

Lab 0.1

Toolset

Z. Hutchinson
zachary.s.hutchinson@maine.edu

August 31, 2021

Introduction

The goal of this lab is to introduce you to the tools programmers use to write, debug and execute computer code. This collection of tools, or programs, is a programmer's toolset. It is important to note that there is no single set, and certainly no *perfect* set, of programming tools to cover all occasions. One's toolset changes depending on the programming language, project and software development approach. More importantly, one's toolset is usually a personal choice. As you gain more programming experience, you will choose tools that most harmonize with your style of work. For this class, we will introduce a default toolset consisting of VSCode (or VSCodium), the Python3 interpreter and your OS's command line program. However, you are free to (and encouraged to) explore the space and use other tools.

Download Links

VSCode can be found here: <https://code.visualstudio.com/>

VSCodium can be found here: <https://vscodium.com/>

The Python Interpreter is available here: <https://www.python.org/downloads/>

NOTE: Windows Users - When installing Python, make sure you check the box that says ADD TO PATH. If you forget, you can run the installer again and *modify* the installation.

Tools

Code Editor

What is a code editor?

A code editor is a program which resembles a typical text editor (e.g. TextEdit, Wordpad, Notepad, etc.) but has additional functionality to aid in the writing of code. For example, many code editors recognize a set of programming languages and will automatically highlight (via color changes) language specific keywords and data types. Another mechanism in some code editor is a *linter* which can identify syntax or other errors in code as you write it, much like word processing software will underline misspelled words.

Which one should I use?

For those who do not have experience with either a multi-language code editor or a Python editor, I suggest you start with the class default which is either VSCode or VSCodium. My reason for suggesting you start with VSCode is that it is:

- A Configurable: It supports more languages than just Python. Meaning you can use it in other CS classes that use a different language.

- B Simple: By simple I mean it does not contain a lot of extra tools. The interface is straight-forward with just a few buttons.
- C Cross-platform: Works on Mac, Linux and Windows. The experience is mostly similar on all platforms.
- D Used by professionals: In other words, it isn't a tool with training wheels that you will never see outside of an educational setting.

Why two suggestions?

What is the difference between the VSCode and VSCodium? They are essentially identical. VSCode was created and is maintained by Microsoft. While the VSCode tool is itself open source, when it is built by Microsoft, it is linked to proprietary libraries used for things like analytics. VSCodium is the same tool built from the same open source, but it is not linked to Microsoft's libraries. From the programmer's perspective, the experience is the same.

There is one other difference. There are some extensions which do not work with VSCodium because they rely on Microsoft servers. One such extension is Live Share.

What are some other suggestions for code editors?

Not all code editors support all languages. Some are designed around specific domains, such as web development. At the end of this document you will find a short list of code editors or IDEs which support Python. The list is not exhaustive by any means.

Python Interpreter

The Python Interpreter is available here: <https://www.python.org/downloads/>

What is an interpreter?

How human-readable computer code is translated and run by the operating system differs across language classes. Python is an *interpreted* language. This means a line is read by the *interpreter*, translated and executed before the interpreter moves onto the next line. This is different from a *compiled* language which translates (compiles) all source code prior to running any part of it. Thus, rather obviously, the Python Interpreter is the program used to interpret Python code. The class of interpreted languages includes JavaScript, Lua, BASIC, and Perl.

How do I use an interpreter?

In this class, we will learn how to run code from the *command line* or *terminal*. Keep in mind that command line syntax differs by platform. However, the basic format for running python code is:

Command Line

```
$ python main.py
```

In the above example, the command to execute a Python script consists of two parts: a call to the interpreter and the name of the file you wish to run. Or in this case, *python* invokes the interpreter and the name of our file is *main.py*. NOTE: the \$ is the command line prompt and is not part of the command. The \$ is the typical linux or mac prompt. The Windows Powershell version would be something like:

Command Line

```
PS C:\Users\zax> python main.py
```

NOTE: Here again I used *python* as the name of the interpreter. However, this does differ across platforms. Some systems differentiate between Python2 and Python3. Linux and Mac do this. *python* typically invokes

the python2 interpreter. *python3* is required to invoke the Python3 interpreter. Others use an abbreviated form, e.g. *py*. If for some reason, none of these suggestions work, please get in touch with an MLA or TA. Additionally, this SO post provides some background info: <https://stackoverflow.com/a/50896577>

Command Line

A code editor answers the question: *How do I write code?* The Python interpreter answers the question: *How do I run the code?* The final question is: *How do I put the two together: interpreter and code?*

In a window-ed environment, we're used to clicking or double-clicking an icon to open files or run programs. But, oddly, in spite of all the advances in human-computer interface, executing code through the command line is still, more often than not, easier once a few basic commands are learned. We will demonstrate several commands in lab; however, for reference I have given a few below.

Access to the command line is through a program that differs between operating systems. On Mac, the program is called: *Terminal* and can be found in *Applications/Utilities/*. On Windows, I suggest using Powershell. On Linux, each distro comes with one or more shell programs: *bash*, *ksh*, or *csh*. Use whatever comes with your distro.

I believe the following commands work on Mac, Windows (Powershell) and Linux. Some of these do not work in the Windows Command Prompt.

- **cd**: Change Directory. This command moves you up or down in the filesystem hierarchy.
- **ls**: List. Lists the contents of the current working directory.
- **pwd**: Print Working Directory. Prints, starting from the root, where you are in the filesystem. This is helpful if your prompt does not display the full path. Windows usually does. Mac and Linux often do not.
- **rm**: Remove (or delete). Deletes files or folders (directories). WARNING: This command is dangerous. When using *rm* make sure you know what you are deleting as *rm* is unrecoverable.
- **mkdir**: Make Directory. This is the same as *New Folder* on most systems.
- **cp**: Copy. Use this to make copies of files.
- **mv**: Move. Use this to move files from one place to another (without copying). You can also use this to rename files.

If you learn only two, learn *cd* and *ls*.

Troubleshooting

While we have tried to pick a minimal toolset that is easy to use across the major platforms, problems do arise. Some of the following will only make sense once you start writing code.

You're in the interactive interpreter

One common mistake students make is they unwittingly invoke Python's interactive interpreter. For example, this happens if you type *python* and hit *Return* without adding a source file. The interactive interpreter allows you to type and run Python code one line at a time via the terminal. If you are seeing an error that looks like this,

Command Line

```
>>> python main.py
File "<stdin>", line 1
python main.py
    ^
SyntaxError: invalid syntax
>>>
```

you are in the interactive interpreter. You can always differentiate between the command line prompt and the interpreter prompt because the interactive interpreter always uses `>>>`.

To exit the interactive interpreter, type:

Command Line

```
>>> quit()
```

You're not in the right directory

Another common error is trying to run code from the wrong place in the filesystem. Imagine this scenario. You have open the contents of your *Downloads* directory. However, you want to open a file that is actually in your *Documents* directory. Before you can open the file, you have to navigate to the correct directory.

The same problem exists when using the command line. Most often you will realize this when you get an error stating *file not found*. Here is an example on a WindowsOS machine:

Command Line

```
PS C:\Users\zax> python main.py
C:\Users\zax\AppData\Local\Programs\Python\Python38-32\python.exe:
can't open file 'main.py': [Errno 2] No such file or directory
PS C:\Users\zax>
```

Notice, I am in my home directory. I am trying to run *main.py*, but it does not exist there. Pay attention to the path (or where you are in the filesystem). If your prompt does not display the path, use the command *pwd*. In the above example, the first line shows that I am in the *zax* directory which is in the *User* directory which is on the *C* drive.

The solution is to either re-open VSCode to the correct directory, or use *cd* to change the directory until you are in the correct place. We will cover both methods in class.

I changed something in my code, but when I run the program the same thing happens.

Short answer: you did not save your source code.

VSCode displays a white circle on the tab of modified files. Make sure you save before running. *CTRL+S* is the save shortcut. As silly as it seems, this still happens to me and it will happen to you.

My VSCode terminal does not display anything.

This seems only to affect Windows users. When opening a new terminal, the terminal appears blank and does not display a prompt. Usually, clicking into the terminal window to set focus and then hitting *RETURN* or *ENTER* will cause the prompt to appear. Once there, it does not disappear until you open a new terminal. I am unsure what causes this.

Closing the terminal window does not kill the terminal process.

This aspect of VSCode will become important once you learn loops and are in danger of writing what are known as *infinite loops*. For now, suffice to say, clicking the *X* on the terminal window does not kill the terminal process. If you have a program running in the terminal, it will continue running. The proper way to close AND kill the terminal is to click the *garbage can* icon. Killing the terminal process stops programs which were started using it.

How can I stop my program while it is running?

The best way is to place focus in the terminal window and hit *CTRL+C*.

EXTRA: Other Code Editors and IDEs

In your search for a better toolset you might come across the abbreviation, IDE. IDE stands for Integrated Development Environment. There is no well-defined line between code editors and IDEs. For now, let's just say that IDEs, at their core, are code editors. The difference is, an IDE comes with an expanded array of tools (e.g. profiler, debugger, refactoring, etc.) and the means to compile and execute the code is *integrated* into the editor itself.

I believe all suggestions below are free or have a free version.

In no particular order:

- **Atom:** <https://atom.io>
- **Sublime:** <https://www.sublimetext.com/>
- **Emacs:** <http://www.gnu.org/software/emacs/>
- **PyCharm:** <https://www.jetbrains.com/pycharm/>
- **Spyder:** <https://www.spyder-ide.org/>
- **PyDev:** <https://www.pydev.org/> (NOTE: This is an extension for Eclipse. I do not recommend this unless you prefer Eclipse over everything else.)