Installing Python and Setting up a Virtual Environment

Python Installation Options

- Mac or Windows users can download open source Python for free from python.org.
- Mac users can also use Homebrew to install Python.
- Linux users can use the package managers like apt (Ubuntu, Debian), dnf (Fedora), or yum (CentOS).
- Anaconda is a free open source data science platform that is available to Windows, Mac and Linux platforms.

If you are new to Python and using Windows or a Mac, you might want to start with python.org as there is help and a community available.

Python Virtual Environment

A Python virtual environment is useful for creating a project that depends on installed packages. Packages give Python access to code functionality beyond the basic language. If you do not use a virtual environment and install a Python package globally, you may run into problems when different projects require different versions of a package. The virtual environment allows you to select the versions that you need for the project you're working on.

With a virtual environment in place you can rely on **Isolation**, **Cleanliness** and **Reproducibility**.

- **Isolation:** Prevents conflicts between dependencies of different projects. Project A might need `libraryX` version 1.0, while Project B needs `libraryX` version 2.0. With virtual environments, each project can have its specific dependencies without clashing.

- **Cleanliness:** Keeps your global Python installation clean and free from project-specific clutter.
- **Reproducibility:** Makes it easier to share your project with others. You can provide a `requirements.txt` file (generated with `pip freeze > requirements.txt`) that lists all project dependencies, allowing others to easily recreate your environment.

Anaconda

The Python Anaconda installation provides a virtual environment. If you are using a python.org or Homebrew install, you will need to set up your own virtual environments. A virtual environment is associated with a directory on your file system.

If you are using Anaconda already, you can continue to use it for this course.

Setting Up Virtual Environments

If you're using Python downloaded directly from `python.org` (or installed via a package manager like `apt` on Linux, `brew` on macOS, or a standard Windows installer, as opposed to Anaconda), the standard and recommended way to create a virtual environment is using the built-in `venv` module.

Here are the instructions:

Creating a Virtual Environment with `venv`

Note: These instructions assume you can use Bash or Zsh commands. If you are using a Mac, you likely use Zsh. If you are using Windows, you can install Git Bash or WSL (Windows Subsystem for Linux) in order to execute the commands.

The `venv` module is included with Python 3.3 and later, so you don't need to install anything extra.

Step 1: Open Terminal or Command Prompt

- **Windows:** Search for "cmd" or "Command Prompt" in the Start Menu and open it.
- **macOS / Linux:** Open your "Terminal" application.

Step 2: Navigate to Your Project Directory (Optional but Recommended)

It's good practice to **create your virtual environment inside your project's main directory**. If you don't have one yet, you can create one:

Bash or Zsh

```
# Create a new directory for your project
mkdir my_python_project
# Change into that directory
cd my_python_project
```

If you already have a project directory, navigate into it:

Bash or Zsh

```
cd path/to/your/existing_project
```

Step 3: Create the Virtual Environment

Once you are in your desired directory, run the following command to create a virtual environment. You can name the environment anything you like, but common conventions are `venv`, `.venv`, or `env`.

Bash or Zsh

```
python -m venv venv
```

- `python`: This invokes your Python interpreter. If you have multiple Python versions installed, you might need to use `python3` or `py` (on Windows, sometimes `py -3.9` for a specific version).
- `-m venv`: This tells Python to run the `venv` module as a script.

- `venv`: This is the name of the directory where your virtual environment will be created. You can replace `venv` with any name you prefer (e.g., `myenv`, `env`, `.venv`).

This command will create a new directory (e.g., `venv`) in your current location. Inside this directory, you'll find a copy of the Python interpreter, `pip`, and other necessary files for an isolated environment.

Step 4: Activate the Virtual Environment

After creation, you need to "activate" the virtual environment. Activating it modifies your terminal's `PATH` variable for that session so that `python`, `pip`, and `jupyter` (if installed in the `venv`) refer to the executables within the virtual environment, not your global Python installation.

For Windows:

Bash
```
.\venv\Scripts\activate
```

For macOS / Linux:

Bash or Zsh
```
source venv/bin/activate
```

Step 5: Verify Activation (Optional)

Once activated, your terminal prompt will usually change to indicate the active virtual environment (e.g., `(venv)` at the beginning of the line).

You can also verify by checking the Python interpreter being used:

On macOS/Linux Bash or Zsh
```
which python
python --version
```

On Windows Bash
```
where python
```

This should show the path to the Python executable within your `venv` directory.

And check the `pip` version:

Bash or Zsh
```
pip --version
```

This should also show that `pip` is associated with your `venv`.

Step 6: Install Packages within the Virtual Environment

Now that your virtual environment is active, you can install any Python packages you need for your project using `pip`. These packages will be installed **only** within this virtual environment and will not affect your global Python installation or other virtual environments.

Bash,Zsh
```
pip install jupyter pandas
```

Step 7: Launch Jupyter Notebook (if installed in venv)

If you installed `jupyter` within your active virtual environment:

Bash or Zsh
```
jupyter notebook
```

This will launch Jupyter Notebook using the Python and packages from your activated `venv`.

Step 8: Deactivate the Virtual Environment

When you're done working on your project or want to switch to another environment, simply type:

Bash or Zsh
```
deactivate
```

Your terminal prompt will return to normal, and `python`, `pip`, etc., will once again refer to your global Python installation.