Part V
The Part of Tens

Enjoy an additional Part of Tens article about ten sites with unique designs at www.dummies.com/extras/beginningprogrammingwithpython.

In this part

- Discover really cool resources that you can use to make your Python programming experience better.
- Earn a living with the Python knowledge you gain.
- Get the tools you need to work more efficiently with Python.
- Make Python do even more by adding libraries.

Chapter 17

Ten Amazing Programming Resources

In This Chapter

- Using the Python documentation
- Accessing an interactive Python tutorial
- Creating online applications using Python
- Extending Python using third-party libraries
- Obtaining a better editor than Python's for Python application development
- Getting the syntax for your Python application correct
- Working with XML
- Becoming a professional coder with less effort than usual
- Overcoming the Unicode obstacle
- Creating applications that run fast

This book is a great start to your Python programming experience, but you'll want additional resources at some point. This chapter provides you with ten amazing programming resources that you can use to make your development experience better. By using these resources, you save both time and energy in creating your next dazzling Python application.

Of course, this chapter is only the beginning of your Python resource experience. Literally reams of Python documentation are out there, along with mountains of Python code. It might be possible to write an entire book (or two) on just the Python libraries. This chapter is designed to provide you with ideas of where to look for additional information that's targeted toward meeting your specific needs. Don't let this be the end of your search — consider this chapter the start of your search instead.

Working with the Python Documentation Online

An essential part of working with Python is knowing what is available in the base language and how to extend it to perform other tasks. The Python documentation at `https://docs.python.org/3/` (created for the 3.4.1 version of the product at the time of this writing; it may be updated by the time you read this chapter) contains a lot more than just the reference to the language that you receive as part of a download. In fact, you see these topics discussed as part of the documentation:

- New features in the current version of the language
- Access to a full-fledged tutorial
- Complete library reference
- Complete language reference
- How to install and configure Python
- How to perform specific tasks in Python
- Help with installing Python modules from other sources (as a means of extending Python)
- Help with distributing Python modules you create so that others can use them
- How to extend Python using C/C++ and then embed the new features you create
- Complete reference for C/C++ developers who want to extend their applications using Python
- Frequently Asked Questions (FAQ) pages

All this information is provided in a form that is easy to access and use. In addition to the usual table-of-contents approach to finding information, you have access to a number of indexes. For example, if you aren't interested in anything but locating a particular module, class, or method, you can use the Global Module Index.

The `https://docs.python.org/3/` web page is also the place where you report problems with Python. It's important to work through problems you're having with the product, but as with any other language, Python does have bugs in it. Locating and destroying the bugs will only make Python a better language.

Using the LearnPython.org Tutorial

Many tutorials are available for Python and many of them do a great job, but they're all lacking a special feature that you find when using the LearnPython.org tutorial at `http://www.learnpython.org/` — interactivity. Instead of just reading about a Python feature, you read it and then try it yourself using the interactive feature of the site.

You have already worked through all the material in the simple tutorials in this book. However, you haven't worked through the advanced tutorials yet. These tutorials present the following topics:

- **Generators:** Specialized functions that return iterators.
- **List comprehensions:** A method to generate new lists based on existing lists.
- **Multiple function arguments:** An extension of the methods described in the "Using methods with variable argument lists" in Chapter 14.
- **Regular expressions:** Wildcard setups used to match patterns of characters, such as telephone numbers.
- **Exception handling:** An extension of the methods described in Chapter 9.
- **Sets:** Demonstrates a special kind of list that never contains duplicate entries.
- **Serialization:** Shows how to use a data storage methodology called JavaScript Object Notation (JSON).
- **Partial functions:** A technique for creating specialized versions of simple functions that derive from more complex functions. For example, if you have a `multiply()` function that requires two arguments, a partial function named `double()` might require only one argument that it always multiplies by 2.
- **Code introspection:** Provides the ability to examine classes, functions, and keywords to determine their purpose and capabilities.
- **Decorator:** A method for making simple modifications to callable objects.

Performing Web Programming Using Python

This book discusses the ins and outs of basic programming, so it relies on desktop applications because of their simplicity. However, many developers specialize in creating online applications of various sorts using Python. The Web Programming in Python site at `https://wiki.python.org/moin/WebProgramming` helps you make the move from the desktop to online application development. It doesn't just cover one sort of online application — it covers almost all of them (an entire book free for the asking). The tutorials are divided into these three major (and many minor) areas:

- Server
  - Developing server-side frameworks for applications
  - Creating a Common Gateway Interface (CGI) script
  - Providing server applications
  - Developing Content Management Systems (CMS)
  - Designing data access methods through web services solutions
- Client
  - Interacting with browsers and browser-based technologies
  - Creating browser-based clients
  - Accessing data through various methodologies, including web services
- Related
  - Creating common solutions for Python-based online computing
  - Interacting with DataBase Management Systems (DBMSs)
  - Designing application templates
  - Building Intranet solutions

Getting Additional Libraries

The Pythonware site (`http://www.pythonware.com/`) doesn't look all that interesting until you start clicking the links. It provides you with access to a number of third-party libraries that help you perform additional tasks

using Python. Although all the links provide you with useful resources, the "Downloads (`downloads.effbot.org`)" link is the one you should look at first. This download site provides you with access to

- **aggdraw:** A library that helps you create anti-aliased drawings.
- **celementtree:** An add-on to the elementtree library that makes working with XML data more efficient and faster.
- **console:** An interface for Windows that makes it possible to create better console applications.
- **effbot:** A collection of useful add-ons and utilities, including the EffNews RSS news reader.
- **elementsoap:** A library that helps you create Simple Object Access Protocol (SOAP) connections to Web services providers.
- **elementtidy:** An add-on to the elementtree library that helps you create nicer-looking and more functional XML tree displays than the standard ones in Python.
- **elementtree:** A library that helps you interact with XML data more efficiently than standard Python allows.
- **exemaker:** A utility that creates an executable program from your Python script so that you can execute the script just as you would any other application on your machine.
- **ftpparse:** A library for working with FTP sites.
- **grabscreen:** A library for performing screen captures.
- **imaging:** Provides the source distribution to the Python Imaging Library (PIL) that lets you add image-processing capabilities to the Python interpreter. Having the source lets you customize PIL to meet specific needs.
- **pil:** Binary installers for PIL, which make obtaining a good installation for your system easier. (There are other PIL-based libraries as well, such as pilfont — a library for adding enhanced font functionality to a PIL-based application.)
- **pythondoc:** A utility for creating documentation from the comments in your Python code that works much like JavaDoc.
- **squeeze:** A utility for converting your Python application contained in multiple files into a one- or two-file distribution that will execute as normal with the Python interpreter.
- **tkinter3000:** A widget-building library for Python that includes a number of subproducts. *Widgets* are essentially bits of code that create controls, such as buttons, to use in GUI applications. There are a number of add-ons for the tkinter3000 library, such as wckgraph, which helps you add graphing support to an application.

Creating Applications Faster Using an IDE

An Interactive Development Environment (IDE) helps you create applications in a specific language. The Integrated DeveLopment Environment (IDLE) editor that comes with Python worked well for the needs of the book, but you may find it limited after a while. For example, IDLE doesn't provide the advanced debugging functionality that many developers favor. In addition, you may find that you want to create graphical applications, which is difficult using IDLE.

You can talk to 50 developers and get little consensus as to the best tool for any job, especially when discussing IDEs. Every developer has a favorite product and isn't easily swayed to try another. Developers invest many hours learning a particular IDE and extending it to meet specific requirements (when the IDE allows such tampering).

An inability (at times) to change IDEs later is why it's important to try a number of different IDEs before you settle on one. (The most common reason for not wanting to change an IDE after you select one is that the project types are incompatible, which would mean having to re-create your projects every time you change editors, but there are many other reasons that you can find listed online.) The PythonEditors wiki at `https://wiki.python.org/moin/PythonEditors` provides an extensive list of IDEs that you can try. The table provides you with particulars about each editor so that you can eliminate some of the choices immediately.

Checking Your Syntax with Greater Ease

The IDLE editor provides some level of syntax highlighting, which is helpful in finding errors. For example, if you mistype a keyword, it doesn't change color to the color used for keywords on your system. Seeing that it hasn't changed makes it possible for you to know to correct the error immediately, instead of having to run the application and find out later that something has gone wrong (sometimes after hours of debugging).

The python.vim utility (`http://www.vim.org/scripts/script.php?script_id=790`) provides enhanced syntax highlighting that makes finding errors in your Python script even easier. This utility runs as a script, which makes it fast and efficient to use on any platform. In addition, you can tweak the source code as needed to meet particular needs.

Using XML to Your Advantage

The eXtensible Markup Language (XML) is used for data storage of all types in most applications of any substance today. You probably have a number of XML files on your system and don't even know it because XML data appears under a number of file extensions. For example, many `.config` files, used to hold application settings, rely on XML. In short, it's not a matter of if you'll encounter XML when writing Python applications, but when.

XML has a number of advantages over other means of storing data. For example, it's platform independent. You can use XML on any system, and the same file is readable on any other system as long as that system knows the file format. The platform independence of XML is why it appears with so many other technologies, such as web services. In addition, XML is relatively easy to learn and because it's text, you can usually fix problems with it without too many problems.

It's important to learn about XML itself, and you can do so using an easy tutorial such as the one found on the W3Schools site at `http://www.w3schools.com/xml/default.ASP`. Some developers rush ahead and later find that they can't understand the Python-specific materials that assume they already know how to write basic XML files. The W3Schools site is nice because it breaks up the learning process into chapters so that you can work with XML a little at a time, as follows:

- Taking a basic XML tutorial
- Validating your XML files
- Using XML with JavaScript (which may not seem important, but JavaScript is prominent in many online application scenarios)
- Gaining an overview of XML-related technologies
- Using advanced XML techniques
- Working with XML examples that make seeing XML in action easier

Using W3Schools to your advantage

One of the most used online resources for learning online computing technologies is W3Schools. You can find the main page at `http://www.w3schools.com/`. This single resource can help you discover every web technology needed to build any sort of modern application you can imagine. The topics include:

- HTML
- CSS
- JavaScript
- SQL
- JQuery
- PHP
- XML
- ASP.NET

However, you should realize that this is just a starting point for Python developers. Use the W3Schools material to get a good handle on the underlying technology, and then rely on Python-specific resources to build your skills. Most Python developers need a combination of learning materials to build the skills required to make a real difference in application coding.

After you get the fundamentals down, you need a resource that shows how to use XML with Python. One of the better places to find this information is the Python and XML Processing site at `http://pyxml.sourceforge.net/topics/`. Between these two resources, you can quickly build a knowledge of XML that will have you building Python applications that use XML in no time.

Getting Past the Common Python Newbie Errors

Absolutely everyone makes coding mistakes — even that snobby fellow down the hall who has been programming for the last 30 years (he started in kindergarten). No one likes to make mistakes and some people don't like to own up to them, but everyone does make them. So you shouldn't feel too bad when you make a mistake. Simply fix it up and get on with your life.

Of course, there is a difference between making a mistake and making an avoidable, common mistake. Yes, even the professionals sometimes make the common mistakes, but it's far less likely because they have seen the mistake in the past and have trained themselves to avoid it. You can gain an advantage over your competition by avoiding the newbie mistakes that everyone has to learn about sometime. To avoid these mistakes, check out this two-part series:

- Python: Common Newbie Mistakes, Part 1 (http://blog.amir.rachum.com/blog/2013/07/06/python-common-newbie-mistakes-part-1/)
- Python: Common Newbie Mistakes, Part 2 (http://blog.amir.rachum.com/blog/2013/07/09/python-common-newbie-mistakes-part-2/)

Many other resources are available for people who are just starting with Python, but these particular resources are succinct and easy to understand. You can read them in a relatively short time, make some notes about them for later use, and avoid those embarrassing errors that everyone tends to remember.

Understanding Unicode

Although this book tries to sidestep the thorny topic of Unicode, you'll eventually encounter it when you start writing serious applications. Unfortunately, Unicode is one of those topics that had a committee deciding what Unicode would look like, so we ended up with more than one poorly explained definition of Unicode and a multitude of standards to define it. In short, there is no one definition for Unicode.

You'll encounter a wealth of Unicode standards when you start working with more advanced Python applications, especially when you start working with multiple human languages (each of which seems to favor its own flavor of Unicode). Keeping in mind the need to discover just what Unicode is, here are some resources you should check out:

- The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!) (http://www.joelonsoftware.com/articles/Unicode.html)
- The Updated Guide to Unicode on Python (http://lucumr.pocoo.org/2013/7/2/the-updated-guide-to-unicode/)
- Python Encodings and Unicode (http://eric.themoritzfamily.com/python-encodings-and-unicode.html)
- Unicode Tutorials and Overviews (http://www.unicode.org/standard/tutorial-info.html)
- Explain it like I'm five: Python and Unicode? (http://www.reddit.com/r/Python/comments/1g62eh/explain_it_like_im_five_python_and_unicode/)
- Unicode Pain (http://nedbatchelder.com/text/unipain.html)

Making Your Python Application Fast

Nothing turns off a user faster than an application that performs poorly. When an application performs poorly, you can count on users not using it at all. In fact, poor performance is a significant source of application failure in enterprise environments. An organization can spend a ton of money to build an impressive application that does everything, but no one uses it because it runs too slowly or has other serious performance problems.

Performance is actually a mix of reliability, security, and speed. In fact, you can read about the performance triangle on my blog at `http://blog.johnmuellerbooks.com/2012/04/16/considering-the-performance-triangle/`. Many developers focus on just the speed part of performance but end up not achieving their goal. It's important to look at every aspect of your application's use of resources and to ensure that you use the best coding techniques.

Numerous resources are available to help you understand performance as it applies to Python applications. However, one of the best resources out there is "A guide to analyzing Python performance," at `http://www.huyng.com/posts/python-performance-analysis/`. The author takes the time to explain why something is a performance bottleneck, rather than simply tell you that it is. After you read this article, make sure to check out the PythonSpeed Performance Tips at `https://wiki.python.org/moin/PythonSpeed/PerformanceTips` as well.