virtualenv

Before jumping completely into Flask, let's take some time to learn about **virtualenv**

Why would I use this?

Let's imagine that you were working for a company, coding in Python, and juggling a handful of different projects. Let's say we had

- Project 1 - Using Python 2.6.3, Flask 0.3
- Project 2 - Using Python 3.0, Flask 0.10
- Project 3 - Using Python 3.4.3, No Flask

Considering our system, we probably only have one version of Python that will run in the terminal and all of our packages that are installed globally on it. How would the computer know which version of Python to use and which versions of the different modules to import?

Virtualenv is a Python module that allows us to create a Python environment that is segregated from our system-wide Python installation. Therefore, we can install specific versions of Python and the required dependencies unique to each project. It also protects us from upgrading and changing dependencies that will be used in some projects but not on others. This is a great tool!

Installing virtualenv

To install virtualenv globally;

```
$ pip install virtualenv
FOR PC's DEPENDING ON YOUR SETUP:
$ python -m pip install virtualenv
```

Creating a new virtual environment

To get started let's navigate to our desktop, create a new folder and navigate into it:

```
$ cd ~/Desktop
$ mkdir my_new_project
$ cd my_new_project
```

Now we will create a new virtual environment local to this folder. Below we will use the name venv but you can pass any name as ar argument:

```
$ virtualenv venv

FOR PCs:
$ python -m virtualenv venv
```

You should see a new folder created in the my_new_project directory called *venv* which has the following structure. Note the directory created will reflect the argument passed to the virtualenv command

```
venv/
  | # the lib and include directories contain supporting lib files for a new
  |- bin/  # where executable files live -> like python
  |- lib/  # the modules we install with pip will be installed in here
  |- include/<br>
```

For now, you will follow this process for each new assignment/project

Activating

After having your virtualenv created, activate it by typing:

```
$ source venv/bin/activate # for OSX
$ source venv/Scripts/activate # for Windows

# depending which terminal you are using, i.e. CMD prompt or Git CMD, the '/' may need to be '\'
# or you may need to drop the word 'source' if you are using Git CMD
```

Your terminal/command prompt should change and look something like this:

```
(venv) $
```

The **(venv)** indicates that your virtualenv is now active. If we install any dependencies they will install into our local venv folder:

```
(venv) $ pip install flask
```

When the virtual environment is active, we will use local Python and all the pip modules rather than systemwide!

Deactivating

When you're finished working on a project or want to work on a different one, we can deactivate simply by typing deactivate into our prompt and our prompt will return to normal:

```
(venv) $ deactivate
$
```

To work another project, we activate another virtual environment. **Make sure the virtual environment for each specific project is active when running your project or you may have some unexpected errors.**