B

TEXT EDITORS AND IDEs

Programmers spend a lot of time writing, reading, and editing code, and using a text editor or an *integrated development environment (IDE)* to make this work as efficient as possible is essential. A good editor will do simple tasks, like highlight your code's structure so you can catch common bugs as you're working. But it won't do so much that it distracts you from your thinking. Editors also have useful features like automatic indenting, markers to show appropriate line length, and keyboard shortcuts for common operations.

An IDE is a text editor with a number of other tools included, like interactive debuggers and code introspection. An IDE examines your code as you enter it and tries to learn about the project you're building. For example, when you start typing the name of a function, an IDE might show you all the arguments that function accepts. This behavior can be very helpful when everything works and you understand what you're seeing. But it can also be overwhelming as a beginner and difficult to troubleshoot when you aren't sure why your code isn't working in the IDE.

I encourage you to use a simple text editor while you're learning to code. Text editors also put a much lighter load on your system; so if you're working on an older machine or one with fewer resources, a text editor will work better than an IDE. If you're already familiar with IDEs or if people around you use an IDE and you want to use a similar environment, by all means try them out.

Don't worry too much about choosing your tools at this point; your time will be better spent digging into the language and working on the projects you're interested in. Once you've mastered the basics, you'll have a better idea what tools work for you.

In this appendix, we'll set up the Sublime Text text editor to help you work more efficiently. We'll also take a brief look at a number of other editors you might consider using or see other Python programmers using.

Customizing Sublime Text Settings

In Chapter 1, you configured Sublime Text to use the Python version you want it to when running your programs. Now we'll configure it to do some of the tasks mentioned at the beginning of this appendix.

Converting Tabs to Spaces

If you use a mix of tabs and spaces in your code, it can cause problems in your programs that are difficult to diagnose. To avoid this, you can configure Sublime Text to always use spaces for indentation, even when you press the TAB key. Go to **View ▸ Indentation** and make sure the **Indent Using Spaces** option is selected. If it's not, select it. Also, make sure the **Tab Width** is set to **4** spaces.

If you already have a mix of tabs and spaces in one of your programs, you can convert all tabs to spaces by clicking **View ▸ Indentation ▸ Convert Tabs to Spaces**. You can also access these settings by clicking **Spaces** at the bottom right of the Sublime Text window.

Now you can use the TAB key to indent lines of code, and Sublime Text will insert spaces automatically to indent those lines.

Setting the Line Length Indicator

Most editors allow you to set up a visual cue, usually a vertical line, to show where your lines should end. In the Python community, the convention is to limit lines to 79 characters or less. To set this feature, select **View**

▸ **Ruler**, and then click **80**. Sublime Text will place a vertical line at the 80-character mark to help restrict your code lines to an appropriate length.

Indenting and Unindenting Code Blocks

To indent an entire block of code, highlight it and select **Edit ▸ Line ▸ Indent** or press CTRL-], or ⌘-] on macOS. To unindent a block of code, click **Edit ▸ Line ▸ Unindent** or press CTRL-[, or ⌘-[.

Commenting Out Blocks of Code

To temporarily disable a block of code, you can highlight the block and comment it so Python will ignore it. Select **Edit ▸ Comment ▸ Toggle Comment** (CTRL-/ or ⌘-/). The selected lines will be commented out with a hash mark (#) indented at the same level as the line of code to indicate these are not regular comments. When you want to uncomment the block of code, highlight the block and reissue the same command.

Saving Your Configuration

Some of the settings mentioned only affect the current file you're working in. To make your settings affect all files you open in Sublime Text, you'll need to define your user settings. Select **Sublime Text ▸ Preferences ▸ Settings**, and look for the file *Preferences.sublime-settings – User*. Enter the following in this file:

```
{
   "rulers": [80],
   "translate_tabs_to_spaces": true
}
```

Save this file, and your ruler and tab settings will apply to all files you work with in Sublime Text. If you add more settings to this file, make sure each line ends with a comma except the last line. You can look at other users' settings files online and customize your editor to the settings that help you work most efficiently.

Further Customizations

You can customize Sublime Text in many ways to help you work even more efficiently. As you're exploring the menus, keep an eye out for keyboard shortcuts for the menu items you use most often. Every time you use a keyboard shortcut instead of reaching for the mouse or trackpad, you become a bit more efficient. Don't try to learn everything at once; just try to become efficient with the actions you use most, and be on the lookout for other features that might help you develop your own workflow.

Other Text Editors and IDEs

You'll hear about and see people using a number of other text editors. Most of them can be configured to help you in the same way you customized Sublime Text. Here's a small selection of text editors you might hear about.

IDLE

IDLE is a text editor that's included with Python. It's a little less intuitive to work with than Sublime Text, but you'll see references to it in other tutorials aimed at beginners, so you might want to give it a try.

Geany

Geany is a simple text editor that lets you run all of your programs directly from the editor. It displays all of your output in a terminal window, which helps you become comfortable using terminals. Geany has a very simple interface, but it's powerful enough that a significant number of experienced programmers still use it.

Emacs and Vim

Emacs and Vim are two popular editors favored by many experienced programmers because they're designed to be used so your hands never have to leave the keyboard. This makes writing, reading, and modifying code very efficient once you learn how the editor works. It also means both editors have a fairly steep learning curve. Vim is included on most

Linux and macOS machines, and both Emacs and Vim can be run entirely inside a terminal. For this reason, they're often used to write code on servers through a remote terminal session.

Programmers will often recommend that you give them a try. But many proficient programmers forget how much new programmers are already trying to learn. It's beneficial to be aware of these editors, but hold off on using them until you're comfortable working with code in a simpler editor that lets you focus on learning to program rather than learning to use an editor.

Atom

Atom is a text editor with some features that you'd normally find in an IDE. You can open an individual file you're working on, or you can open a project folder and Atom will instantly make all the files in that project easily accessible. Atom is integrated with Git and GitHub, so as you start to use version control you'll be able to work with local and remote repositories from within your editor instead of having to do so in a separate terminal.

Atom allows you to install packages as well, so you can extend its behavior in many ways. A number of these packages incorporate behavior that makes Atom behave more like an IDE.

Visual Studio Code

Visual Studio Code, also called VS Code, is another editor that acts more like an IDE. VS Code supports efficient use of a debugger, has integrated version control support, and also offers code completion tools.

PyCharm

PyCharm is a popular IDE among Python programmers because it was built to work specifically with Python. The full version requires a paid subscription, but a free version called the PyCharm Community Edition is also available that many developers find useful.

PyCharm features a *linter*, which checks that your coding style matches common Python conventions, and offers suggestions when you deviate from normal Python formatting. It also has an integrated debugger to help you resolve errors proficiently and modes that help you work efficiently with a number of popular Python libraries.

Jupyter Notebooks

Jupyter Notebook is a different kind of tool than traditional text editors or IDEs in that it's a web app primarily built of blocks; each block is either a code block or a text block. The text blocks are rendered in Markdown, so you can include simple formatting in your text blocks.

Jupyter Notebooks were developed to support the use of Python in scientific applications, but they have since expanded to become useful in a wide variety of situations. Rather than just writing comments inside a *.py* file, you can write clear text with simple formatting, such as headers, bulleted lists, and hyperlinks in between sections of code. Every code block can be run independently, allowing you to test small pieces of your program, or you can run all the code blocks at once. Each code block has its own output area, and you can toggle the output areas on or off as needed.

Jupyter Notebooks can be confusing at times because of the interactions between different cells. If you define a function in one cell, that function is available to other cells as well. This is beneficial most of the time, but it can be confusing in longer notebooks and if you don't fully understand how the Notebook environment works.

If you're doing any scientific or data-focused work in Python, you'll almost certainly see Jupyter Notebooks at some point.