파이썬 가상환경

▼ 파이썬 가상환경이 필요한 이유와 사용법 (venv, virtualenv)

https://windybay.net/post/13/

> 프로젝트와 그 프로젝트에 사용된 패키지들은 언제나 한 묶음으로 움직이는 것이 좋겠다.

▼ 가상환경이 필요한 이유

▼ 1. 이 프로젝트에 필요한 패키지 파악 및 환경설정

파이썬에서 프로젝트를 진행할 때는 각각의 프로젝트 별로 가상환경을 만들어서 진행해 주는 것이 좋다. 사실 프로젝트라 할만한 것을 제대로 진행해 보기 전, 책을 보면서 코어 파이썬 문법을 배우고 예제만 따라해 보는 수준에서는 왜 굳이 귀찮게 가상환경을 설치하는것인지 잘 이해가 되지 않았다.

처음 파이썬을 배울때 아나콘다(Anaconda) 배포판을 이용해서 파이선을 설치했는데, 아나콘다를 설치하면 수백가지의 패키지가 같이 설치된다. 이후로도 이것저것 배우면서 설치한 패키지들이 많이 있었는데, 이렇게 한번 설치해 놓으면 패키지를 아무거나 필요한 대로 불러서 쓸 수 있는데 굳이 왜 가상환경을 만들어서 이미 설치했던 패키지를 또 설치하라는건지 이해가 되지 않았다.

그런데 Django를 배우면서 직접 프로젝트들을 한두 개 진행하다 보니 왜 가상환경이 필요한지 저절로 알게 되었다.

1. 프로젝트를 배포하면 원격 서버에 따로 패키지들을 설치해 줘야 하는데, 내가 이 프로젝트만을 위해서 설치한 패키지들이 어떤 것들이 있는지 알 수가 없었다. 내가 설치한 아나콘다 배포판에는 기본적으로 수많은 패키지가 포함되어 있고 그 환경에 내가 추가로 설치한 패키지들이 뒤섞여 있는 상태이다.

한편 배포하려는 서버에는 처음에는 아무것도 깔려 있지 않기 때문에 필요한 것들을 설치해 주어야 하는 상황이다. 그런데 유료로 사용하는 서버에 공간만 차지하고 쓰지도 않을 패키지들을 설치할 필요는 없으니 딱 내가 사용한 패키지들만 설치를 해야 하는데 그걸 구분하는게 문제였다.

물론 어떤 패키지를 설치할 때마다 따로 적어 놓는다든지 할 수도 있겠지만, 프로그래밍을 배운다는 입장에서 그런 원시적인 짓을 할수는 없지 않나. 그리고 pip 명령을 이용해서 어떤 패키지를 설치하면 딱 그 패키지만 설치되는게 아니

고 딸려 있는 여러가지 이름이 다른 패키지들도 자동으로 설치가 되는 경우가 많기 때문에, 그런것들까지 다 확인한다는것은 사실상 불가능하다.

그리고 서버에 일일이 그 패키지들을 설치해주는 것이 혹시 가능하다 하더라도 정확히 내가 사용했던 버전과 버전이 맞는지 일일이 확인해야 할텐데, 그걸 수작업으로 한다는것도 역시 원시적이고 말이 안된다. 그래서 미리 가상환경을 만들어 내가 실제로 이 프로젝트에 사용한 패키지들만 설치를 해야 할 필요성을 느꼈다.

▼ 2. 의존적인 패키지 관리

2. 그리고 시간이 지남에 따라 패키지들이 업데이트되는데, 이것저것 업데이트를 하다 보면 서로 의존적인 패키지들 사이에 버전이 맞지 않아 호환이 되지 않는 경우들이 생긴다. 그렇다고 특정한 프로젝트 하나를 위해 언제 지원이 끊길지 모르는 예전 버전의 패키지를 로컬에 계속 유지할 수도 없고, 패키지가 업데이트되어 호환성 문제가 생길 때마다 프로젝트의 코드를 일일이 수정하는 것도 실질적으로 불가능하다.

아예 마음먹고 프로젝트를 메이저 버전업을 하면서 갈아 엎어버리는 경우라면 모를까, 구 버전에서 잘 작동하는 프로젝트의 코드를 개별 패키지들의 마이너 버전업 때마다 체크하고 수정하는 것도 불가능하고, 가능하더라도 시간낭비인 경우가 많을것이다. 그래서 한 프로젝트를 위해 확실히 작동하는 버전의 여러 패키지들을 한데 모아서 관리하기 위해서도 가상환경이 필요하다.

▼ 3. 파이썬3 사용

3. 파이썬 버전 자체가 다른 환경인 경우도 있겠다. 사실 나는 처음부터 파이썬 3로 입문해서 로컬에서 작업할 때는 파이썬 3만 쓰기 때문에 문제가 없지만, 프로젝트를 배포하려는 서버에는 파이썬 2만 설치된 경우도 있고, 2와 3이 같이 있는 경우도 있는 등 다양한 환경이다. 그런데 예를 들어 최신버전의 Django 같은 경우에는 파이썬 2에서는 아예 동작을 하지 않기 때문에 명시해서 파이썬 3를 설치해 주고 파이썬 3에 맞는 가상환경을 설정해 주지 않으면 프로젝트 구동 자체가 불가능하다.

▼ 파이썬 가상환경 사용해보기

https://wikidocs.net/70588

▼ 1. 가상환경 만들기

윈도우에서 명령 프롬프트(cmd)를 실행하고 다음 명령어를 입력해 C:/venvs 라는 디렉터리를 만들자.(루트 디렉터리를 반드시 C:/venvs로 해야 하는 것은 아니지만 실습 편의를 위해 이대로 지정하자.)

```
C:\Users\pahkey> cd \
C:\> mkdir venvs
C:\> cd venvs
```

파이썬 가상 환경을 만드는 다음 명령어를 입력해 실행하자.

```
C:\venvs> python -m venv mysite
```

명령에서 python -m venv는 파이썬 모듈 중 venv라는 모듈을 사용한다는 의미다. 그 뒤의 mysite는 여러분이 생성할 가상 환경의 이름이다. 가상 환경의 이름을 반드시 mysite로 지을 필요는 없다. 만약 가상 환경의 이름을 awesomesite와 같이 지정했다면 책에 등장하는 mysite라는 가상 환경 이름을 awesomesite로 대체하여 읽으면 된다. (하지만 실습 진행의 편의를 위해 가상 환경 이름을 동일하게 하기를 권장한다.)

명령을 잘 수행했다면 C:/venvs 디렉터리 아래에 mysite라는 디렉터리가 생성되었을 것이다. 이 디렉터리를 가상 환경이라 생각하면 된다. 그런데 가상 환경을 만들었다 해서 바로 가상 환경을 사용할 수는 없다. 가상 환경을 사용하려면 가상 환경에 진입해야 한다.

▼ 2. 가상환경 진입하기

가상 환경에 진입하려면 우리가 생성한 mysite 가상 환경에 있는 Scripts 디렉터리의 activate 명령을 수행해야 한다. 다음 명령을 입력하여 mysite 가상 환경에 진입해 보자.

```
C:\venvs>cd C:\venvs\mysite\Scripts
C:\venvs\mysite\Scripts> activate
(mysite) C:\venvs\mysite\Scripts>
```

그러면 C:/ 왼쪽에 (mysite)라는 프롬프트를 확인할 수 있다. 이름에서 볼 수 있듯 현재 여러분이 진입한 가상 환경을 의미한다.

현재 진입한 가상 환경에서 벗어나려면 deactivate라는 명령을 실행하면 된다. 이 명령은 어느 위치에서 실행해도 상관없다.

```
(mysite) C:\venvs\mysite\Scripts> deactivate
c:\venvs\mysite\Scripts>
```

가상 환경에서 벗어났다면 C:/ 왼쪽에 있던 (mysite)라는 프롬프트가 사라졌을 것이다. 지금까지 가상 환경의 개념과 실습을 진행해 보았다. 가상 환경이라는 개념이 조금은 생소하겠지만 익혀 두면 여러분의 웹 프로그래밍 경험에 도움이 될 것이다.

▼ 가상환경에서의 패키지 관리; requirments.txt

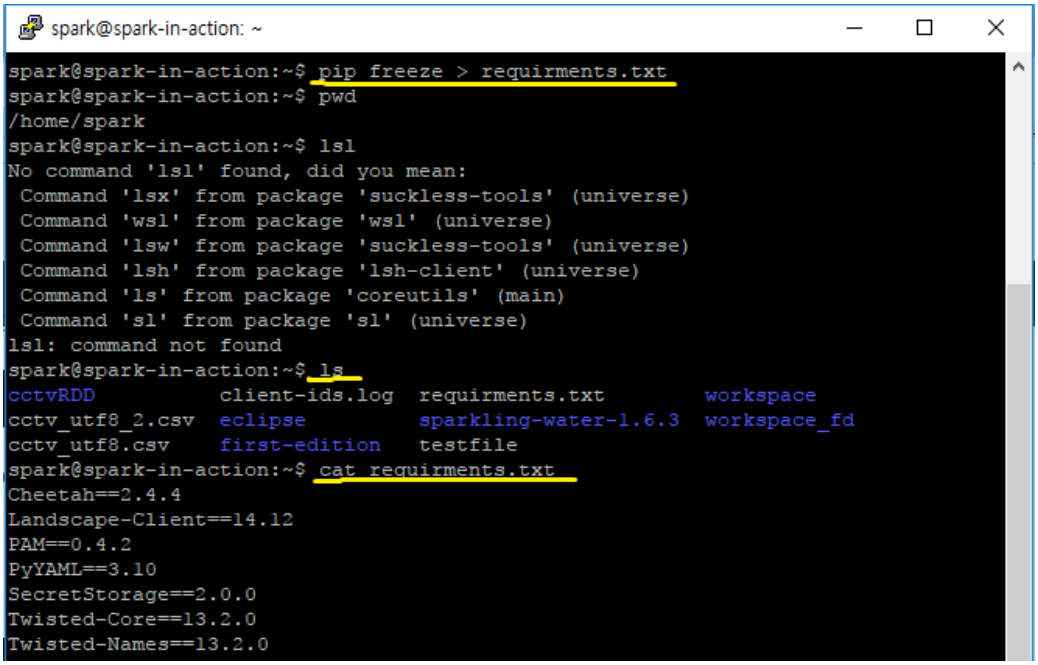
우선 로컬에서 가상환경을 활성화시키고 필요한 모든 패키지들이 설치되었으면, 어느 환경에서든 같은 패키지들이 한 묶음으로 설치되도록 requirements.txt 를 만들어 주는 것이 좋다.

일일이 손으로 적는게 아니라

```
$ pip freeze > requirements.txt
```

를 입력하면 현재 환경에서 설치된 모든 패키지들의 이름과 버전이 명시된 requirements.txt 라는 파일이 자동으로 만들어진다.

▼ ex-capture



그리고 원격 서버에서 이 파일을 이용해 패키지들을 일괄 설치하려면

```
$ pip install -r requirements.txt
```

여기서 -r 은 requirement 를 설치할때는 붙이라고 되어 있는데 공식 documentation 을 보아도 아주 속시원한 설명은 없는 듯하다. 각종 옵션에 이런것들이 많은데 다소 심오한 파이썬 언어의 코어에 가까운 이야기들이라 일단은 그냥 시키는대로 하자..

위의 명령어로 필요한 패키지들을 일괄 설치해 주면 로컬과 원격지에서 똑같은 환경으로 작업이 가능하다. 다만 requirement.txt 를 이용한 설치 동작은 서로 의존적인 패키지들간에 간섭이 있어 잘 설치되지 않거나 에러가 나는 경우도 종종 있는 것 같다. 이런 경우에는 프로젝트 안의 패키지들을 pip list 로 다시 확인해서 잘 설치되지 않은 패키지들은 번거롭지만 수작업으로 설치해줘야 할 때도 있다.

사실 처음에는 좀 번거롭다고 생각되기도 했는데, 익숙해지니 가상환경을 만들지 않고 작업하는 것은 생각하기 어렵게 되었다. 깔끔한 작업 환경을 위해서 항상 가상환경을 먼저 구성하도록 하자.