Princeton University
COS 333: Advanced Programming Techniques
A COS 333 Computing Environment

This document describes how to configure your Mac, Microsoft Windows, or (Ubuntu-based) Linux computer to do the COS 333 assignments. The same configuration is appropriate for building and running many of the lecture example programs.

(All) Suggestions

(All) Use your favorite editor (VS Code, Sublime Text, Emacs, etc.) to compose Python programs.

(All) Don't use the terminal pane within your favorite editor to configure your computer to use Python. Also don't use the terminal pane within your favorite editor to build and run your Python programs. That is, don't use your favorite editor as an IDE (Integrated Development Environment). Using an IDE complicates configuration; **it's impossible for the course instructors to support configuration of the wide variety of IDEs that are available at any given time**.

(Mac) Use the Mac Terminal application to configure Python on your computer, and to build and run Python programs on your computer.

(MS Windows) Use the Command Prompt application to configure Python on your computer, and to build and run Python programs on your computer.

(Linux) Use your terminal application to configure Python on your computer, and to build and run Python programs on your computer.

1. Selecting the Login Shell for Your Computer

*If your computer is a Mac or Linux system, then you will use a **shell**. You should use the **bash** shell.*

*Your **login shell** is the shell that executes automatically when you open a terminal window. On most Linux distributions, the default login shell is bash. On Mac systems that run operating systems older than macOS 10.15 Catalina, the default login shell also is bash. However, on Mac systems that run macOS 10.15 Catalina or newer, the default login shell is **zsh**. You should make sure that your default login shell is bash.*

(Mac and Linux) From a terminal, issue the command `printenv SHELL`. If the output is `/bin/bash`, then your login shell is bash. If the output is not `/bin/bash`, then issue the command `chsh -s /bin/bash`, enter your password when prompted, exit your terminal session, start a new terminal session, and again issue the command `printenv SHELL` to confirm that the output is `/bin/bash`.

2. Installing Python

*You must install **Python 3.12** on your computer.*

(Mac) Don't use the python@3.12 application from Homebrew. Instead browse to https://www.python.org/downloads/. Click on the link for Python 3.12.x (where x is the newest available). In the resulting page click on the *macOS 64-bit univeral2 installer* link. In the Mac Finder, double click on the resulting .pkg file to install. Follow the instructions at the end of the installation process to install certificates. To test, issue the command `python3.12` at a shell prompt in a Terminal window. The Python interpreter/compiler should launch, and identify itself as Python 3.12. To exit, at the >>> prompt call the `quit()` function.

(MS Windows) Browse to https://www.python.org/downloads/. Click on the link for Python 3.12.x (where x is the newest available). In the resulting page click on the *Windows installer (64-bit)* link. In Windows Explorer, double click on the resulting .exe file to install. Near the beginning of the installation, check the appropriate checkbox to indicate that you want the installer to add python.exe to your path. To test, issue the command `python` in a Command Prompt window. The Python interpreter/compiler should launch, and identify itself as Python 3.12. To exit, at the >>> prompt call the `quit()` function.

(Linux) Use your Linux package manager to download and install the python3.12 and python3.12-venv (or similarly named) packages. If those packages are unavailable in your package manager's repository, then download and install them from the *deadsnakes* repository by issuing these commands:

```
$ sudo apt update && sudo apt upgrade -y
$ sudo apt install software-properties-common -y
$ sudo add-apt-repository ppa:deadsnakes/ppa -y
$ sudo apt update
$ sudo apt install python3.12
$ sudo apt install python3.12-venv
```

To test, issue the command `python3.12` at a shell prompt in a terminal window. The Python interpreter/compiler should launch, and identify itself as Python 3.12. To exit, at the >>> prompt call the `quit()` function.

3. Creating a Python Virtual Environment

You must create a Python ***virtual environment****. A Python virtual environment specifies a Python compiler/interpreter, and the modules that it should use. Name your Python virtual environment* ***cos333****.*

(Mac and Linux) Issue these commands in a terminal window:

```
$ # Create a directory to store your virtual environment(s).
$ mkdir ~/.virtualenvs

$ # Using the 3.12 Python compiler/interpreter that you installed
$ # previously, create a virtual environment named cos333.
$ python3.12 -m venv ~/.virtualenvs/cos333

$ # Activate your cos333 virtual environment.
$ source ~/.virtualenvs/cos333/bin/activate

$ # Upgrade pip to the most recent version.
$ python -m pip install --upgrade pip

$ # Install the coverage testing tool
$ # into your cos333 virtual environment.
$ python -m pip install coverage

$ # Install the pylint static code analysis tool
$ # into your cos333 virtual environment.
$ python -m pip install pylint
$ # Install the Flask web application framework
$ # into your cos333 virtual environment.
$ python -m pip install flask

$ # Install the Playwright web application tester
$ # into your cos333 virtual environment.
$ python -m pip install playwright
$ playwright install

$ # Deactivate your cos333 virtual environment.
$ deactivate
```

(MS Windows) Issue these commands in a Command Prompt window:

```
C:\>REM Create a directory to store your virtual environment(s).
C:\>mkdir %HOMEPATH%\.virtualenvs

C:\>REM Using the Python 3.12 compiler/interpreter that you installed
C:\>REM previously, create a virtual environment named cos333.
C:\>python -m venv %HOMEPATH%\.virtualenvs\cos333

C:\>REM Activate your cos333 virtual environment.
C:\>%HOMEPATH%\.virtualenvs\cos333\Scripts\activate.bat

C:\>REM Upgrade pip to the most recent version.
C:\>python -m pip install --upgrade pip

C:\>REM Install the coverage tool
C:\>REM into your cos333 virtual environment.
C:\>python -m pip install coverage

C:\>REM Install the pylint static code analysis tool
C:\>REM into your cos333 virtual environment.
C:\>python -m pip install pylint

C:\>REM Install the Flask web application framework
C:\>REM into your cos333 virtual environment.
C:\>python -m pip install flask

C:\>REM Install the Playwright web application tester
C:\>REM into your cos333 virtual environment.
C:\>python -m pip install playwright
C:\>playwright install

C:\>REM Deactivate your cos333 virtual environment.
C:\>deactivate
```

(Mac and Linux) To test, issue these commands in a terminal window. In all cases the absence of error messages indicates success:

```
$ # Activate your cos333 virtual environment.
$ source ~/.virtualenvs/cos333/bin/activate
$ # Launch python and enter a few statements.
$ python
>>> import coverage
>>> import pylint
>>> import flask
>>> import playwright
>>> quit()

$ # Deactivate your cos333 virtual environment.
$ deactivate
```

(MS Windows) To test, issue these commands in a Command Prompt window. In all cases the absence of error messages indicates success:

```
C:\>REM Activate your cos333 virtual environment.
C:\>%HOMEPATH%\.virtualenvs\cos333\Scripts\activate.bat

C:\>REM Launch python and enter a few of statements.
C:\>python
>>> import coverage
>>> import pylint
>>> import flask
>>> import playwright
>>> quit()

C:\>REM Deactivate your cos333 virtual environment.
C:\>deactivate
```

(Mac and Linux) Then throughout the semester issue these commands in a terminal window:

```
$ # Activate your cos333 virtual environment.
$ source ~/.virtualenvs/cos333/bin/activate
```

(Use Python as desired.)

```
$ # Deactivate your cos333 virtual environment.
$ deactivate
```

(MS Windows) Then throughout the semester issue these commands in a terminal window:

```
C:\>REM Activate your cos333 virtual environment.
C:\>%HOMEPATH%\.virtualenvs\cos333\Scripts\activate.bat
```

(Use Python as desired.)

```
C:\>REM Deactivate your cos333 virtual environment.
C:\>deactivate
```

(All) Concerning Your *Home* Directory

(Mac) Your *home* directory is the one which is your working directory when you first launch a Mac Terminal window. At any time during a Terminal session you can issue the command `cd` to make your home directory your working directory. Your home directory probably is `/Users/`*`yourusername`*.

(MS Windows) Your *home* directory is the one which is your working directory when you first launch a Command Prompt window. At any time during a Command Prompt session you can issue the command `cd %HOMEPATH%` to make your home directory your working directory. Your home directory probably is `\Users\`*`yourusername`*.

(Linux) Your *home* directory is the one which is your working directory when you first launch some terminal window. At any time during a terminal session you can issue the command `cd` to make your home directory your working directory. Your home directory probably is `/home/`*`yourusername`*.

(Mac) Handling Files Whose Names Start with a Period

By default the Mac Finder doesn't display any file name that begins with a period. That might be reasonable for ordinary Mac users, but it's not reasonable for Mac users who are programmers.

You can execute these commands in a Terminal window to cause the Mac Finder to display any file name that begins with a period:

```
defaults write com.apple.finder AppleShowAllFiles YES
killall Finder /System/Library/CoreServices/Finder.app
```

You can execute these commands in a Terminal window to cause the Mac Finder *not* to display any file name that begins with a period:

```
defaults write com.apple.finder AppleShowAllFiles NO
killall Finder /System/Library/CoreServices/Finder.app
```

(Mac and Linux) Shortcut Suggestion

Make sure you have a `.profile` or `.bash_profile` file in your home directory, and that it contains this command:

```
source .bashrc
```

Make sure you have a `.bashrc` file in your home directory, and add this command to it:

```
alias activate333="source ~/.virtualenvs/cos333/bin/activate"
```

thereby defining a Bash alias named `activate333`. Exit any terminal windows, and launch a new terminal window. Thereby bash interprets the new `.bashrc` file. Thereafter you can issue the command `activate333` instead of the more verbose `source ~/.virtualenvs/cos333/bin/activate` to activate your cos333 virtual environment.

(Mac) Shortcut Suggestion

Combining the previous two boxes… You might want to create `showFiles` and `hideFiles` Bash aliases by adding these two lines to your `.bashrc` file:

```
alias showFiles='defaults write com.apple.finder AppleShowAllFiles YES; killall
Finder /System/Library/CoreServices/Finder.app'

alias hideFiles='defaults write com.apple.finder AppleShowAllFiles NO; killall
Finder /System/Library/CoreServices/Finder.app'
```

(MS Windows) Shortcut Suggestion

Create a directory named `bin` immediately subordinate to your home directory. That is, create a directory named `%HOMEPATH%\bin`. In that directory create a file named `activate333.bat` that contains this line:

```
%HOMEPATH%\.virtualenvs\cos333\Scripts\activate.bat
```

(Mac) Shortcut suggestion:

Then in *Control Panel* navigate to *System and Security* → *System* → *Advanced System Settings* → *Environment Variables*. In the *User Variables for yourloginid* area:

If the `Path` environment variable exists, then add the directory `%HOMEPATH%\bin` to its value.

If the `Path` environment variable does not exist, then create it such that its value is the directory `%HOMEPATH%\bin`.

Exit any existing Command Prompt windows, and launch a new Command Prompt window.
Thereafter you can issue the command `activate333` instead of the more verbose `%HOMEPATH%\.virtualenvs\cos333\Scripts\activate.bat` to activate your cos333 virtual environment.

4. Configuring pylint

In COS 217 you used a C static code analysis tool named ***splint****. In COS 333 you'll use a Python static code analysis tool named* ***pylint****. You must configure pylint as appropriate for the course.*

(All) Create a file in your home directory named `.pylintrc` (note the leading period) having these contents:

```
[MASTER]
disable=duplicate-code

[EXCEPTIONS]
overgeneral-exceptions=builtins.BaseException

[MESSAGES CONTROL]
disable=missing-module-docstring,
        missing-class-docstring,
        missing-function-docstring,
        redefined-builtin,
        consider-using-f-string

[FORMAT]
max-line-length=72
max-module-lines=500
```

(All) Creating Your .pylintrc File

To create your `.pylintrc` file, you can use either of these approaches:

(1) Type the contents manually into your favorite editor (VS Code, Sublime Text, etc.), and save to the `.pylintrc` file in your home directory. That approach is error prone, and so is not recommended.

(2) Copy the desired file contents from this document to your computer's clipboard, paste them from your computer's clipboard into your favorite editor, and save to the `.pylintrc` file in your home directory.

5. Configuring Your Computer to Use the SQLite Client

*You must use the **SQLite client** – a program whose name is **sqlite3** – to access SQLite databases that reside on your computer. So you must install the sqlite3 program.*

(Mac) The sqlite3 program probably is bundled with your operating system. To test, in a terminal window issue the command `sqlite3`. Make sure sqlite3 launches. Enter the command `.quit` to exit the program.

(Mac) If the sqlite3 program is not bundled with your operating system, then browse to https://www.sqlite.org/download.html, and download the file named `sqlite-tools-osx-x64-3480000.zip`. In the Finder double click on the downloaded file to unzip it, thereby creating a directory that contains the `sqlite3` file. To test, in a terminal window issue the command `path/sqlite3`, where `path` is the directory that contains the `sqlite3` file. Make sure sqlite3 launches. Enter the command `.quit` to exit the program.

(MS Windows) Browse to https://www.sqlite.org/download.html, and download the file named `sqlite-tools-win-x64-3480000.zip`. In Windows Explorer unzip the downloaded file, thereby creating a directory that contains the `sqlite3.exe` file. To test, in a Command Prompt window issue the command `path\sqlite3.exe`, where `path` is the directory that contains the `sqlite3.exe` file. Make sure sqlite3 launches. Enter the command `.quit` to exit the program.

(Linux) Use your Linux package manager to download and install the `sqlite3` (or some similarly named) package. To test, in a terminal window issue the command `sqlite3`. Make sure sqlite3 launches. Enter the command `.quit` to exit the program.

6. Thereafter...

You are allowed to work with one teammate on each assignment, and we encourage you to do so.

Suppose students Alice and Bob are working together on an assignment. Having configured their computers by following the instructions provided in this document, Alice and Bob should work together on an assignment by following these steps:

Step 1: Alice creates a private GitHub git repository for the assignment, and Alice gives Bob permission to use that GitHub git repository.

Step 2: Alice clones the GitHub git repository to her computer to create a local git repository. Bob also clones the GitHub git repository to his computer to create a local git repository.

Step 3: Alice uses an editor (such as VS Code, Sublime Text, etc.) to create/edit source code files comprising the application and its test programs.

Step 4: Alice uses a Terminal application (on Mac or Linux computers) or a Command Prompt window (on MS Windows computers) to build and run the application and its test programs.

Step 5: When Alice is ready to share with Bob, Alice commits files to her local git repository, and pushes from her local git repository to the GitHub git repository.

Step 6: Bob pulls from the GitHub git repository to his local git repository.

Step 7: Bob uses an editor (such as VS Code, Sublime Text, etc.) to create/edit source code files comprising the application and its test programs.

Step 8: Bob uses a Terminal application (on Mac or Linux computers) or a Command Prompt window (on MS Windows computers) to build and run the application and its test programs.

Step 9: When Bob is ready to share with Alice, Bob commits files to his local git repository, and pushes from his local git repository to the GitHub git repository.

Step 10: Alice pulls from the GitHub git repository to her local git repository.

Step 11: Alice and Bob repeat steps 3-10 as appropriate.

The document from the first lecture entitled *Git and GitHub Primer* provides more precise instructions on using git and GitHub.