Python Installation & Django Workflow

Installing Python

Installing Python on Windows is easy. Just find the Windows Executable Installer on the [Python downloads page](https://www.python.org/downloads/release/python-352/), run it, and you're good to go. When the Installation Wizard pops up, be sure to check the Add python to PATH option, and at the end of install allow python to bypass PATH limit. This will do any manual work involved in adding Python to your PATH automatically.

Mac users should install [Homebrew](http://brew.sh/), and then run brew install python3.

If you're on Linux, you're probably already running Python 3.4.x+. Users of Ubuntu 12.10+ already have Python 3.4.x: Just use python3 and pip3 instead of python and pip. Users of other distributions presumably know what they're doing.

Installing Virtualenv

You *can* install Django globally. But you shouldn't. It's better to maintain the environments of projects that rely on particular versions of packages separately. In Python-land, we use [virtualenv](http://docs.python-guide.org/en/latest/dev/virtualenvs/) for this, and our package manager is called [pip](https://pypi.python.org/pypi/pip).

Virtualenv effectively allows us to create an isolated environment in which we can use a *specific* Python version and install *specific* versions of the packages we want. What you do elsewhere won't interfere at all with this isolated environment.

We'll be using Django 1.10.x, which requires Python 3.4.x+. Since most of you will have both Python 2.7 *and* 3.4/5 on your system, you need to make sure you're using the right version of pip, since there's one for each.

First, run: pip --version. This will print the version of pip you have, and the version of Python it's using.

*pip --version output.*

If you see Python 3.4.3 or higher to the right, keep using just use the pip command.

If you see Python 2.7, as I do, try again with pip3.

*pip3 --version output.*

If that works, use pip3 instead. If it doesn't, Slack a TA or instructor—it should.

Next, we'll install virtualenv. This is a package you *do* want to install globally, since it's an environment manager.

Run:

Open GitBash as Adminstrator if using windows. Use sudo command on Mac. I.E sudo pip install virtualenv --user.

# Or pip3 install virtualenv --user, if appropriate.

pip install virtualenv --user

That should install virtualenv globally.

**Using Virtualenv**

You'll mostly be doing three things with virtualenv: **Creating** virtual environments; **sourcing** them; and **deactivating** them. It's all quite straightforward.

**Creating a virtualenv**

To create a new virtualenvironment, run:

virtualenv $ENVIRONMENT\_NAME

... Where $ENVIRONMENT\_NAME can be anything. I generally use virtualenv env.

Note that this will use your system's *default* Python as the version for the virtual environment. If typing python --version prints Python 3.4.x or higher, this is fine. If you have to use pip3 or python3 explicitly, though, you may need to tell virtualenv to use Python 3 explicitly.

There are two steps.

1. Figure out where your Python 3 installation lives. Write: which python3. I'll call the output of this command $PYTHON\_3.
2. Run: virtualenv -p $PYTHON\_3 $ENVIRONMENT\_NAME, where $PYTHON\_3 is the path to your Python 3 installation.

*Output of which python3.*

*Creating a virtualenv with Python 3.*

**Sourcing a virtualenv**

To actually *use* your sandbox, you need to "source" the virtualenv. This is straightforward:

For Windows, run: source $ENVIRONMENT/Scripts/activate. For Mac, run source $ENVIRONMENT/bin/activate.

Again, I usually use env as my $ENVIRONMENT\_NAME, so I write:

source env/Scripts/activate

or

source env/Scripts/activate

*Sourcing a virtualenv.*

You'll notice an (env) marker pop up to the left of your command prompt. This simply indicates that you're running in a virtualenv, not your normal system environment.

At this point, you can install packages with pip just like you normally would. The only difference is that, if you created your virtualenv with Python 3, you can just write pip instead of pip3.

For example, to install the version of Django we'll use this week, run:

pip install Django==1.10.1

This installs Django 1.10.1, but *only* in your sandboxed virtual environment. Once you deactivate it, the rest of your system won't know anything about it.

**Deactivating**

To exit your virtualenv sandbox and return to your normal system environment, just use deactivate.

*Deactivating a virtualenv.*

Workflow

Everyone settles on their own workflows, but the following cycle is common in Django.

* Write unit tests for your models.
* Define your models.
* Migrate your database.
* Write functional tests for your user stories.
* Define views corresponding to your models.
* Wire your URLs to respond to the views.
* Write templates to "decorate" your views.