만 가지고 있지 않다. 수십가지 프로젝트를 동시에 진행하고, 각 프로젝트는 수십가지의 라이브러리에 대한 의존성을 가지고 있다. 이 라이브러리 전체를 최신버전으로 유지하고 모든 프로젝트에서 동일한 버전의 라이브러리를 사용하도록 만드는 것은 완전 악몽이다.

### 어떻게 virtualenv는 이 문제를 해결하나?

virtualenv는 각 프로그램별로 완전히 독립적인 가상의 환경을 만들어냄으로써 이 문제를 해결한다. 여기서 환경이란 파이썬 프로그램을 실행시키는데 필요한 모든것의 복사본을 가지고 있는 단순한 폴더이다. 전체 파이썬 스탠다드 라이브러리 복사본, pip 설치 프로그램 복사본, 그리고 위에서 언급한 site-packages 복사본 등을 포함한다. 여러분이 virtualenv 도구를 이용해 생성된 pip 복사본을 이용해 PyPI로부터 패키지를 설치하면, **virtualenv 폴더 내부의 site-packages 폴더에 이를 설치**한다. 그리고 설치된 패키지는 이전과 동일한 방법으로 파이썬 프로그램 내부에서 사용할 수 있다.

### 어떻게 vitualenv를 설치하나?

만약 여러분이 이미 pip를 설치하였다면, virtualenv를 설치하는 가장 쉬운 방법은 sudo pip install virtualenv 명령어이다. Pip와 virtualenv는 일반적으로 글로벌 설치가 되어야 하는 유일한 패키지이다. 이 두개를 설치하고 나면 나머지 패키지들은 가상 환경에 설치하면 된다.

사실, virtualenv는 pip의 복사본을 수반하고 있다. 그렇기에 필요한 것은 사실상 virtualenv 뿐이다. Virtualenv는 Pypi에서 설치하는 것이 아닌 독립적인 패키지로 설치가 가능하다. 이런 방법은 윈도우 사용자들에게 더 쉬울 수 있다. Virtualenv.org에 있는 설치 안내를 참고한다.

### 어떻게 새로운 가상 환경을 생성할 수 있나?

새로운 환경을 만들기 위해선느 virtualenv만 있으면 된다. 프로젝트의 루트 폴더로 이동한 이후, 아래와 같이 virtualenv 명령어로 생성하면 된다.

|  |
| --- |
| $ cd ~/myproject  $ virtualenv env  New python executable in /env/bin/python  Installing setuptools … done.  Installing pip … done. |

여기에서, env는 여러분의 가상 환경을 생성할 폴더의 이름이다. 아무런 이름을 붙여도 되지만, 보통 env라고 부르고 프로젝트 디렉토리 내부에 이를 만드는 것이 일반적인 관습이다. 예를 들어, 여러분이 코드를 ~/projectname/ 에 보관한다고 하면, ~/projectname/env가 새롭게 만들어진 환경이라고 생각하면 된다. 하지만, 여러분이 원하는 폴더 이름으로 만들어도 상관없고, 프로젝트 폴더 외부에 위치시켜도 상관은 없다.

|  |
| --- |
| *만약 여러분이 git과 같은 버전 컨트롤 시스템을 사용하고 있다면, env 디렉토리를 커밋에 포함시키지 않는 것을 추천한다. 반드시 .gitignore 파일에 env 디렉토리를 추가한다.* |

### 어떻게 새로운 가상 환경을 사용하나?

새롭게 생성한 env 폴더 내부를 살펴보면, 아래와 같은 몇몇개의 폴더를 확인할 수 있다.

|  |
| --- |
| $ ls env  bin include lib |

여러분이 가장 관심을 기울여야하는 폴더는 bin 이다. 이 폴더는 파이썬 라이브러리의 로컬 복사본과 pip 설치 복사본이 있는 곳이다. 이제 pip 복사본을 이용해서 virtualenv 내부에 requests 라이브러리를 설치해보자.

|  |
| --- |
| $ env/bin/pip install requests  Downloading/ unpacking requests  Downloading requests-1.1.0.tar.gz (337Kb) : 337Kb downloaded  Running setup.py egg\_info for package requests  Installing collected packages : requests  Running setup.py install for requests  Successfully installed requests  Cleaning up … |

여기에서 중요한 것은 sudo를 사용하지 않았다는 점이다. 이제 request를 글로벌로 설치하지 않고 home 폴더내부에 설치하기 때문에, 더 이상 sudo가 필요하지 않은 것이다.

이제, Python shell을 실행하기 위해 python 명령어를 실행하지 않고, **env/bin/python** 명령어를 실행하면 된다.

|  |
| --- |
| >>> import requests  >>> requests.get(‘http://dabapps.com’)  <Response [200]> |

### 편리하기 사용하기

virtualenv는 숨겨진 다양한 트릭들을 가지고 있다. env/bin/python 그리고 env/bin/pip

를 매번 입력하는 대신, 우리가 만든 가상 환경을 실행시킬 수 있다.

source env/bin/activate 명령어를 통해서 활성화시키면 이 스크립트는 여러분의 shell의 몇몇 변수들을 일시적으로 조정합니다. 그래서, python을 타이핑하면 글로벌 python 대신 virtualenv 내부에 있는 python 실행파일을 얻을 수 있다.

|  |
| --- |
| $ which python  /usr/bin/python  $ source env/bin/activate  $ which python  /Users/JMS/myproject/env/bin/python |

이제 여러분이 env/bin/pip install requests 를 입력하는 대신에 pip install requests 라고 입력하면 글로벌이 아닌 우리가 가상환경에 라이브러리를 설치하게 된다. 스크립트가 터미널에 준 이 변화는 터미널이 열려있는 동안만 유지가 된다. 때문에 여러분이 새로운 터미널을 실행시키면 다시 source env/bin/activate 스크립트를 실행시켜야 한다. 다른 프로젝트로 이동하고 싶다면 deactivate 명령어를 통해 현재 가상환경의 사용을 종료시킬 수 있다. 이후, 해당 프로젝트를 이동해 source /env/bin/activate 명령어를 실행시키면 된다.

Activating과 Deactivating은 여러분이 타이핑을 하는 것을 감소시켜 준다. 하지만, 이것을 사용하면 조금은 혼란스러울 수 있다. 이를 사용할지 말지는 여러분에 달려 있다.

### Requirements 파일

pip의 requirements 기능을 사용할 때 virtualenv와 pip는 서로 아주 좋은 동료가 된다. 여러분의 각 작업 프로젝트는 각기 자기들만의 requirements.txt 파일을 가지고 있다. 이 파일을 가상환경에서 필요한 라이브러리들을 설치하는데 사용할 수 있다.

|  |
| --- |
| env/bin/pip install –r requirements.txt |

더 자세한 정보는 pip 문서를 참고한다.

## 정리

* pip는 Python Package Index에서 패키지를 설치하는 도구이다.
* Virtualenv는 python, pip, PyPI 부터 설치된 라이브러리들의 복사본을 만듦으로써, 독립적인 파이썬 환경을 만들어 주는 도구이다.
* 이는 여러분이 동시에 하나의 기기에서 여러 개의 프로젝트가 가지는 다른 의존성을 다룰 수 있도록 디자인되었다.
* 설치 관련 가이드는 virtualeng.org 에서 찾을 수 있다.
* 설치 후에, env라는 가상환경 폴더를 만들고 싶다면 virtualenv env 명령어를 실행시키면 된다.
* 여러분의 각 프로젝트마다 이러한 환경이 하나씩 필요하게 된다. 이 폴더들을 버전 컨트롤 시스템에서 제외하는 것을 잊지 말자.
* 가상 환경에서 python과 pip를 사용하고 싶다면, env/bin/python 그리고 env/bin/pip를 사용하면 된다.
* 가상 환경을 soruce env/bin/activate 스크립트로 활성화 시킬 수 있고 deactivate 명령어를 통해 비활성화 시킬 수 있다. 선택적이지만 개발을 조금 더 편하게 도와 줄 것이다.

## virtualenv installation

python의 가상환경이란, 작은 python을 새로 설치해서 내가 원하는 모듈만 운용하는 바구니라고 생각하면 된다. 운영체제 안에서 새로 운영체제를 만들어내는 가상 머신 (virtual machine)과 같은 맥락이라고 볼 수 있다. 같은 모듈이라도 이 버전 저 버전 다른 버전이 필요할 때나, python 프로그램을 실행하기 위한 최소한의 환경을 마련하고자 할 때, 그리고 github 등의 저장소와 연계하고자 할 때 등 가상환경은 매우 다양하게 사용될 수 있다. 이젠 핋수적인 요수가 된 python 가상 환경의 리눅스/윈도우에서의 사용법을 정리한다.

### Ubuntu Environment

https://dgkim5360.tistory.com/entry/python-virtualenv-on-linux-ubuntu-and-windows

Ubuntu의 현재 최신 LTS 버전인 Ubuntu 16.04 LTS는 운영체제에 기본적으로 python 2.7 버전과 3.5 버전이 모두 탑재되어 있다. 그리고 각 버전의 실행은 다음 명령어를 통해 할 수 있다.

|  |
| --- |
| $ python  Python 2.7.12 (default, Jul 1 2016, 15:12:24)  [GCC 5.4.0 20160609] on linux2  Type “help”, “copyright”, “credits” or “license” for more information.  >>>  $python3  Python 3.5.2 (default, Sep 10 2016, 08:21:44)  [GCC 5.4.0 20160609] on linux  Type “help”, “copyright”, “credits” or “license” for more information.  >>> |

그리고 python과 함께 딸려오는 것이 있는데 pypi(python package installer)이다. Pip install .. 로 익숙한 그것이다. 마찬가지로 python 버전에 따라 다음과 같이 실행되고 –version 옵션을 통해 그 버전을 확인해 볼 수 있다.

|  |
| --- |
| $ pip –version  pip 8.1.2 from /home/user/.local/lib/python2.7/site-packages (python 2.7)  $ pip3 –version  pip 8.1.2 from /home/user/.local/lib/python3.5/site-packages (python 3.5) |

그럼 이제 가상환경을 사용하기 위한 가상환경 모듈 virtualenv 를 설치하자. 아마도 sudo와 함께 써야 할 것이다.

|  |
| --- |
| # python 2.7  $ pip install virtualenv  # python 3.5  $ pip3 install virtualenv |

이제 virtualenv가 설치되었고, 바로 쉘에서 다음과 같이 가상환경을 시작할 수 있다.

|  |
| --- |
| $ virtualenv venv  Running virtualenv with interpreter /usr/bin/python  New python executable in /home/don/venv/bin/python  Installing setuptools, pip, wheel .. done. |

위 작업이 완료되면 내가 지정했던 venv 라는 폴더가 생성되고 그 안에 새로 python이 설치되어있음을 알 수 있따. 이제 venv를 가동시키고, 정말로 새로운 python 환경이 생겼는지 확인해보자.

|  |
| --- |
| $ source venv/bin/activate  (venv) $ pip list  pip (8.1.2)  setuptools (28.8.1)  wheel (0.30.0a0) |

여기서 두가지를 확인할 수 있다. 첫 번째로, source 명령어를 통해 venv 가상환경의 activate를 실행하니 그 다음부터는 쉘 명령창 앞부분에 (venv)가 따라 붙어다니는 것을 볼 수 있다. 이는 이제부터 venv 가상환경 안에 있는 거라고 말해주는 것이다. 두번째로, venv안에서 설치된 모듈을 pip list 명령을 통해 확인한 결과… 아무것도 없다는 것이다. Pip, setuptools, wheel은 가상환경을 시작할 때 기본으로 설치된 것이다. (이미 pip list를 쓰고 있지 않은가)

한 가지 궁금증이 생길 법한 것이 있는데, 다시 한번 버전 문제이다. 2버전, 3버전의 python 가상환경을 쓰려면 어떻게 해야하나? 다음과 같은 방안을 쭉 나열해 본다.

|  |
| --- |
| # python 2  $ python –m virtualenv venv  $ virtualenv venv –python=python  $ virtualenv venv –python=python2.7  # python 3  $ python3 –m virtualenv venv  $ virtualenv venv --python=python3  $ virtualenv venv --python=python3.5 |

이제 내가 사용하고자 하는 모듈을 맘대로 설치하면 된다. 그 설치한 목록은 다음 명령어를 통해 정확한 리스트로 저장해두자.

|  |
| --- |
| (venv) $ pip freeze > requirements.txt |

이제 requirements.txt 파일에서 설치된 모듈과 그 버전이 리스트되어 저장되어 있음을 알 수 있다. 가상환경을 나가고 싶으면 간단히 deactivate라고 명령하면 된다.

### Window Environment

사실 윈도우라고 다를 것은 없다. 기본적으로 python을 설치하면, 자동적으로 pypi가 함께 설치되기 때문에 위 리눅스일 때와 동일하게 virtualenv를 설치하고 시작할 수 있다.

|  |
| --- |
| > pip install virtualenv  > virtualenv venv |

이제 activate해야 되는데, 여기서는 soruce 대신 call을 사용한다.

|  |
| --- |
| > call venv/scripts/activate  (venv) > pip list |

# Git

|  |
| --- |
| git config --global user.name "이름"  git config --global user.email "깃허브 메일주소" // 매번 물어보는 귀찮음을 피하기 위해 설정.  mkdir ~/MyProject // 로컬 디렉토리 만들고  cd ~/myproject // 디렉토리로 들어가서  git init // 깃 명령어를 사용할 수 있는 디렉토리로 만든다.  git status // 현재 상태를 훑어보고  git add 화일명.확장자 // 깃 주목 리스트에 화일을 추가하고 or  git add . // 이 명령은 현재 디렉토리의 모든 화일을 추가할 수 있다.  git commit -m “현재형으로 설명” // 커밋해서 스냅샷을 찍는다.  git remote add origin https://github.com/username/myproject.git // 로컬과 원격 저장소를 연결한다.  git remote -v // 연결상태를 확인한다.  git push origin master // 깃허브로 푸시한다. |