Richard Blum and Christine Bresnahan

Sams Teach Yourself
Python Programming for Raspberry Pi

in 24 Hours
SECOND EDITION

SAMS
800 East 96th Street, Indianapolis, Indiana, 46240 USA
# Contents at a Glance

Introduction .................................................................................... 1

Part I: Python Programming on the Raspberry Pi

**HOUR 1** Setting Up the Raspberry Pi .............................................................. 5
  2 Understanding the Raspbian Linux Distribution ............................... 29
  3 Setting Up a Programming Environment ......................................... 47

Part II: Python Fundamentals

**HOUR 4** Understanding Python Basics .................................................. 73
  5 Using Arithmetic in Your Programs ................................................. 99
  6 Controlling Your Program ............................................................ 117
  7 Learning About Loops .................................................................. 137

Part III: Advanced Python

**HOUR 8** Using Lists and Tuples ............................................................ 159
  9 Dictionaries and Sets .................................................................... 179
  10 Working with Strings ................................................................. 207
  11 Using Files ................................................................................... 225
  12 Creating Functions ..................................................................... 249
  13 Working with Modules ................................................................. 269
  14 Exploring the World of Object-Oriented Programming .................... 291
  15 Employing Inheritance ................................................................. 307
  16 Regular Expressions .................................................................... 331
  17 Exception Handling ..................................................................... 351

Part IV: Graphical Programming

**HOUR 18** GUI Programming ................................................................. 373
  19 Game Programming .................................................................... 397
Contents at a Glance

Part V: Business Programming

HOUR 20 Using the Network ................................................................. 427
HOUR 21 Using Databases in Your Programming ................................. 453
HOUR 22 Web Programming ............................................................... 475

Part VI: Raspberry Pi Python Projects

HOUR 23 Creating Basic Pi/Python Projects ......................................... 497
HOUR 24 Working with Advanced Pi/Python Projects .......................... 533

Appendixes

A Loading the Raspbian Operating System onto an SD Card .............. 557
B Raspberry Pi Models Synopsis ......................................................... 567

Index ............................................................................................. 573
# Table of Contents

**Introduction** .............................................................. 1
  Programming with Python .............................................. 1
  Who Should Read This Book? ........................................... 3
  Conventions Used in This Book ........................................ 3

**Part I: Python Programming on the Raspberry Pi**

**HOUR 1: Setting Up the Raspberry Pi** .......................... 5
  Obtaining a Raspberry Pi ................................................ 5
  Acquiring a Raspberry Pi ................................................ 8
  Determining the Necessary Peripherals ............................ 10
  Nice Additional Peripherals ............................................. 15
  Deciding How to Purchase Peripherals .............................. 18
  Getting Your Raspberry Pi Working ................................. 18
  Troubleshooting Your Raspberry Pi ................................. 24
  Summary ........................................................................ 26
  Q&A ............................................................................. 26
  Workshop ....................................................................... 27

**HOUR 2: Understanding the Raspbian Linux Distribution** 29
  Learning About Linux .................................................... 29
  Interacting with the Raspbian Command Line .................. 30
  Interacting with the Raspbian GUI ................................. 35
  The LXDE Graphical Interface ........................................ 36
  Summary ........................................................................ 43
  Q&A ............................................................................. 44
  Workshop ....................................................................... 44

**HOUR 3: Setting Up a Programming Environment** .......... 47
  Exploring Python .......................................................... 47
  Checking Your Python Environment ................................. 48
  Installing Python and Tools ............................................ 50
Learning About the Python Interpreter ...................................................... 52
Learning About the Python Interactive Shell .............................................. 53
Learning About the Python Development Environment ............................... 57
Creating and Running Python Scripts ....................................................... 63
Knowing Which Tool to Use and When ....................................................... 68
Summary ................................................................................................ 69
Q&A ....................................................................................................... 69
Workshop ................................................................................................ 70

Part II: Python Fundamentals

HOUR 4: Understanding Python Basics .......................................................... 73
  Producing Python Script Output ............................................................... 73
  Formatting Scripts for Readability ............................................................ 80
  Understanding Python Variables .............................................................. 83
  Assigning Value to Python Variables ....................................................... 85
  Learning About Python Data Types .......................................................... 89
  Allowing Python Script Input .................................................................... 90
  Summary ................................................................................................ 96
  Q&A ....................................................................................................... 97
  Workshop ................................................................................................ 97

HOUR 5: Using Arithmetic in Your Programs .................................................. 99
  Working with Math Operators ................................................................. 99
  Calculating with Fractions ..................................................................... 105
  Using Complex Number Math ............................................................... 107
  Getting Fancy with the math Module ...................................................... 108
  Using the NumPy Math Libraries ............................................................ 112
  Summary .............................................................................................. 114
  Q&A ..................................................................................................... 115
  Workshop .............................................................................................. 115

HOUR 6: Controlling Your Program .............................................................. 117
  Working with the if Statement ............................................................... 117
  Grouping Multiple Statements ............................................................... 119
  Adding Other Options with the else Statement ....................................... 121
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adding More Options Using the <code>elif</code> Statement</td>
<td>123</td>
</tr>
<tr>
<td>Comparing Values in Python</td>
<td>126</td>
</tr>
<tr>
<td>Checking Complex Conditions</td>
<td>130</td>
</tr>
<tr>
<td>Negating a Condition Check</td>
<td>131</td>
</tr>
<tr>
<td>Summary</td>
<td>132</td>
</tr>
<tr>
<td>Q&amp;A</td>
<td>132</td>
</tr>
<tr>
<td>Workshop</td>
<td>133</td>
</tr>
<tr>
<td><strong>HOUR 7: Learning About Loops</strong></td>
<td>137</td>
</tr>
<tr>
<td>Performing Repetitive Tasks</td>
<td>137</td>
</tr>
<tr>
<td>Using the <code>for</code> Loop for Iteration</td>
<td>137</td>
</tr>
<tr>
<td>Using the <code>while</code> Loop for Iteration</td>
<td>148</td>
</tr>
<tr>
<td>Creating Nested Loops</td>
<td>154</td>
</tr>
<tr>
<td>Summary</td>
<td>156</td>
</tr>
<tr>
<td>Q&amp;A</td>
<td>156</td>
</tr>
<tr>
<td>Workshop</td>
<td>157</td>
</tr>
<tr>
<td><strong>Part III: Advanced Python</strong></td>
<td>159</td>
</tr>
<tr>
<td><strong>HOUR 8: Using Lists and Tuples</strong></td>
<td>159</td>
</tr>
<tr>
<td>Introducing Tuples</td>
<td>159</td>
</tr>
<tr>
<td>Introducing Lists</td>
<td>164</td>
</tr>
<tr>
<td>Using Multidimensional Lists to Store Data</td>
<td>171</td>
</tr>
<tr>
<td>Working with Lists and Tuples in Your Scripts</td>
<td>172</td>
</tr>
<tr>
<td>Creating Lists by Using List Comprehensions</td>
<td>173</td>
</tr>
<tr>
<td>Working with Ranges</td>
<td>174</td>
</tr>
<tr>
<td>Summary</td>
<td>175</td>
</tr>
<tr>
<td>Q&amp;A</td>
<td>175</td>
</tr>
<tr>
<td>Workshop</td>
<td>176</td>
</tr>
<tr>
<td><strong>HOUR 9: Dictionaries and Sets</strong></td>
<td>179</td>
</tr>
<tr>
<td>Understanding Python Dictionary Terms</td>
<td>179</td>
</tr>
<tr>
<td>Exploring Dictionary Basics</td>
<td>180</td>
</tr>
<tr>
<td>Programming with Dictionaries</td>
<td>186</td>
</tr>
<tr>
<td>Understanding Python Sets</td>
<td>192</td>
</tr>
<tr>
<td>Exploring Set Basics</td>
<td>193</td>
</tr>
<tr>
<td>HOUR 10: Working with Strings</td>
<td>207</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----</td>
</tr>
<tr>
<td>The Basics of Using Strings</td>
<td>207</td>
</tr>
<tr>
<td>Using Functions to Manipulate Strings</td>
<td>210</td>
</tr>
<tr>
<td>Formatting Strings for Output</td>
<td>217</td>
</tr>
<tr>
<td>Summary</td>
<td>223</td>
</tr>
<tr>
<td>Q&amp;A</td>
<td>223</td>
</tr>
<tr>
<td>Workshop</td>
<td>223</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HOUR 11: Using Files</th>
<th>225</th>
</tr>
</thead>
<tbody>
<tr>
<td>Understanding Linux File Structures</td>
<td>225</td>
</tr>
<tr>
<td>Managing Files and Directories via Python</td>
<td>227</td>
</tr>
<tr>
<td>Opening a File</td>
<td>229</td>
</tr>
<tr>
<td>Reading a File</td>
<td>233</td>
</tr>
<tr>
<td>Closing a File</td>
<td>239</td>
</tr>
<tr>
<td>Writing to a File</td>
<td>240</td>
</tr>
<tr>
<td>Summary</td>
<td>246</td>
</tr>
<tr>
<td>Q&amp;A</td>
<td>246</td>
</tr>
<tr>
<td>Workshop</td>
<td>247</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HOUR 12: Creating Functions</th>
<th>249</th>
</tr>
</thead>
<tbody>
<tr>
<td>Utilizing Python Functions in Your Programs</td>
<td>249</td>
</tr>
<tr>
<td>Returning a Value</td>
<td>253</td>
</tr>
<tr>
<td>Passing Values to Functions</td>
<td>254</td>
</tr>
<tr>
<td>Handling Variables in a Function</td>
<td>260</td>
</tr>
<tr>
<td>Using Lists with Functions</td>
<td>263</td>
</tr>
<tr>
<td>Using Recursion with Functions</td>
<td>264</td>
</tr>
<tr>
<td>Summary</td>
<td>265</td>
</tr>
<tr>
<td>Q&amp;A</td>
<td>265</td>
</tr>
<tr>
<td>Workshop</td>
<td>266</td>
</tr>
<tr>
<td>Working with Web Servers</td>
<td>436</td>
</tr>
<tr>
<td>Linking Programs Using Socket Programming</td>
<td>442</td>
</tr>
<tr>
<td>Summary</td>
<td>449</td>
</tr>
<tr>
<td>Q&amp;A</td>
<td>449</td>
</tr>
<tr>
<td>Workshop</td>
<td>450</td>
</tr>
</tbody>
</table>

**HOUR 21: Using Databases in Your Programming**  
| Working with the MySQL Database              | 453 |
| Using the PostgreSQL Database                | 464 |
| Summary                                      | 472 |
| Q&A                                          | 472 |
| Workshop                                     | 473 |

**HOUR 22: Web Programming**  
| Running a Web Server on the Pi               | 475 |
| Programming with the Common Gateway Interface| 478 |
| Expanding Your Python Webpages               | 481 |
| Processing Forms                             | 488 |
| Summary                                      | 493 |
| Q&A                                          | 494 |
| Workshop                                     | 494 |

**Part VI: Raspberry Pi Python Projects**

**HOUR 23: Creating Basic Pi/Python Projects**  
| Thinking About Basic Pi/Python Projects      | 497 |
| Displaying HD Images via Python              | 497 |
| Playing Music                                | 517 |
| Summary                                      | 530 |
| Q&A                                          | 530 |
| Workshop                                     | 530 |

**HOUR 24: Working with Advanced Pi/Python Projects**  
| Exploring the GPIO Interface                 | 533 |
| Using the RPi.GPIO Module                     | 539 |
| Controlling GPIO Output                       | 541 |
| Detecting GPIO Input                          | 546 |
Summary .............................................................................................. 553
Q&A ..................................................................................................... 553
Workshop .............................................................................................. 554

Appendixes

APPENDIX A: Loading the Raspbian Operating System onto an SD Card .... 557
  Downloading NOOBS ................................................................. 558
  Verifying NOOBS Checksum ...................................................... 559
  Unpacking the NOOBS Zip File ................................................... 561
  Formatting the MicroSD Card ..................................................... 562
  Copying NOOBS to a MicroSD Card ........................................... 566

APPENDIX B: Raspberry Pi Models Synopsis ........................................ 567
  Raspberry Pi 2 Model B .............................................................. 567
  Raspberry Pi 1 Model B+ ............................................................ 568
  Raspberry Pi 1 Model A+ ............................................................ 569
  Older Raspberry Pi Models ......................................................... 570

Index ..................................................................................................... 573
About the Authors

Richard Blum has worked in the IT industry for more than 30 years as a network and systems administrator, managing Microsoft, Unix, Linux, and Novell servers for a network with more than 3,500 users. He has developed and teaches programming and Linux courses via the Internet to colleges and universities worldwide. Rich has a master’s degree in management information systems from Purdue University and is the author of several Linux books, including Linux Command Line and Shell Scripting Bible (coauthored with Christine Bresnahan); Linux for Dummies, Ninth Edition; and Professional Linux Programming (coauthored with Jon Masters). When he’s not busy being a computer nerd, Rich enjoys spending time with his wife, Barbara, and two daughters, Katie Jane and Jessica.

Christine Bresnahan started working in the IT industry more than 30 years ago as a system administrator. Christine is currently an adjunct professor at Ivy Tech Community College in Indianapolis, Indiana, teaching Python programming, Linux system administration, and Linux security classes. Christine produces Unix/Linux educational material and is the author of Linux Bible, Eighth Edition (coauthored with Christopher Negus) and Linux Command Line and Shell Scripting Bible (coauthored with Richard Blum). She has been an enthusiastic owner of a Raspberry Pi since 2012.
Dedication

To the Lord God Almighty.

“I am the vine, you are the branches; he who abides in Me and I in him, he bears much fruit, for apart from Me you can do nothing.”

—John 15:5

Acknowledgments

First, all glory and praise go to God, who through His Son, Jesus Christ, makes all things possible and gives us the gift of eternal life.

Many thanks go to the fantastic team of people at Sams Publishing for their outstanding work on this project. Thanks to Rick Kughen, the executive editor, for offering us the opportunity to work on this book and keeping things on track. We are grateful to development editor Mark Renfrow, who provided diligence in making our work more presentable. Thanks to the production editor, Seth Kerney, for making sure the book was produced. Many thanks to the copy editor, Megan Wade-Taxter, for her endless patience and diligence in making our work readable. Also, we are indebted to our technical editor, Kevin E. Ryan, who put in many long hours double-checking all our work and keeping the book technically accurate.

Thanks to Tonya of Tonya Wittig Photography, who created incredible pictures of our Raspberry Pis and was very patient in taking all the photos we wanted for the book, and to the talented Daniel Anez (theanez.com) for his illustration work. We would also like to thank Carole Jelen at Waterside Productions, Inc., for arranging this opportunity for us and for helping us out in our writing careers.

Christine would also like to thank her student, Paul Bohall, for introducing her to the Raspberry Pi, and her husband, Timothy, for his encouragement to pursue the “geeky stuff” students introduce her to.
We Want to Hear from You!

As the reader of this book, you are our most important critic and commentator. We value your opinion and want to know what we’re doing right, what we could do better, what areas you’d like to see us publish in, and any other words of wisdom you’re willing to pass our way.

We welcome your comments. You can email or write to let us know what you did or didn’t like about this book—as well as what we can do to make our books better.

*Please note that we cannot help you with technical problems related to the topic of this book.*

When you write, please be sure to include this book’s title and author as well as your name and email address. We will carefully review your comments and share them with the author and editors who worked on the book.

Email: consumer@samspublishing.com

Mail: Sams Publishing
     ATTN: Reader Feedback
     800 East 96th Street
     Indianapolis, IN 46240 USA

Reader Services

Visit our website and register this book at informit.com/register for convenient access to any updates, downloads, or errata that might be available for this book.
This page intentionally left blank
Introduction

Officially launched in February 2012, the Raspberry Pi personal computer took the world by storm, selling out the 10,000 available units immediately. It is an inexpensive credit card–sized exposed circuit board, a fully programmable PC running the free open-source Linux operating system. The Raspberry Pi can connect to the Internet, can be plugged into a TV, and—with the latest version 2—runs on a fast ARM processor, rivaling the performance of many tablet devices, all for around $35.

Originally created to spark schoolchildren’s interest in computers, the Raspberry Pi has caught the attention of home hobbyists, entrepreneurs, and educators worldwide. Estimates put the sales figures around 6 million units as of June 2015.

The official programming language of the Raspberry Pi is Python. Python is a flexible programming language that runs on almost any platform. Thus, a program can be created on a Windows PC or Mac and run on the Raspberry Pi, and vice versa. Python is an elegant, reliable, powerful, and very popular programming language. Making Python the official programming language of the popular Raspberry Pi was genius.

Programming with Python

The goal of this book is to help guide both students and hobbyists through using the Python programming language on a Raspberry Pi. You don’t need to have any programming experience to benefit from this book; we walk through all the necessary steps in getting your Python programs up and running!

Part I, “Python Programming on the Raspberry Pi,” walks through the core Raspberry Pi system and how to use the Python environment that’s already installed in it. Hour 1, “Setting Up the Raspberry Pi,” demonstrates how to set up a Raspberry Pi system, and then in Hour 2, “Understanding the Raspbian Linux Distribution,” we take a closer look at Raspbian—the Linux distribution designed specifically for the Raspberry Pi. Hour 3, “Setting Up a Programming Environment,” examines the various ways you can run your Python programs on the Raspberry Pi, and it goes through some tips on how to build your programs.
Part II, “Python Fundamentals,” focuses on the Python 3 programming language. Python v3 is the newest version of Python and is fully supported in the Raspberry Pi. Hours 4–7 take you through the basics of Python programming, from simple assignment statements (Hour 4, “Understanding Python Basics”), arithmetic (Hour 5, “Using Arithmetic in Your Programs”), and structured commands (Hour 6, “Controlling Your Program”), to complex structured commands (Hour 7, “Learning About Loops”).

Hour 8, “Using Lists and Tuples,” and Hour 9, “Dictionaries and Sets,” kick off Part III, “Advanced Python,” showing how to use some of the fancier data structures supported by Python—lists, tuples, dictionaries, and sets. You’ll use these a lot in your Python programs, so it helps to know all about them!

In Hour 10, “Working with Strings,” we take a little extra time to go over how Python handles text strings. String manipulation is a hallmark of the Python programming language, so we want to make sure you’re comfortable with how that all works.

After that primer, we walk through some more complex concepts in Python: using files (Hour 11, “Using Files”), creating your own functions (Hour 12, “Creating Functions”), creating your own modules (Hour 13, “Working with Modules”), object-oriented Python programming (Hour 14, “Exploring the World of Object-Oriented Programming”), inheritance (Hour 15, “Employing Inheritance”), regular expressions (Hour 16, “Regular Expressions”), and working with exceptions (Hour 17, “Exception Handling”).

Part IV, “Graphical Programming,” is devoted to using Python to create real-world applications. Hour 18, “GUI Programming,” discusses GUI programming so you can create your own windows applications. Hour 19, “Game Programming,” introduces you to the world of Python game programming.

Part V, “Business Programming,” takes a look at some business-oriented applications you can create. In Hour 20, “Using the Network,” we look at how to incorporate network functions such as email and retrieving data from webpages into your Python programs. Hour 21, “Using Databases in Your Programming,” shows how to interact with popular Linux database servers, and Hour 22, “Web Programming,” demonstrates how to write Python programs you can access from across the Web.

Who Should Read This Book?

This book is aimed at readers interested in getting the most from their Raspberry Pi system by writing their own Python programs, including these three groups:

- Students interested in an inexpensive way to learn Python programming
- Hobbyists who want to get the most out of their Raspberry Pi system
- Entrepreneurs looking for an inexpensive Linux platform to use for application deployment

If you are reading this book, you are not necessarily new to programming, but you might be new to using Python programming, or at least Python programming in the Raspberry Pi environment. This book will prove to be a good resource for quickly finding Python features and modules that you can use for all types of programs.

Conventions Used in This Book

To make your life easier, this book includes various features and conventions that help you get the most out of this book and out of your Raspberry Pi:

<table>
<thead>
<tr>
<th>Steps</th>
<th>Throughout the book, we’ve broken many coding tasks into easy-to-follow, step-by-step procedures.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filenames, folder names, and code</td>
<td>These things appear in a monospace font.</td>
</tr>
<tr>
<td>Commands</td>
<td>Commands and their syntax use bold.</td>
</tr>
<tr>
<td>Menu commands</td>
<td>We use the following style for all application menu commands: Menu, Command, where Menu is the name of the menu you pull down and Command is the name of the command you select. Here’s an example: File, Open. This means you select the File menu and then select the Open command.</td>
</tr>
</tbody>
</table>

This book also uses the following boxes to draw your attention to important or interesting information:

BY THE WAY

By the Way boxes present asides that give you more information about the current topic. These tidbits provide extra insights that offer better understanding of the task.
DID YOU KNOW?
Did You Know boxes call your attention to suggestions, solutions, or shortcuts that are often hidden, undocumented, or just extra useful.

WATCH OUT!
Watch Out! boxes provide cautions or warnings about actions or mistakes that bring about data loss or other serious consequences.
What You’ll Learn in This Hour:

- How to produce output from a script
- Making a script readable
- How to use variables
- Assigning value to variables
- Types of data
- How to put information into a script

In this hour, you get a chance to learn some Python basics, such as using the `print` function to display output. You will read about using variables and how to assign values to variables, and you will gain an understanding of their data types. By the end of the hour, you will know how to get data into a script by using the `input` function and write your first Python script!

**Producing Python Script Output**

Understanding how to produce output from a Python script is a good starting point for those who are new to the Python programming language. You get instant feedback on your Python statements from the Python interactive interpreter and can experiment with proper syntax. The `print` function, which you met in Hour 3, “Setting Up a Programming Environment,” is a good place to focus your attention.

**Exploring the `print` Function**

A function is a group of Python statements that are put together as a unit to perform a specific task. You simply enter a single Python statement to perform a task for you.
BY THE WAY

The “New” print Function
In Python v2, print is not a function. It became a function when Python v3 was created. This is important to know, in case you are ever tasked with converting a script from v2 to v3.

The print function’s task is to output items. The items to output are correctly called an argument. The basic syntax of the print function is as follows:

\[ \text{print}\left(\text{argument}\right) \]

DID YOU KNOW?

Standard Library of Functions
The print function is called a built-in function because it is part of the Python standard functions library. You don’t need to do anything special to get this function. It is provided for your use when you install Python.

The argument portion of the print function can be characters, such as ABC or 123. It also can be values stored in variables. You learn about variables later in this hour.

Using Characters as print Function Arguments
To display characters (also called string literals) using the print function, you need to enclose the characters in either a set of single quotes or double quotes. Listing 4.1 shows using a pair of single quotes to enclose characters (a sentence) so it can be used as a print function argument.

Listing 4.1 Using a Pair of Single Quotes to Enclose Characters

```python
>>> print('This is an example of using single quotes.')
This is an example of using single quotes.
```

Listing 4.2 shows the use of double quotes with the print function. You can see that the resulting output in both Listing 4.1 and Listing 4.2 does not contain the quotation marks, only the characters.

Listing 4.2 Using a Pair of Double Quotes to Enclose Characters

```python
>>> print("This is an example of using double quotes.")
This is an example of using double quotes.
```
BY THE WAY

Choose One Type of Quotes and Stick with It

If you like to use single quotation marks to enclose string literals in a `print` function argument, then consistently use them. If you prefer double quotation marks, then consistently use them. Even though Python doesn’t care, it is considered poor form to use single quotes on one `print` function argument and then double quotes on the next. Mixing your quotation marks back and forth makes the code harder for humans to read.

Sometimes you need to output a character string that contains a single quote mark to show possession or a contraction. In such a case, you should use double quotes around the `print` function argument, as shown in Listing 4.3.

**LISTING 4.3  Protecting a Single Quote with Double Quotes**

```python
>>> print("This example protects the output's single quote.")
This example protects the output's single quote.

>>> 
```

At other times, you need to output a string of characters that contain double quotes, such as for a quotation. Listing 4.4 shows an example of protecting a quote, using single quotes in the argument.

**LISTING 4.4  Protecting a Double Quote with Single Quotes**

```python
>>> print('I said, "I need to protect my quotation!" and did so.')
I said, "I need to protect my quotation!" and did so.

>>> 
```

DID YOU KNOW?

Protecting Single Quotes with Single Quotes

You also can embed single quotes within single quote marks and double quotes within double quote marks. However, when you do, you need to use something called an escape sequence, which is covered later in this hour.

Formatting Output with the `print` Function

You can perform various output formatting features by using the `print` function. For example, you can insert a single blank line by using the `print` function with no arguments, like this:

```python
print()
```
The screen in Figure 4.1 shows a short Python script that inserts a blank line between two other lines of output.

Another way to format output using the `print` function is via triple quotes. Triple quotes are simply three sets of double quotes (""").

Listing 4.5 shows how to use triple quotes to embed a linefeed character (via pressing the Enter key). When the output is displayed, each embedded linefeed character causes the next sentence to appear on the next line. Thus, the linefeed moves your output to the next new line. Notice that you cannot see the linefeed character embedded on each code line—you can see only its effect in the output.

**LISTING 4.5 Using Triple Quotes**

```python
>>> print("\"\"\"This is line one.\nThis is line two.\nThis is line three.\"\"\")
This is line one.
This is line two.
This is line three.
```

**BY THE WAY**

**But I Prefer Single Quotes**

Triple quotes don’t have to be three sets of double quotes. You can use three sets of single quotes instead to get the same result!
By using triple quotes, you also can protect single and double quotes that need to be displayed in the output. Listing 4.6 shows triple quotes in action to protect both single and double quotes in the same character string.

**LISTING 4.6 Using Triple Quotes to Protect Single and Double Quotes**

```python
>>> print('''Raz said, "I didn't know about triple quotes!" and laughed.''')
Raz said, "I didn't know about triple quotes!" and laughed.
```

**Controlling Output with Escape Sequences**

An *escape sequence* is a character or series of characters that allow a Python statement to escape from normal behavior. The new behavior can be the addition of special formatting for the output or the protection of characters typically used in syntax. Escape sequences all begin with the backslash (\) character.

An example of using an escape sequence to add special formatting for output is the \n escape sequence. The \n escape sequence forces any characters listed after it onto the displayed output’s next line. This escape sequence is called a *newline*, and the formatting character it inserts is a linefeed. Listing 4.7 shows an example of using \n to insert a linefeed. Notice that it causes the output to be formatted exactly as it was in Listing 4.5 using triple quotes.

**LISTING 4.7 Using an Escape Sequence to Add a Linefeed**

```python
>>> print("This is line one.\nThis is line two.\nThis is line three.")
This is line one.
This is line two.
This is line three.
```

Typically, the `print` function puts a linefeed only at the end of displayed output. However, the `print` function in Listing 4.7 is forced to escape its normal formatting behavior because of the \n escape sequence addition.

**DID YOU KNOW?**

**Quotes and Escape Sequences**

Escape sequences work whether you use single quotes, double quotes, or triple quotes to surround your `print` function argument.
You also can use escape sequences to protect various characters used in syntax. Listing 4.8 shows the backslash (\) character used to protect a single quote so that it will not be used in the print function’s syntax. Instead, the quote is displayed in the output.

**LISTING 4.8 Using an Escape Sequence to Protect Quotes**

```python
>>> print('Use backslash, so the single quote isn\'t noticed."
Use backslash, so the single quote isn't noticed.

```  

You can use many different escape sequences in your Python scripts. Table 4.1 shows a few of the available sequences.

**TABLE 4.1 A Few Python Escape Sequences**

<table>
<thead>
<tr>
<th>Escape Sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;</td>
<td>Displays a single quote in output</td>
</tr>
<tr>
<td>&quot;</td>
<td>Displays a double quote in output</td>
</tr>
<tr>
<td>\</td>
<td>Displays a single backslash in output</td>
</tr>
<tr>
<td>\a</td>
<td>Produces a bell sound with output</td>
</tr>
<tr>
<td>\f</td>
<td>Inserts a form feed into the output</td>
</tr>
<tr>
<td>\n</td>
<td>Inserts a linefeed into the output</td>
</tr>
<tr>
<td>\t</td>
<td>Inserts a horizontal tab into the output</td>
</tr>
<tr>
<td>\u####</td>
<td>Displays the Unicode character denoted by the character's four hexadecimal digits (####)</td>
</tr>
</tbody>
</table>

Notice in Table 4.1 that not only can you insert formatting into your output, but you can produce sound as well! Another interesting escape sequence involves displaying Unicode characters in your output.

**Now for Something Fun!**

Thanks to the Unicode escape sequence, you can print all kinds of characters in your output. You learned a little about Unicode in Hour 3, “Setting Up a Programming Environment.” You can display Unicode characters by using the \u escape sequence. Each Unicode character is represented by a hexadecimal number. These hexadecimal numbers are found at www.unicode.org/charts. There are lots of Unicode characters!
The Unicode hexadecimal number for the pi (π) symbol is 03c0. To display this symbol using the Unicode escape sequence, you must precede the number with \u in your print function argument. Listing 4.9 displays the pi symbol to output.

**LISTING 4.9 Using a Unicode Escape Sequence**

```python
>>> print("I love my Raspberry \u03c0!")
I love my Raspberry π!
>>> 
```

**Create Output with the print Function**

This hour you have been reading about creating and formatting output by using the print function. Now it is your turn to try this versatile Python tool. Follow these steps:

1. If you have not already done so, power up your Raspberry Pi and log in to the system.
2. If you do not have the GUI started automatically at boot, start it now by typing startx and pressing Enter.
3. Open a terminal by double-clicking the Terminal icon.
4. At the command-line prompt, type `python3` and press Enter. You are taken to the Python interactive shell, where you can type Python statements and see immediate results.
5. At the Python interactive shell prompt (>>>), type `print('I learned about the print function.')` and press Enter.
6. At the prompt, type `print('I learned about single quotes.')` and press Enter.
7. At the prompt, type `print("Double quotes can also be used.")` and press Enter.

**BY THE WAY**

**Multiple Lines with Triple Double Quotes**

In steps 8–10, you will not be completing the print function on one line. Instead, you will be using triple double quotes to enable multiple lines to be entered and displayed.

8. At the prompt, type `print("***I learned about things like... and press Enter.
9. Type triple quotes, and press Enter.
10. Type and displaying text on multiple lines."***)` and press Enter. Notice that the Python interactive shell did not output the Python print statement’s argument until you had fully completed it with the closing parenthesis.
11. At the prompt, type `print('Single quotes protect "double quotes" in output.')` and press Enter.

12. At the prompt, type `print("Double quotes protect 'single quotes' in output.")` and press Enter.

13. At the prompt, type `print("A backslash protects "double quotes" in output.")` and press Enter.

14. At the prompt, type `print("A backslash protects 'single quotes' in output.")` and press Enter. Using the backslash to protect either single or double quotes enables you to maintain your chosen method of consistently using single (or double) quotes around your `print` function argument.

15. At the prompt, type `print("The backslash character \ is an escape character.")` and press Enter.

16. At the prompt, type `print("Use escape sequences to \n insert a linefeed.")` and press Enter. In the output, notice how part of the sentence, `Use escape sequences to`, is on one line and the end of the sentence, `insert a linefeed.`, is on another line. This is due to your insertion of the escape sequence `\n` in the middle of the sentence.

17. At the prompt, type `print("Use escape sequences to \t\t insert two tabs or" and press Enter.

18. At the ... prompt, type "insert a check mark: \u2714") and press Enter.

You can do a lot with the `print` function to display and format output! In fact, you could spend this entire hour just playing with output formatting. However, there are additional important Python basics you need to learn, such as formatting scripts for readability.

Formatting Scripts for Readability

Just as the development environment, IDLE, will help you as your Python scripts get larger, a few minor practices also will be helpful to you. Learn these tips early on, so they become habits as your Python skills grow (and as the length of your scripts grow!).

Long Print Lines

Occasionally you will have to display a very long output line using the `print` function, such as a paragraph of instructions for the script user. The problem with long output lines is that they make your script code hard to read and the logic behind the script harder to follow. Python is supposed to “fit in your brain.” The habit of breaking up long output lines will help you meet that goal. There are a couple of ways you can accomplish this.
BY THE WAY

A Script User?
You might be one of those people who have never heard the term *user* in association with computers. A *user* is a person who is using the computer or running the script. Sometimes the term *end user* is used instead. You should always keep the user in mind when you write your scripts, even if the user is just you!

The first way to break up a long output character line is to use something called string concatenation. *String concatenation* takes two or more strings of text and “glues” them together, so they become one text string. The “glue” in this method is the plus (+) symbol. However, to get this to work properly, you also need to use the backslash (\) to escape out of the print function’s normal behavior—putting a linefeed at a character string’s end. Thus, the two items you need are +\, as shown in Listing 4.10.

**LISTING 4.10** String Concatenation for Long Text Lines

```python
>>> print("This is a really long line of text " +\n... "that I need to display!")
This is a really long line of text that I need to display!
```

As Listing 4.10 shows, the two strings are concatenated and displayed as one string in the output. However, there is an even simpler and cleaner method of accomplishing this. You can forgo the +\ and simply keep each character string in its own sets of quotation marks. The character strings will be automatically concatenated by the print function! The print function handles this perfectly and is a lot cleaner looking. This method is demonstrated in Listing 4.11.

**LISTING 4.11** Combining for Long Text Lines

```python
>>> print("This is a really long line of text "
... "that I need to display!")
This is a really long line of text that I need to display!
```

It is always a good rule to keep your Python syntax simple to provide better script readability. However, sometimes you need to use complex syntax. This is where comments will help you. No, not comments spoken aloud, like “I think this syntax is complicated!” We’re talking about comments that are embedded in your Python script.
Creating Comments

In scripts, comments are notes from the Python script author. A comment’s purpose is to provide understanding of the script’s syntax and logic. The Python interpreter ignores any comments. However, comments are invaluable to humans who need to modify or debug scripts.

DID YOU KNOW?

Standard of Good Form

If you are serious about Python programming, it’s important that you consistently have good form in your code. The good form standard is the Style Guide for Python Code located at https://www.python.org/dev/peps/pep-0008/.

To add a comment to a script, you precede it with the pound or hash symbol (#). The Python interpreter ignores anything that follows the hash symbol.

For example, when you write a Python script, it is a good idea to insert comments that include your name, when you wrote the script, and the script’s purpose. Figure 4.2 shows an example. Some script writers believe in putting these comments at their script’s top, while others put them at the bottom. At the very least, if you include a comment with your name as the author in your script, when the script is shared with others, you will get credit for its writing.

![Figure 4.2](image_url)

**FIGURE 4.2**

Comments in a Python script.

You also can provide clarity by breaking up sections of your scripts using long lines of the # symbol. Figure 4.2 shows a long line of hash symbols used to separate the comment section from the main body of the script.

Finally, you can put comments at the end of a Python statement. Notice in Figure 4.2 that the `print()` statement is followed by the comment `# Inserts a blank line in output.` A comment placed at the statement’s end is called an end comment, and it provides clarity about that particular code line.
Those few simple tips will help improve your code’s readability. Putting these tips into practice will save you time as you write and modify Python scripts.

Understanding Python Variables

A variable is a name that stores a value for later use in a script. A variable is like a coffee cup. A coffee cup typically holds coffee, of course! But a coffee cup also can hold tea, water, milk, rocks, gravel, sand...you get the picture. Think of a variable as an object holder that you can look at and use in your Python scripts.

BY THE WAY

An Object Reference

Python really doesn’t have variables. Instead, they are object references. However, for now, just think of them as variables.

When you name your coffee cup...err, variable, you need to be aware that Python variable names are case sensitive. For example, the variables named CoffeeCup and coffeecup are two different variables. Other rules are associated with creating Python variable names, as well:

- You cannot use a Python keyword as a variable name.
- The first character of a variable name cannot be a number.
- No spaces are allowed in a variable name.

Python Keywords

The Python keywords list changes every so often. Therefore, it is a good idea to take a look at the current keywords list before you start creating variable names. To look at the keywords, you need to use a standard library function. However, this function is not built in, like the print function is. You have this function on your Raspbian system, but before you can use it, you need to import the function into Python. (You’ll learn more about importing a function in Hour 13, “Working with Modules.”) The function’s name is keyword.kwlist. Listing 4.12 shows you how to import into Python and determine keywords.

LISTING 4.12   Determining Python Keywords

```python
>>> import keyword
>>> print(keyword.kwlist)
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except',
 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'raise', 'return', 'try', 'while', 'with', 'yield']
```
In Listing 4.12, the command import keyword brings the keyword functions into the Python interpreter so they can be used. Then the statement print(keyword.kwlist) uses the keyword.kwlist and print functions to display the current list of Python keywords. These keywords cannot be used as Python variable names.

Creating Python Variable Names

For the first character in your Python variable name, you must not use a number. The first character in the variable name can be any of the following:

- A letter a–z
- A letter A–Z
- The underscore character (_)

After the first character in a variable name, the other characters can be any of the following:

- The numbers 0–9
- The letters a–z
- The letters A–Z
- The underscore character (_)

DID YOU KNOW?

Using Underscore for Spaces

Because you cannot use spaces in a variable’s name, you should use underscores in their place to make your variable names readable. For example, instead of creating a variable name like coffeecup, use the variable name coffee_cup.

After you determine a name for a variable, you still cannot use it. A variable must have a value assigned to it before it can be used in a Python script.
Assigning Value to Python Variables

Assigning a value to a Python variable is fairly straightforward. You put the variable name first, then an equal sign (=), and finish up with the value you are assigning to the variable. This is the syntax:

```
variable=value
```

Listing 4.13 creates the variable `coffee_cup` and assigns a value to it.

**Listing 4.13  Assigning a Value to a Python Variable**

```python
>>> coffee_cup='coffee'
>>> print(coffee_cup)
coffee
```

As Listing 4.13 shows, the `print` function can output the variable’s value without any quotation marks around it. You can take output a step further by putting a string and a variable together as two `print` function arguments. The `print` function knows they are two distinct arguments because they are separated by a comma (,), as shown in Listing 4.14.

**Listing 4.14  Displaying Text and a Variable**

```python
>>> print("My coffee cup is full of", coffee_cup)
My coffee cup is full of coffee
```

Formatting Variable and String Output

Using variables brings additional formatting issues. For example, the `print` function automatically inserts a space whenever it encounters a comma (,) in a statement. This is why you do not need to add a space at the `My coffee cup is full of string`’s end, as shown in Listing 4.14. Sometimes, however, you might want something else besides a space to separate a character string from a variable in the output. In such a case, you can use a separator in your statement. Listing 4.15 uses the `sep` separator to place an asterisk (*) in the output instead of a space.

**Listing 4.15  Using Separators in Output**

```python
>>> coffee_cup='coffee'
>>> print("I love my", coffee_cup, "!", sep='*')
I love my*coffee*!
```
Notice you also can put variables between various strings in your `print` statements. In Listing 4.15, four arguments are given to the `print` function:

- The string "I love my"
- The variable `coffee_cup`
- The string "!
- The separator designation '\*'

The variable `coffee_cup` is between two strings. Thus, you get two asterisks (*), one between each argument to the `print` function. Mixing strings and variables in the `print` function gives you a lot of flexibility in your script's output.

**BY THE WAY**

**At the End**

Using the `end` keyword instead of `sep` allows you to tack on characters (and/or one of the escape sequences in Table 4.1) at a `print` statement's end. For example, you could tack on an exclamation mark and a linefeed using this statement: `print("I love my", coffee_cup, end='!/n')`

---

**Avoiding Unassigned Variables**

You cannot use a variable until you have assigned a value to it. A variable is created when it is assigned a value and not before. Listing 4.16 shows an example of this.

**LISTING 4.16  Behavior of an Unassigned Variable**

```python
>>> print(glass)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'glass' is not defined

>>> glass='water'
>>> print(glass)
water
>>> 
```

When the first `print(glass)` statement was issued in Listing 4.16, the `glass` variable had not been given a value. Thus, the Python interpreter delivered an error message. Before the second time the `print(glass)` statement was issued, the `glass` variable was assigned the character"
Assigning Value to Python Variables

string, water. Therefore, the `glass` variable was created and no error message was delivered for the second `print(glass)` statement.

Assigning Long String Values to Variables

If you need to assign a long string value to a variable, you can break it up onto multiple lines by using a couple methods. Earlier in the hour, in the “Formatting Scripts for Readability” section, you looked at using the `print` function with multiple lines of outputted text. The concept here is similar.

The first method involves using string concatenation (+) to put the strings together and an escape character (\) to keep a linefeed from being inserted. Listing 4.17 shows that two long lines of text were concatenated together in the `long_string` variable assignment.

**LISTING 4.17  Concatenating Text in Variable Assignment**

```python
>>> long_string="This is a really long line of text" +
... " that I need to display!"
>>> print(long_string)
This is a really long line of text that I need to display!

Another method is to use parentheses to enclose your variable’s value. Listing 4.18 eliminates the +\ and uses parentheses () on either side of the entire long string. This makes the value into a single long character string in output.

**LISTING 4.18  Combining Text in Variable Assignment**

```python
>>> long_string=("This is a really long line of text" 
... " that I need to display!")
>>> print(long_string)
This is a really long line of text that I need to display!

The method used in Listing 4.18 is a much cleaner method. It also helps improve the script’s readability.

**BY THE WAY**

Assigning Short Strings to Variables

You can use parentheses for assigning short strings to variables, too! This is especially useful and may also improve the readability of your Python script.
More Variable Assignments

A variable’s value does not have to be only a character string—it also can be a number. In Listing 4.19, the amount of coffee consumed is assigned to the variable `cups_consumed`.

**LISTING 4.19 Assigning a Numeric Value to a Variable**

```python
>>> coffee_cup='coffee'
>>> cups_consumed=3
>>> print("I had", cups_consumed, "cups of", coffee_cup, "today!")
I had 3 cups of coffee today!
```  

You also can assign an expression’s result to a variable. The equation `3+1` is calculated in Listing 4.20, and the resulting value `4` is assigned to the variable `cups_consumed`.

**LISTING 4.20 Assigning an Expression Result to a Variable**

```python
>>> coffee_cup='coffee'
>>> cups_consumed=3 + 1
>>> print("I had", cups_consumed, "cups of", coffee_cup, "today!")
I had 4 cups of coffee today!
```  

You learn more about performing mathematical operations within Python scripts in Hour 5, “Using Arithmetic in Your Programs.”

Reassigning Values to a Variable

After you assign a value to a variable, the variable is not stuck with that value. It can be reassigned. Variables are called variables because their values can be varied. (Say that three times fast!)

In Listing 4.21, the variable `coffee_cup` has its value changed from coffee to tea. To reassign a value, you simply enter the assignment syntax with a new value at its end.

**LISTING 4.21 Reassigning a Variable**

```python
>>> coffee_cup='coffee'
>>> print("My cup is full of", coffee_cup)
My cup is full of coffee
>>> coffee_cup='tea'
>>> print("My cup is full of", coffee_cup)
My cup is full of tea
```
DID YOU KNOW?

**Variable Name Case**

Python script writers tend to use all lowercase letters in the names of variables whose values might change, such as `coffee_cup`. For variable names that are never reassigned values, all uppercase letters are used (for example, `PI=3.14159`). These unchanging variables are called *symbolic constants*.

---

**Learning About Python Data Types**

When a variable is created by an assignment such as `variable=value`, Python determines and assigns a data type to the variable. A *data type* defines how the variable is stored and the rules governing how the data can be manipulated. Python uses the variable’s assigned value to determine its type.

So far, this hour has focused on character strings. When the Python statement `coffee_cup='tea'` was entered, Python saw the characters in quotation marks and determined the variable `coffee_cup` to be a *string literal* data type, or *str*. Table 4.2 lists a few of the basic data types Python assigns to variables.

**TABLE 4.2 Python Basic Data Types**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>float</td>
<td>Floating-point number</td>
</tr>
<tr>
<td>int</td>
<td>Integer</td>
</tr>
<tr>
<td>long</td>
<td>Long integer</td>
</tr>
<tr>
<td>str</td>
<td>Character string or string literal</td>
</tr>
</tbody>
</table>

You can determine which data type Python has assigned to a variable by using the `type` function. In Listing 4.22, the variables have been assigned two different data types.

**LISTING 4.22 Assigned Data Types for Variables**

```python
>>> coffee_cup='coffee'
>>> type(coffee_cup)
<class 'str'>
>>> cups_consumed=3
>>> type(cups_consumed)
<class 'int'>
```
Python assigned the data type `str` to the variable `coffee_cup` because it saw a string of characters between quotation marks. However, for the `cups_consumed` variable, Python saw a whole number, and thus it assigned the integer data type, `int`.

**DID YOU KNOW?**

**The `print` Function and Data Types**

The `print` function assigns to its arguments the string literal data type `str`. It does this for anything that is given as an argument, such as quoted characters, numbers, variables values, and so on. Thus, you can mix data types in your `print` function argument. The `print` function will evaluate any variables, convert everything to a string literal data type, and spit it out to the display.

Making a small change in the `cups_consumed` variable assignment statement causes Python to change its data type. In Listing 4.23, the number assigned to `cups_consumed` is reassigned from 3 to 3.5. This causes Python to reassign the data type to `cups_consumed` from `int` to `float`.

**LISTING 4.23  Changed Data Types for Variables**

```python
>>> cups_consumed=3
>>> type(cups_consumed)
<class 'int'>
>>> cups_consumed=3.5
>>> type(cups_consumed)
<class 'float'>
```

You can see that Python does a lot of the “dirty work” for you. This is one of the many reasons Python is so popular.

**Allowing Python Script Input**

Sometimes you might need a script user to provide data into your script from the keyboard. To accomplish this task, Python provides the `input` function. The `input` function is a built-in function and has the following syntax:

```
variable=input(user prompt)
```

In Listing 4.24, the variable `cups_consumed` is assigned the value returned by the `input` function. The script user is prompted to provide this information. An `input` function argument designates the prompt provided to the user. The script user types an answer and presses the Enter key. This action causes the `input` function to assign the answer 3 as a value to the variable `cups_consumed`. 
Allowing Python Script Input

LISTING 4.24  Variable Assignment via Script Input

```python
>>> cups_consumed=input("How many cups did you drink? ")
How many cups did you drink? 3
>>> print("You drank", cups_consumed, "cups!")
You drank 3 cups!
```  

For the user prompt, you can enclose the prompt’s string characters in either single or double quotes. The prompt is shown enclosed in double quotes in Listing 4.24’s `input` function.

BY THE WAY

Be Nice to Your Script User

Be nice to the user of your script, even if it is just yourself. It is no fun typing an answer that is “squished” up against the prompt. Add a space at the end of each prompt to give the end user a little breathing room for prompt answers. Notice in the `input` function in Listing 4.24 that a space is added between the question mark (?) and the enclosing double quotes.

The `input` function treats all input as strings. This is different from how Python handles other variable assignments. Remember that if `cups_consumed=3` were in your Python script, it would be assigned the data type integer, `int`. When using the `input` function, as shown in Listing 4.25, the data type is set to string, `str`.

LISTING 4.25  Data Type Assignments via Input

```python
>>> cups_consumed=3
>>> type(cups_consumed)
<class 'int'>
>>> cups_consumed=input("How many cups did you drink? ")
How many cups did you drink? 3
>>> type(cups_consumed)
<class 'str'>
```  

To convert variables (input from the keyboard) from strings, you can use the `int` function. The `int` function will convert a number from a string data type to an integer data type. You can use the `float` function to convert a number from a string to a floating-point data type. Listing 4.26 shows how to convert the variable `cups_consumed` to an integer data type.

LISTING 4.26  Data Type Conversion via the `int` Function

```python
>>> cups_consumed=input("How many cups did you drink? ")
How many cups did you drink? 3
```
You can get really tricky here and use a nested function. Nested functions are functions within functions. The general format is as follows:

```
variable = functionA(functionB())
```

Listing 4.27 uses this method to properly change the input data type from a string to an integer.

**LISTING 4.27 Using Nested Functions with input**

```python
>>> cups_consumed = int(input("How many cups did you drink? ")
How many cups did you drink? 3
```

Using nested functions makes a Python script more concise. However, the trade-off is that the script is a little harder to read.

**TRY IT YOURSELF**

**Explore Python Input and Output with Variables**

You are now going to explore Python input and output using variables. In the following steps, you write a script to play with, instead of using the interactive Python shell:

1. If you have not already done so, power up your Raspberry Pi and log in to the system.
2. If you do not have the GUI started automatically at boot, start it now by typing `startx` and pressing Enter.
3. Open the Terminal by double-clicking the Terminal icon.
4. If you want to follow along with the book, you will need to create a directory to hold your Python scripts. At the command-line prompt, type `mkdir py3prog` and press Enter.
5. At the command-line prompt, type `nano py3prog/script0401.py` and press Enter. The command puts you into the nano text editor and creates the file `py3prog/script0401.py`.
6. Type the following code into the nano editor window, pressing Enter at the end of each line:

```python
# My first real Python script.
# Written by <your name here>
#
############ Define Variables ###########
#
amount=4                  # Number of vessels.
vessels='glasses'         # Type of vessels used.
liquid='water'            # What is contained in the vessels.
location='on the table'   # Location of vessels.
#
############ Output Variable Description ###########################
#
print("This script has four variables pre-defined in it.")
print()
#
print("The variables are as follows:")
#
print("name: amount", "data type: ", type(amount), "value: ", amount)
#
print("name: vessels", "data type: ", type(vessels), "value: ", vessels)
#
print("name: liquid", "data type: ", type(liquid), "value: ", liquid)
#
print("name: location", "data type: ", type(location), "value: ", location)
print()
#
############ Output Sentence Using Variables #
#
print("There are", amount, vessels, "full of", liquid, location, end='\n')
print()
```

BY THE WAY

Be Careful!

Be sure to take your time here and avoid making typographical errors. Double-check and make sure you have entered the code into the nano text editor window as shown here. You can make corrections by using the Delete key and the up- and down-arrow keys.

7. Write out the information you just typed in the text editor to the script by pressing Ctrl+O. The script filename will show along with the prompt filename to write. Press Enter to write out the contents to the `script0401.py` script.

8. Exit the nano text editor by pressing Ctrl+X.
9. Type `python3 py3prog/script0401.py` and press Enter to run the script. If you encounter any errors, note them so you can fix them in the next step. You should see output like the output shown in Figure 4.3. The output is okay, but it's a little sloppy. You can clean it up in the next step.

```
pi@raspberrypi:$ python3 py3prog/script0401.py
This script has four variables pre-defined in it.
The variables are as follows:
name: amount data type: <class 'int'> value: 4
name: vessels data type: <class 'str'> value: glasses
name: liquid data type: <class 'str'> value: water
name: location data type: <class 'str'> value: on the table
There are 4 glasses full of water on the table.
pi@raspberrypi:$
```

**FIGURE 4.3**
Output for the Python script `script0401.py`.

10. At the command-line prompt, type `nano py3prog/script0401.py` and press Enter. The command puts you into the nano text editor, where you can modify the `script0401.py` script.

11. Go to the Output Variable Description portion of the script and add a separator to the end of each line. The lines of code to be changed; how they should look when you are done is shown here:

```python
print("name: amount", "data type: ", type(amount), "value: ", amount, sep='\t')
#
print("name: vessels", "data type: ", type(vessels), "value: ", vessels, sep='\t')
#
print("name: liquid", "data type: ", type(liquid), "value: ", liquid, sep='\t')
#
print("name: location","data type: ",type(location), "value: ",location,sep='\t')
```

12. Write out the modified script by pressing Ctrl+O. **Don't** press Enter yet! Change the filename to `script0402.py` and then press Enter. When nano asks Save file under DIFFERENT NAME ?, type `Y` and press Enter.

13. Exit the nano text editor by pressing Ctrl+X.

14. Type `python3 py3prog/script0402.py` and press Enter to run the script. You should see output like the output shown in Figure 4.4. Much neater!
15. To try adding some input into your script, at the command-line prompt, type `nano py3prog/script0402.py` and press Enter.

16. Go to the bottom of the script and add the Python statements shown here:

```python
# Get Input
print()
print("Now you may change the variables' values. ")
print()
amount=int(input("How many vessels are there? "))
print()

vessels = input("What type of vessels are being used? ")
print()

liquid = input("What type of liquid is in the vessel? ")
print()

location=input("Where are the vessels located? ")
print()

# Display New Input to Output
print("So you believe there are", amount, vessels, "of", liquid, location, end='.
"
print()

# End of Script
```

17. Write out the modified script by pressing Ctrl+O. *Don't* press Enter yet! Change the filename to `script0403.py` and then press Enter. When nano asks *Save file under DIFFERENT NAME ?*, type `Y` and press Enter.
18. Exit the nano text editor by pressing Ctrl+X.

19. Type `python3 py3prog/script0403.py` and press Enter to run the script. Answer the prompts any way you want. (You are supposed to be having fun here!) Figure 4.5 shows what your output should look like.

![Figure 4.5](image)

Run this script as many times as you want. Experiment with the various types of answers you enter and see what the results are. Also try making some minor modifications to the script and see what happens. Experimenting and playing with your Python script will enhance your learning.

**Summary**

In this hour, you got an overview of Python basics. You learned about output and formatting output from Python, creating legal variable names and assigning values to variables, and about various data types and when they are assigned by Python. You explored how Python can handle input from the keyboard and how to convert the data types of the variables receiving that input. Finally, you got to play with your first Python script. In Hour 5, your Python exploration will continue as you delve into mathematical algorithms with Python.
Q&A

Q. Can I do any other kind of output formatting besides what I learned about in this chapter?
A. Yes, you can also use the `format` function, which is covered in Hour 5.

Q. Which is better to use with a `print` function, double quotes or single quotes?
A. Neither one is better than the other. Which one you use is a personal preference. However, whichever one you choose, it’s best to consistently stick with it.

Q. Bottles of tea on the wall?!
A. This is a family-friendly tutorial. Feel free to modify your answers to script0403.py to your liking.

Workshop

Quiz

1. The `print` function is part of the Python standard library and is considered a built-in function. True or false?

2. When is a variable created and assigned a data type?

3. A(n) ________ sequence enables a Python statement to “escape” from its normal behavior.

4. Which of the following is a valid Python escape sequence?
   a. //</b
   b. \'
   c. ESC

5. Which Python escape sequence will insert a linefeed in output?

6. A comment in a Python script should begin with which character?

7. Which of the following is a valid Python data type?
   a. int
   b. input
   c. print

8. Which function enables you to view a variable’s data type?
9. If a variable is assigned the number 3.14, which data type will it be assigned?

10. The `input` function is part of the Python standard library and is considered a built-in function. True or false?

**Answers**

1. True. The `print` function is a built-in function of the standard library. There is no need to import it.

2. A variable is created and assigned a data type when it is assigned a value. The value and data type for a variable can be changed with a reassignment.

3. An escape sequence enables Python statement to “escape” from its normal behavior.

4. b. `
` is a valid Python escape sequence. Refer to Table 4.1 for a few valid Python escape sequences.

5. The `\n` Python escape sequence will insert a linefeed in output.

6. A comment in a Python script should begin with the pound or hash symbol (#) for the Python interpreter to ignore it.

7. a. `int` is a Python data type. `input` and `print` are both Python functions. Refer to Table 4.2 for a refresher on the data types.

8. The `type` function enables you to view a variable’s data type.

9. If a variable is assigned the number 3.14, it will be assigned the `float` data type by Python. Refer to Table 4.2 for data types.

10. True. The `input` function is a built-in function of the standard library. There is no need to import it.
Symbols

* (asterisk) in regular expressions, 342-343
() (braces) in regular expressions, 344
, (comma), comma-separated text files, 225
< comparison operator (Python scripts), 124
<= comparison operator (Python scripts), 124
> comparison operator (Python scripts), 124
== (double equal signs) in Python scripts, 118
| (pipe symbol) in regular expressions, 344-345
+ (plus sign) in regular expressions, 344
? (question mark) in regular expressions, 343
" (double quotes)
  displaying via print function, 74-75
  formatting via print function, 76-77
' (single quotes)
  displaying via print function, 74-75

formatting via print function, 76-77

*** (triple quotes), formatting via print function, 76-77

A

absolute directory references
  (Linux directory structure), 226-227, 232
accessor methods (OOP classes), 295-297
Allied Electronics, Inc. website, 9
Amazon.com website, 29
anchor characters (regular expressions), 337-339
Android phones, 29
Apache web server, 475-476
  CGI programming, 478-479
    creating Python programs, 479-480
    defining, 479
    running Python programs, 479
files and folders, 476
HTML files, serving, 477-478
installing, 476-477
web forms, 488
cgi module, 491-493
creating, 488-490
HTML elements, 488-489
webpages, publishing, 478
arguments, passing to functions, 254-256
default parameter values, setting, 256-257
variable numbers of arguments, 258-259
arithmetic in Python scripts
complex numbers, 107-108
fractions, 105-106
imaginary numbers, 107
math module, 108-112
math operators, 99-105
NumPy math libraries, 112-114
arrays in NumPy math libraries, 113-114
ASCII, Python v3, 207-208
asterisk (*) in regular expressions, 342-343
asynchat module (network programming), 427
asynchronous events, GPIO interface input, 551-553
asyncore module (network programming), 427
attributes (OOP classes), 292
default values, 293-294
defining, 293-294
private attributes, 295

B

binary files, 225
blank passwords, 32

Blender3D game library, 398
Boolean comparisons (Python scripts), 128
booting straight to GUI, 37
braces ( { } ) in regular expressions, 344
break statements (Python scripts), 151
bus-powered USB hubs, 18
Button widget (GUI programming), 374, 384-385
buying
peripherals
cases, 16-17
determining necessary peripherals, 10
keyboards, 14-15
kits (prepackaged), 18
MicroSD cards, 10-12
mouses (mice), 14-15
network cables, 15
output displays, 14
portable power supplies, 17
power supplies, 12-13
USB hubs, 18
Wi-Fi adapters, 15
Raspberry Pi retailers, 9-10
tips for, 8-9-10
.bzip2 files, 225

network cables, buying, 15
Pi Cobbler ribbon cable, 537
power supplies, 12
troubleshooting connections, 24
calendar command, 33
cases
buying, 16-17
Raspberry Pi 1 Model B, 17
Raspberry Pi 2 Model B, 16
static electricity, 17
cat command, 31
cd command, 31
centering HD images, 507-508
CGI (Common Gateway Interface) programming and Apache web server, 478-479
defining, 479
Python programs
creating Python programs, 479-480
debugging, 486-488
running, 479
web forms, 491-493
cgi module
network programming, 427
web programming, 491-493
Checkbutton widget (GUI programming), 374, 385-387
checksums
defining, 20
NOOBS installation software, 559-560
downloading, 20-21
Linux checksums, 560
mismatched checksums, 561
OS X checksums, 560
Windows checksums, 560

cables
connections, troubleshooting, 24
HDMI cables, new Raspberry Pi setups, 22
circuits boards and static electricity, 17

classes (OOP), 292
  attributes, 292
    default values, 293-294
    defining, 293-294
    private attributes, 295
    defining, 292
destructors, 299-300
documenting, 300-301
duplication in, 307-308
instances
  creating, 293
deleting, 299-300
methods, 292, 294
  accessor methods, 295-297
  constructors, 297-299
  customizing output, 299
  helper methods, 297-302
  mutator methods, 294-295
  property() helper method, 301-302
modules
  creating class modules, 302-304
  sharing code with, 302-304
  problem with, 307-308
  subclasses and inheritance, 308-316
client programs (socket programming), 446-449

closing files, 239-240
cocos2d game library, 398
command-line
  LXTerminal command-line interface (LXDE GUI), 39
  Raspbian OS
    basic commands, 31
directory-related commands, 33
  entering commands, 31-33
file-related commands, 33
data types
  for loops, assigning data types from lists, 141-142
MySQL database, 458
NumPy math libraries, 112
Python scripts, 89-92
databases
  MySQL database, 453
    creating databases, 455-456
    creating Python scripts, 460-464
    creating tables, 457-459
    creating user accounts, 456-457
data types, 458
database connections, 461
database security, 461
downloading Debian packages, 460
inserting data, 461-463
installing, 454
installing Python MySQL/Connector module, 459-460
installing Python PostgreSQL module, 469
primary key data constraints, 463
querying data, 463-464
root user accounts, 454
setting up, 454-459
PostgreSQL database, 464
  creating databases, 465-466
  creating tables, 467-469
  creating user accounts, 466-467
databases

- database connections, 469-470
- formatting data, 470
- inserting data, 470-471
- installing, 464
- psycopg2 module, 469-472
- querying data, 471-472
- setting up, 464-469
- security, 461-485

Debian
- online resources, 30
- packages, downloading in MySQL database, 460
- Raspbian OS distribution, 29-30

debugging Python programs, 486-488

destructors (OOP classes), 299-300

development environments (IDE), 53, 57
- IDLE development environment shell, 57-58
  - grouping statements, 119
  - if statements, 117-119
  - interactive mode, 59-60
  - scripting in, 60-66
- Komodo IDE development environment shell, 57
- PyCharm development environment shell, 57
- PyDev Open Source Plug-In for Eclipse, 57

dictionaries (Python), 180
- creating, 180
- defining, 179-180
- management operations, 185-186
- obtaining data from, 182-184
- populating, 180-181
- programming, 186-192
- retrieving values from dictionaries for functions, 259-260
- updating, 184-185

directories
- command-line commands, 33
  - opening, 231
  - troubleshooting, 231
- Linux directory structure, 226
  - absolute directory references, 226-227, 232
  - relative directory references, 226-227
  - top root directory, 226
- modules
  - creating in test directories, 278-279
  - moving to production directories, 280-284
  - opening files, 231
  - Python directories, 227
  - scripts, displaying, 227
  - troubleshooting files, 231

displays (output)
- buying, 14
- DVI, 14
- HDMI, 14
- NTSC color encoding, 25
- PAL color encoding, 25
- RCA connectors, 14
- troubleshooting, 25
- VGA, 14

documenting classes (OOP), 300-301

dot character (regular expressions), 339-340

double equal signs (==) in Python scripts, 118, 126

double quotes (") in Python scripts, 74-77

downloading NOOBS installation software, 19-21

DVI output displays, 14

dynamic webpages, 482-485

E

electricity (static)
  - cases, 17
  - circuit boards, 17

element14.com website, 9
elif statements (Python scripts), 123-126
else statements (Python scripts), 121-123
email module (network programming), 427
email servers and network programming, 428-429
  - Gmail security, 436
  - Linux modular email environment, 429-431
  - Postfix, 430
  - remote email servers, 432
  - sending email messages example of, 433-435
  - Gmail security, 436
  - to multiple recipients, 436
  - smtplib module, 431-433
  - sendmail, 430
  - smtplib module, 430-431
  - class methods of, 431
classes of, 431
sending email messages, 431-433
Entry widget (GUI programming), 374, 387-388
equal signs (=) in Python scripts, 118, 126
error exceptions (Python scripts)
defining, 351
exception groups, 362-364
handling
generic exemptions, 364
multiple exceptions, 358-370
try except statement, 356-358, 361-370
runtime error exceptions, 354-356
syntactical error exceptions, 351-353
escape sequences in Python scripts, 77-80
event-driven GUI programming, 374-375, 382-384
exception handling
exception groups, 362-364
multiple exception handling, 358-361
exception groups, 362-364
generic exemptions, 364
try except statement blocks, 361-370
try except statement options, 364-365
try except statement, 356-358
statement blocks, 361-370
statement options, 364-365

F
fifengine game library, 398
files, 225
binary files, 225
.bz2 files, 225
closing, 239-240
command-line commands, 33
comma-separated text files, 225
compressed files, 225
creating, 240-241
.gz files, 225
Linux directory structure, 226
absolute directory references, 226-227, 232
relative directory references, 226-227
top root directory, 226
managing via os function, 227-229
numeric files, 225
opening, 237-240
absolute directory references, 232
designating open mode, 230-231
determining file attributes, 231-232
file object methods, 231-232
open function, 229-230
troubleshooting, 231
Python directories, 227
reading, 233, 237-239
entire files, 233-234
line-by-line, 234-235
nonsequentially, 236-237
stripping newline characters from scripts, 235
string files, 225
types of files unable to be processed by Python, 225-226
writing, 240-245
numbers as strings, 242
preexisting files, 243-244
write mode removals, 241
XML files, 225
.xz files, 225
.zip files, 225
findall() function (regular expressions), 333-335
finding modules, 272-273
finditer() function (regular expressions), 333-335
flapping, GPIO interface input, 548-549
floating point accuracy (Python math operators), 103-104
for loops (Python scripts), 137
data types, assigning from lists, 141-142
indentation in, 138
iterating
character strings in lists, 142-143
iterating using range function, 143-146
iterating using variables, 143
numeric values in lists, 138-140
operation of, 138
syntax of, 138
troubleshooting, 140-141
validating user input via, 146-148
formatting
MicroSD cards, 562-566
webpage data, 480-482
fractions in Python scripts, 105-106
frame templates (GUI programming), 378-379
Frame widget (GUI programming), 374
framing HD images, 508
ftpplib module (network programming), 427
functions, 249
creating, 250
defining, 250-251
redefining functions, 252
using functions before they are defined, 251-252
lists and, 263-264
modules
determining how to use functions within, 274-276
gathering functions for custom modules, 278
listing functions in modules, 274
role of functions in modules, 269
passing values to, 254
passing arguments, 254-256
setting default parameter values, 256-257
variable numbers of arguments, 258-259
recursion and, 264-265
retrieving values via dictionaries, 259-260
returning values from, 253-254
using in scripts, 250-252
variables
  global variables, 260-263
  local variables, 260-261
G

game programming, 397-399
  Blender3D game library, 398
cocos2d game library, 398
developers versus designers, 398
fifengine game library, 398
game screen
  interacting with graphics on screen, 415-416
  moving graphics on screen, 414-423
  setting up, 403-409
image handling, 410-413, 416-423
kivy game library, 398
Panda3D game library, 398
playing games online, 399
PyGame game library, 398, 409
  checking for, 400
  events, 409
  game loops, 409
  game screen setup, 403-409
  image handling, 410-413, 416-423
  initializing, 402-403
  installing, 399-400
  interacting with graphics on screen, 415-416
  loading, 402-403
  modules, 401-402
  moving graphics on screen, 414-423
  object classes, 402
  setting up, 399-400
  sound design, 413-414
  sprites, 402
Pyglet game library, 398
PySoY game library, 398
Python-Ogre game library, 398
SDL, 399
sound design, 413-414
generic exemption handling, 364
Gertboard
  GPIO interface
    connections, 537-539
detecting input, 548
  setting up Gertboard for output, 543
  pin block layout, 539
global variables and functions, 260-263
Gmail security, 436
GNOME GUI, 36
gopherlib module (network programming), 427
GPIO interface, 533
  components of, 533-534
  connections, 536
  Gertboard, 537-539
  Pi Cobbler, 536-537
  input detection, 546
    asynchronous events, 551-553
    flapping, 548-549
  Gertboard setup, 548
  input polling, 549-551
  Pi Cobbler setup, 547-548
  pin setup, 548
  pull-ups/downs, 549
  switch bounce, 553
  synchronous events, 551
LED light, 544-546
output, 541
  Gertboard setup, 543
  Pi Cobbler setup, 541-543
  testing, 543-544
IDE (Integrated Development Environments) 579

pins
Gertboard pin block layout, 539
input detection setup, 548
layout of, 534
referring, 540-541
signals versus, 536
resetting, 544
RPi.GPIO module, 539
installing, 539-540
startup methods, 540-541

grouping
comparisons in Python scripts, 130-131
modules, 271
regular expressions, 345
statements in Python scripts, 119-121

GUI (Graphical User Interface)
accessing, 35
booting straight to, 37
GNOME GUI, 36
KDE GUI, 36
LXDE GUI, 35-36
desktop area, 36-37
LXPanel area, 36-43
programming, 373
creating a GUI program, 392-395
event-driven programming, 374-375
frame templates, 378-379
packages, 375
PyGTK GUI package, 375
PyQT GUI package, 375
tkinter GUI package, 375-395
widgets, 374, 378-382, 384-395

window interface, 374, 376-382
wxPython GUI package, 375
Xfce GUI, 36

.gzip files, 225

H
Halfacree, Gareth, 19
HD (High Definition) images
centering, 507-508
converting, 512-513
defining, 498-500
delays, removing, 511
finding, 501-502
framing, 508
functions, loading instead of
modules, 511-512
image presentation script, 500
megapixels, 498
modifying, 516-517
mouse/keyboard controls, 514
movies, 502
music, playing music with,
525-530
optimized presentations, 514-516
performance, improving, 510-516
preloading, 513
presentation screen setup, 500-501
removable drives, storing on,
502-505
scaling, 505-507
screen buffering, 512
testing, 508-510
title screens, 513-514
HDMI (High-Definition Multimedia Interface)
cables and new Raspberry Pi
setups, 22
output displays, buying, 14
ports, 497
helper methods (OOP classes), 297-302
HTML (Hypertext Markup Language)
Apache web server, HTML
files in, 477-478
web forms, 488-489
webpages, formatting data, 480-482
HTTP (Hypertext Transfer Protocol)
lighttp, 475
Monkey HTTP, 475
httplib module (network
programming), 427
hubs (USB)
buying, 18
self-powered USB hubs, 18
hyperbolic functions (math
module), 111

I
IBM Watson, 29
IDE (Integrated Development
Environments), 53, 57
IDELE development
environment shell, 57-58
grouping statements, 119
if statements, 117-119
IDE (Integrated Development Environments)

interactive mode, 59-60
math operators in Python scripts, 99-102
scripting in, 60-66
Komodo IDE development environment shell, 57
PyCharm development environment shell, 57
PyDev Open Source Plug-In for Eclipse, 57
IDLE development environment shell, 57-58
interactive mode, 59-60
Python scripts
   grouping statements, 119
   if statements, 118-119
   math operators, 99-102
scripting in, 60-66
if statements (Python scripts), 117-121
image handling
   game programming, 410-413
   HD images
      centering, 507-508
      converting, 512-513
      defining, 498-500
      finding, 501-502
      framing, 508
      image presentation script, 500
      improving script performance, 510-516
      megapixels, 498
      mouse/keyboard controls, 514
      movies, 502
      optimized presentation, 514-516
      potential modifications, 516-517
      preloading, 513
   presentation screen setup, 500-501
   scaling, 505-507
   screen buffering, 512
   storing on removable drives, 502-505
   testing script, 508-510
   title screens, 513-514
   music, playing with, 525-530
imaginary numbers (Python scripts), 107
imaplib module (network programming), 427
infinite loops (Python scripts), 151
inheritance and subclasses (OOP classes), 308-311
object module files
   adding additional subclasses to, 313-315
   adding subclasses to, 312-313
   putting a subclass in its own object module file, 315-316
subclasses, creating, 311-312
installation software
   NOOBS installation software, 19
      composite output, 22
      downloading, 19-21
      moving files/folders to MicroSD cards, 21
      OS installation, 22-24
      troubleshooting, 22, 25
   Raspberry Pi setups, 19-21
installing
   NOOBS installation software, 19-22
   OS in new Raspberry Pi setups, 22-24
   Python, 50-51
instances (OOP classes)
   creating, 293
   deleting, 299-300
interactive shell (Python), 53-55
interpreter (Python), 49, 52-53
intersections (sets), 195
iteration (loops), 137
   infinite loops, 151
   lists, 172
for loops, 137
   assigning data types from lists, 141-142
   indentation in, 138
   iterating character strings in lists, 142-143
   iterating numeric values in lists, 138-140
   iterating using range function, 143-146
   iterating using variables, 143
   operation of, 138
   syntax of, 138
   troubleshooting, 140-141
   validating user input via, 146-148
while loops, 148
   break statements, 151-154
   entering data via, 152-154
   infinite loops, 151
   iterating using numeric conditions, 149
   iterating using string conditions, 149-151
   pretests, 149
   syntax of, 148-149
   terminating, 150
while True, 151-154
J - K

KDE GUI, 36
keyboards
  buying, 14-15
  HD image presentation, 514
Python setup, 51-52
USB keyboards, power consumption, 15
keywords in Python scripts, 83-84
Kindle eBook reader, 29
kits (peripherals), buying, 18
kivy game library, 398
Komodo IDE development environment shell, 57

L

Label widget (GUI programming), 374, 384
LED light (GPIO interface), 544-546
lighttp, 475
linked modules, 270
linking programs via socket programming, 442
  client programs, 446-449
  client/server communication process, 442-443
  client programs, 444-449
  running client/server demo, 448-449
  server programs, 444-446
  closing sockets, 449
  defining, 442-443
  server programs, 444-446, 448-449
  socket module, 443-444
Linux, 29
  Debian and Raspbian OS distribution, 29-30
  devices using Linux, 29
directory structure, 226
  absolute directory references, 226-227, 232
  relative directory references, 226-227
top root directory, 226
e-mail servers, 429-430
  MDA, 430
  MTA, 430
  MUA, 430-431
  Postfix, 430
  sendmail, 430
GUI programming
  accessing, 35
  creating a GUI program, 392-395
  frame templates, 378-379
  GNOME GUI, 36
KDE GUI, 36
  LXDE GUI, 35-43
  packages, 375
  PyGTK GUI package, 375
  PyQt GUI package, 375
tkinter GUI package, 375-395
  widgets, 376-395
  wxPython GUI package, 375
  Xfce GUI, 36
Linux shell, 31
MySQL database, 453
  creating databases, 455-456
  creating Python scripts, 460-464
  creating tables, 457-459
  creating user accounts, 456-457
data types, 458
downloading Debian packages, 460
installing, 454
installing Python MySQL/Connector module, 459-460
installing Python PostgreSQL module, 469
root user accounts, 454
setting up, 454-459
NOOBS installation software
  formatting MicroSD cards, 562-564
unpacking zip files, 561
verifying checksums, 560
PostgreSQL database, 464
  creating databases, 465-466
  creating tables, 467-469
  creating user accounts, 466-467
database connections, 469-470
  formatting data, 470
  inserting data, 470-471
installing, 464
psycopg2 module, 469-472
querying data, 471-472
setting up, 464-469
Raspbian OS
  basic command-line commands, 31
  Debian and, 29-30
  entering commands at command-line, 31-33
Linux distribution, 29-30
logins, 30-33
passwords, 32, 35
Listbox widget (GUI programming), 374, 390-391
lists (Python scripts), 164
comprehensions, 173-174
concatenating, 169-170
creating, 164-165
extracting data from, 165
functions and, 263-264
functions of, 170-171
iterating through, 172
multidimensional lists, 171
values
adding new data values, 167-169
deleting, 166-167
popping, 167
replacing, 165-166
reversing, 171-173
sorting, 172-173
sorting in place, 170
local variables and functions, 260-261
logarithmic functions (math module), 109-110
logic operators (Python scripts), 130-131
logins
Raspberry Pi, 30-33
Raspbian OS, 30-33
loops (Python scripts), 137
game loops, 409
infinite loops, 151
for loops, 137
assigning data types from lists, 141-142
indentation in, 138
iterating character strings in lists, 142-143
iterating numeric values in lists, 138-140
iterating using range function, 143-146
iterating using variables, 143
operation of, 138
syntax of, 138
troubleshooting, 140-141
validating user input via, 146-148
nested loops, 154-156
while loops, 148
break statements, 151-154
entering data via, 152-154
infinite loops, 151
iterating using numeric conditions, 149
iterating using string conditions, 149-151
pretests, 149
syntax of, 148-149
terminating, 150
while True, 151-154
ls command, 31, 33
LXDE GUI, 35-36
desktop area, 36-37
LXPanel area, 36-37, 40-43
applets, 37-38
LXDE file manager, 38
LXDE menu, 38
LXDE Screensaver Preferences window, 42-43
LXTerminal command-line interface, 39
LXML module, installing (web servers and network programming), 437-438
M
mailbox module (network programming), 427
mailcap module (network programming), 427
match() function (regular expressions), 333-334
math module in Python scripts, 108
hyperbolic functions, 111
number theory functions, 109
power and logarithmic functions, 109-110
statistical math functions, 111-112
trigonometric functions, 110-111
math operators in Python scripts, 99-101
displaying numbers, 104-105
floating point accuracy, 103-104
operator shortcuts, 105
order of operations, 101-102
variables in math calculations, 102-103
MDA (Mail Delivery Agents), Linux modular email environment, 430
megapixels in HD images, 498
memberships (sets), 194
Menu widget (GUI programming), 374, 391-392
methods (OOP classes), 292, 294
accessor methods, 295-297
constructors, 297-299
customizing output, 299
destuctors, 299-300
documenting classes, 300-301
helper methods, 297-302
mutator methods, 294-295
property() helper method, 301-302
mhlib module (network programming), 427
MicroSD cards
buying, 10-12
NOOBS installation software copying to MicroSD cards, 566
moving files/folders to MicroSD cards, 21
repartitioning drives, 22
preloaded MicroSD cards, 19
Raspberry Pi 2 Model B, 10
SD cards versus, 10
setting up, 21
size of, 12
troubleshooting, 25
mkdir command, 31, 33
modules, 271
built-in modules, 270
categories of, 271-272
custom modules
creating, 277-278, 284-287
creating in test directories, 278-279
gathering functions for, 278
moving to production directories, 280-284
naming, 278
testing, 279-280, 284
using, 284-287
defining, 269
exploring available modules on Raspberry Pi, 276-277
finding, 272-273
flavors of, 269-271
functions
determining how to use functions within, 274-276
listing functions in modules, 274
grouping, 271
importing different flavors of, 270-271
linked modules, 270
moving to production directories, 280-284
naming, 278
network programming, 427-428
online resources, 273
packages, 271
reading module descriptions, 273-274
RPI.GPIO module (GPIO interface), 539
installing, 539-540
startup methods, 540-541
standard modules, 271-272
modules (OOP classes)
creating, 302-304
sharing code with, 302-304
Monkey HTTP, 475
Monty Python’s Flying Circus, 47
mouses (mice)
buying, 14-15
HD image presentation, 514
power consumption, 15
USB mouses (mice), 15
movies (HD), 502
moving NOOBS files/folders to microSD cards, 21
MP3 music format, 517-518
MTA (Mail Transfer Agents), Linux modular email environment, 430
MUA (Mail User Agents), Linux modular email environment, 430-431
multidimensional lists (Python scripts), 171
multiple exception handling, 358-361
exception groups, 362-364
generic exemptions, 364
try except statement statement blocks, 361-370
statement options, 364-365
music, 517
basic music script, 517-518
images, playing music with, 525-530
MP3 format, 517-518
playback control, 520-525
playlists
creating, 519-520
randomizing, 525
queuing songs, 518
storing on removable disks, 518-519
mutator methods (OOP classes), 294-295
MySQL database, 453
data types, 458
installing, 454
Python MySQL/Connector module, installing, 459-460
Python scripts, creating, 460
database connections, 461
database security, 461
inserting data, 461-463
primary key data constraints, 463
querying data, 463-464
root user accounts, 454
setting up, 454-455
creating databases, 455-456
creating tables, 457-459
creating user accounts, 456-457
downloading Debian packages, 460

defining, 442-443
server programs, 444-446, 448-449
socket module, 443-444
web servers, 436
example of, 427-441
LXML module, 437-441
parsing webpage data, 437-442
relocation of webpages, 442
retrieving webpages, 436-437
urllib module, 436-437

Nginx web server, 475

nntplib module (network programming), 427

NOOBS installation software, 19, 557-558

OS X, 560
Windows, 560

NTSC (National Television Systems Committee) color encoding, 25

numbers
complex numbers, 107-108
formatting strings for output, 219-222
imaginary numbers, 107
numeric comparisons (Python scripts), 126
numeric files, 225
theory functions (math module), 109

NumPy math libraries, 112
arrays, 113-114
data types, 112

online resources
Debian-related resources, 30
IDE, 57
Komodo IDE development environment shell, 57
modules, 273
NOOBS installation software, 558
PyCharm development environment shell, 57
PyDev Open Source Plug-In for Eclipse, 57
PyGame game library, 400
Python games, 399
Raspberry Pi Foundation, 19
Raspberry Pi wiki page, 11-12
retailers, buying from, 9-10
OOP (Object-Oriented Programming), 291
classes, 292
attributes, 292-294
creating class modules, 302-304
defining, 292
destructors, 299-300
documenting, 300-301
duplication in, 307-308
inheritance, 310-327
instances, 293, 299-300
methods, 292, 294-302
problem with, 307-308
property() helper method, 301-302
sharing code with class modules, 302-304
subclasses, 308-310, 327
defining, 291-292
inheritance and subclasses (OOP classes), 308-310
opening files, 237-240
absolute directory references, 232
file attributes, determining, 231-232
file object methods, 231-232
open function, 229-230
open mode, designating, 230-231
troubleshooting, 231
OS (Operating Systems)
new Raspberry Pi setups, OS installation, 22-24
Raspbian OS
basic command-line commands, 31
Debian and, 29-30
entering commands at command-line, 31-33
GNOME GUI, 36
KDE GUI, 36
Linux distribution, 29-30
logins, 30-33
LXDE GUI, 35-43
passwords, 32, 35
software packages, 30
Xfce GUI, 36
os function, file/directory management, 227-229
OS X and NOOBS installation software
checksums, verifying, 560
MicroSD cards, formatting, 565-566
zip files, unpacking, 562
output displays
buying, 14
DVI, 14
HDMI, 14
NTSC color encoding, 25
PAL color encoding, 25
RCA connectors, 14
troubleshooting, 25
VGA, 14
packages, 271
PAL (Phase Alternating Line) color encoding, 25
Panda3D game library, 398
passwords
blank passwords, 32
Raspberry Pi, 32, 35, 43
Raspbian OS, 32
peripherals
cases
buying, 16-17
static electricity, 17
keyboards
buying, 14-15
power consumption, 15
Python setup, 51-52
USB keyboards, 15
kits (prepackaged), 18
MicroSD cards
buying, 10-12
moving NOOBS files/folders to MicroSD cards, 21
preloaded MicroSD cards, 19
repartitioning drives, 22
SD cards versus, 10
size of, 12
troubleshooting, 25
mouses (mice)
buying, 14-15
power consumption, 15
USB mouses (mice), 15
necessary peripherals, determining, 10
network cables, buying, 15
new Raspberry Pi setups, plugging in peripherals, 21-22
output displays
buying, 14
DVI, 14
HDMI, 14
NTSC color encoding, 25
PAL color encoding, 25
RCA connectors, 14
troubleshooting, 25
VGA, 14
power supplies
buying, 12-13
cables, 12
portable power supplies, 17
peripherals

troubleshooting, 26
USB hubs
bus-powered USB hubs, 18
buying, 18
self-powered USB hubs, 18
Wi-Fi adapters, buying, 15
Pi Cobbler
GPIO interface
connections, 536-537
detecting input, 547-548
setting up Pi Cobbler for output, 541-543
ribbon cable, 537
pipe symbol (|) in regular expressions, 344-345
plain text searches in regular expressions, 335-337
playback control (music), 520-525
playlists (music)
creating, 519-520
randomizing, 525
plugging in peripherals to new Raspberry Pi setups, 21-22
plus sign (+) in regular expressions, 344
polling and GPIO interface input, 549-551
poplib module (network programming), 427
portable power supplies, buying, 17
POSIX BRE (Basic Regular Expression) engine, 332
POSIX ERE (Extended Regular Expression) engine, 332
Postfix, 430
PostgreSQL database, 464
installing, 464
psycopg2 module, 469
database connections, 469-470
formatting data, 470
inserting data, 470-471
querying data, 471-472
Python PostgreSQL module, installing, 469
setting up, 464-465
creating databases, 465-466
creating tables, 467-469
creating user accounts, 466-467
power and logarithmic functions (math module), 109-110
power supplies
buying, 12-13
cables, 12
portable power supplies, 17
preloaded MicroSD cards, 19
print function (Python), 73-74
displaying characters via, 74-75
formatting output, 75-77
private attributes (OOP classes), 295
procedural programming, 291
Progressbar widget (GUI programming), 374
property() helper method (OOP classes), 301-302
psycopg2 module and PostgreSQL database operation, 469
database connections, 469-470
formatting data, 470
inserting data, 470-471
querying data, 471-472
pull-ups/downs in GPIO interface input, 549
pwd command, 31, 33
PyCharm development environment shell, 57
PyDev Open Source Plug-In for Eclipse, 57
PyGame game library, 398, 409
checking for, 400
events, 409
game loops, 409
game screen
displaying text, 405-409
interacting with graphics on screen, 415-416
moving graphics on screen, 414-415
putting text on, 405
setting up, 403-409
image handling, 410-413
initializing, 402-403
installing, 399-400
loading, 402-403
modules, 401-402
object classes, 402
online resources, 400
setting up, 399-400
sound design, 413-414
sprites, 402
Pyglet game library, 398
PyGTK GUI package, 375
PyQT GUI package, 375
PySoy game library, 398
Python, 47
debugging, 486-488
development environment, 49, 53, 57
IDLE development environment shell, 57-62, 99-102
Komodo IDE development environment shell, 57
PyCharm development environment shell, 57
PyDev Open Source Plug-In for Eclipse, 57
9780672337642_print.indb 586
21/11/15 12:28 AM
dictionaries, 180
creating, 180
defining, 179-180
management operations, 185-186
obtaining data from, 182-184
populating, 180-181
programming, 186-192
retrieving values from
dictionaries for functions, 259-260
updating, 184-185
directories, 227
closing files, 239-240
creating files, 240-241
creating modules in, 278-279
managing, 227-229
moving modules to
production directories, 280-284
opening files, 229-232, 240
reading files, 233-239
writing files, 240-245
error exceptions
defining, 351
exception groups, 362-364
generic exemptions, 364
handling multiple
exceptions, 358-370
handling via try except
statement, 356-358, 361-370
runtime error exceptions, 354-356
syntactical error
exceptions, 351-353
file management
closing files, 239-240
creating files, 240-241
opening files, 229-232, 240
os function, 227-229
reading files, 233-239
writing files, 240-245
functions, 249
creating, 250
defining, 250-252
determining how to use
functions within, 274-276
gathering for custom
modules, 278
global variables, 260-263
lists and, 263-264
local variables, 260-261
modules and, 269, 274
passing values to, 254-259
recursion and, 264-265
retrieving values via
dictionaries, 259-260
returning values from, 253-254
using, 250-252

game programming, 397-399
Blender3D game library, 398
cocos2d game library, 398
developers versus
designers, 398
fifengine game
library, 398
game screen setup,
403-409
image handling, 410-413, 416-423
interacting with graphics
on screen, 415-416
kivy game library, 398
moving graphics on
screen, 414-423
Panda3D game library, 398
playing games online, 399
PyGame game library, 398-423
Pyglet game library, 398
PySoy game library, 398
Python-Ogre game library, 398
SDL, 399
sound design, 413-414
GUI programming, 373
creating a GUI program,
392-395
event-driven programming,
374-375
frame templates,
378-379
packages, 375
PyGTK GUI package, 375
PyQT GUI package, 375
tkinter GUI package, 375
widgets, 374-395
window interface, 374
wxPython GUI package, 375
HD images
centering, 507-508
converting, 512-513
defining, 498-500
finding, 501-502
framing, 508
image presentation script,
500
improving script
performance, 510-516
megapixels, 498
mouse/keyboard controls,
514
movies, 502
optimized presentation, 514-516
playing music with, 525-530
potential modifications, 516-517
preloading, 513
presentation screen setup, 500-501
scaling, 505-507
screen buffering, 512
storing on removable drives, 502-505
testing script, 508-510
title screens, 513-514
history of, 47-48
inheritance, 310-311
adding additional subclasses to object module files, 313-315
adding subclasses to object module files, 312-313
creating subclasses, 311-312
putting a subclass in its own object module file, 315-316
installing, 50-51
interactive shell, 49, 53-55
interpreter, 49, 52-53
introduction to, 1
keyboard setup, 51-52
modules, 271
built-in modules, 270
categories of, 271-272
creating custom modules, 277-279, 284-287
creating in test directories, 278-279
defining, 269
determining how to use functions within, 274-276
exploring available modules on Raspberry Pi, 276-277
finding, 272-273
flavors of, 269-271
grouping, 271
importing different flavors of, 270-271
linked modules, 270
listing functions in modules, 274
moving to production directories, 280-284
naming, 278
online resources, 273
reading module descriptions, 273-274
standard modules, 271-272
testing, 279-280, 284
using, 284-287
music, 517
basic music script, 517-518
creating playlists, 519-520
MP3 format, 517-518
playback control, 520-525
playing with images, 525-530
queuing songs, 518
randomizing playlists, 525
storing on removable disks, 518-519
MySQL database, creating Python scripts, 460-464
network programming
email servers, 428-436
modules, 427-428
socket programming, 442-449
web servers, 436-442
OOP, 291
classes, 291-294, 302-304, 307-308
defining, 291-292
inheritance, 308-310
instances, 293, 299-300
subclasses, 308-310
packages, 271
Python MySQL/Connector module, installing, 459-460
Python PostgreSQL module, installing, 469
Python v2, 48
Python v3, 48
ASCII in, 207-208
Python v2 versus, 48
Raspberry Pi’s relationship to, 7-8
regular expressions
anchor characters, 337-339
asterisk (*) in, 342-343
braces ({} ) in, 344
character classes, 340-343
compiling, 334-335
defining, 331-332
dot character, 339-340
findall() function, 333, 334-335
finditer() function, 333-335
finding() function, 333-335
functions, 333
matching, 345
match() function, 333-334
pipe symbol (|) in, 344-345
plain text searches, 335-337
quotes in Python scripts

single quotes (')
   displaying via print function, 74-75
   formatting via print function, 76-77
triple quotes ("""), formatting via print function, 76-77

R

Radiobutton widget (GUI programming), 374
randomizing music playlists, 525
ranges (Python scripts), 174-175
Raspberry Pi, 5
   buying
      peripherals, 10-18
      retailers, 9-10
      tips for, 8-10
   components, 1, 8-9
   development of, 1, 5-6
   different names for, 6
   GUI, booting straight to, 37
   HDMI port, 497
   history of, 5-6
   introduction to, 1
   logins, 30-33
   models of, 9
   modules available on, 276-277
   passwords, 32, 35, 43
   Python's relationship to, 7-8
   rebooting, 34-35
   setting up
      installation software, 19
      NOOBS installation software, 19-21
      OS installation, 22-24
   plugging in peripherals, 21-22
   researching possible setups, 19
   troubleshooting
      cable connections, 24
      microSD cards, 25
      NOOBS installation software, 25
      output displays, 25
      peripherals, 26
   uses for, 7
Raspberry Pi 1 Model A, 570-571
Raspberry Pi 1 Model A+, 7, 569
diagram of, 569
features of, 569
Raspberry Pi 1 Model B, 570
cases, 17
features of, 571
Raspberry Pi 1 Model B+, 568
diagram of, 567-568
features of, 568
Raspberry Pi 2 Model B, 567
cases, 16
diagram of, 9, 567
features of, 567
microSD cards, 10
SD cards, 10
Raspberry Pi Foundation, 6-7, 19
NOOBS installation software, 557-558
support for, 9
Raspberry Pi User Guide, 19
Raspberry Pi wiki page, 11-12
Raspbian OS
   command-line
      basic commands, 31
      entering commands, 31-33
      Debian and, 29-30
   GUI, accessing, 35
   Linux distribution, 29-30
   logins, 30-33
   passwords, 32, 35
   SD cards, loading Raspbian OS via NOOBS, 557-558
downloading NOOBS, 558-559
   formatting MicroSD cards, 562-566
   unpacking zip files, 561-562
   verifying checksums, 559-561
   software packages, 30
RCA connectors, output displays, 14
reading files, 233, 237-239
   entire files, 233-234
   line-by-line, 234-235
   newline characters, stripping from scripts, 235
   nonsequentially, 236-237
rebooting Raspberry Pi, 34-35
recursion and functions, 264-265
regular expressions
   anchor characters, 337-339
   asterisk (*) in, 342-343
   braces ( { } ) in, 344
   character classes
      asterisk (*) in, 342-343
      creating, 340-341
      negating, 341
      ranges, 341-342
   compiling, 334-335
defining, 331-332
dot character, 339-340
functions, 333
   findall() function, 333-335
   finditer() function, 333-335
scripts

resources (print), Raspberry Pi User Guide, 19

resources (print), Raspberry Pi User Guide, 19

resources (print), Raspberry Pi User Guide, 19

resources (print), Raspberry Pi User Guide, 19

else statements, 121-123

error exceptions

defining, 351

generic exemptions, 364

handling multiple exceptions, 358-370

handling via try except statement, 356-358, 361-370

runtime error exceptions, 354-356

syntactical error exceptions, 351-353

for loops, 137-148

formatting

for readability, 80-83

output via print function, 75-77

fractions, 105-106

functions, 249

creating, 250

defining, 250-252

determining how to use functions within, 274-276

gathering for custom modules, 278

global variables, 260-263

lists and, 263-264

local variables, 260-261

modules and, 269, 274

nested functions, 92

passing values to, 254-259

printing, 73-74

recursion and, 264-265

retrieving values via dictionaries, 259-260

returning values from, 253-254

S

scaling HD images, 505-507

screen buffering and HD images, 512

scripts

Boolean comparisons, 128

break statements, 151-154

comments, 82-83

comparison operators, 124, 126-130

complex numbers, 107-108

condition checks, 130-132

characters, displaying via print function, 74-75

conditions, negating, 131-132

creating, 68

output via print function, 79-80

via MySQL database, 460-464

via text editor, 66-68

data types, 89-92

directories, displaying scripts in, 227

e1if statements, 123-126

relative directory references (Linux directory structure), 226-227

remote email servers, 432

removable disks and music storage, 518-519

removable drives and HD image storage, 502-505

repartitioning drives, MicroSD cards, 22

researching possible setups (Raspberry Pi configuration), 19

resetting GPIO interface, 544

resources (online)

Debian-related resources, 30

IDE, 57

Komodo IDE development environment shell, 57

modules, 273

NOOBS installation software, 558

PyCharm development environment shell, 57

PyDev Open Source Plug-In for Eclipse, 57

PyGame game library, 400

Python games, 399

Raspberry Pi Foundation, 19

Raspberry Pi wiki page, 11-12

retailers, buying from, 9-10

resources (print), Raspberry Pi User Guide, 19

retailers, buying from, 9-10

robotparser module (network programming), 427

root user accounts in MySQL database, 454

RPi.GPIO module, 539

installing, 539-540

startup methods, 540-541

RS Components website, 10

runtime error exceptions, 354-356

pipe symbol (|) in, 344-345

plain text searches, 335-337

plus sign (+) in, 344

POSIX BRE engine, 332

POSIX ERE engine, 332

question mark (?) in, 343

special characters, 337

types of, 332

using, 346-348

break statements, 151-154

comments, 82-83

comparison operators, 124, 126-130

characters, displaying via print function, 74-75

conditions, negating, 131-132

creating, 68

output via print function, 79-80

via MySQL database, 460-464

via text editor, 66-68

data types, 89-92

directories, displaying scripts in, 227

e1if statements, 123-126

else statements, 121-123

error exceptions

defining, 351

generic exemptions, 364

handling multiple exceptions, 358-370

handling via try except statement, 356-358, 361-370

runtime error exceptions, 354-356

syntactical error exceptions, 351-353

for loops, 137-148

formatting

for readability, 80-83

output via print function, 75-77

fractions, 105-106

functions, 249

creating, 250

defining, 250-252

determining how to use functions within, 274-276

gathering for custom modules, 278

global variables, 260-263

lists and, 263-264

local variables, 260-261

modules and, 269, 274

nested functions, 92

passing values to, 254-259

printing, 73-74

recursion and, 264-265

retrieving values via dictionaries, 259-260

returning values from, 253-254

S

scaling HD images, 505-507

screen buffering and HD images, 512

scripts

Boolean comparisons, 128

break statements, 151-154

comments, 82-83

comparison operators, 124, 126-130

complex numbers, 107-108

condition checks, 130-132

characters, displaying via print function, 74-75

conditions, negating, 131-132

creating, 68

output via print function, 79-80

via MySQL database, 460-464

via text editor, 66-68

data types, 89-92

directories, displaying scripts in, 227

e1if statements, 123-126

else statements, 121-123

error exceptions

defining, 351

generic exemptions, 364

handling multiple exceptions, 358-370

handling via try except statement, 356-358, 361-370

runtime error exceptions, 354-356

syntactical error exceptions, 351-353

for loops, 137-148

formatting

for readability, 80-83

output via print function, 75-77

fractions, 105-106

functions, 249

creating, 250

defining, 250-252

determining how to use functions within, 274-276

gathering for custom modules, 278

global variables, 260-263

lists and, 263-264

local variables, 260-261

modules and, 269, 274

nested functions, 92

passing values to, 254-259

printing, 73-74

recursion and, 264-265

retrieving values via dictionaries, 259-260

returning values from, 253-254

S

scaling HD images, 505-507

screen buffering and HD images, 512

scripts

Boolean comparisons, 128

break statements, 151-154

comments, 82-83

comparison operators, 124, 126-130

complex numbers, 107-108

condition checks, 130-132

characters, displaying via print function, 74-75

conditions, negating, 131-132

creating, 68

output via print function, 79-80

via MySQL database, 460-464

via text editor, 66-68

data types, 89-92

directories, displaying scripts in, 227

e1if statements, 123-126

else statements, 121-123

error exceptions

defining, 351

generic exemptions, 364

handling multiple exceptions, 358-370

handling via try except statement, 356-358, 361-370

runtime error exceptions, 354-356

syntactical error exceptions, 351-353

for loops, 137-148

formatting

for readability, 80-83

output via print function, 75-77

fractions, 105-106

functions, 249

creating, 250

defining, 250-252

determining how to use functions within, 274-276

gathering for custom modules, 278

global variables, 260-263

lists and, 263-264

local variables, 260-261

modules and, 269, 274

nested functions, 92

passing values to, 254-259

printing, 73-74

recursion and, 264-265

retrieving values via dictionaries, 259-260

returning values from, 253-254
scripts

modules, 271

- built-in modules, 270
categories of, 271-272
creating custom modules, 277-279, 284-287
creating in test directories, 278-279
defining, 269
determining how to use functions within, 274-276
exploring available modules on Raspberry Pi, 276-277
finding, 272-273
flavors of, 269-271
grouping, 271
importing different flavors of, 270-271
linked modules, 270
listing functions in modules, 274
moving to production directories, 280-284

- naming, 278
online resources, 273
reading module descriptions, 273-274
standard modules, 271-272
testing, 279-280, 284
using, 284-287

- multidimensional lists, 171

- music, 517

- basic music script, 517-518
creating playlists, 519-520
MP3 format, 517-518
playback control, 520-525
playing with images, 525-530
queuing songs, 518
randomizing playlists, 525
storing on removable disks, 518-519
nested functions, 92
nested loops, 154-156
newline characters, stripping from scripts, 235
numeric comparisons, 126
NumPy math libraries, 112-114
packages, 271
print function, 73-74
ranges, 174-175

regular expressions

- anchor characters, 337-339
- asterisk (*) in, 342-343
- braces ( {} ) in, 344
- character classes, 340-343
- compiling, 334-335
defining, 331-332
dot character, 339-340
findall() function, 333-335
finditer() function, 333-335
functions, 333
grouping, 345
match() function, 333-334
pipe symbol (|) in, 344-345
plain text searches, 335-337
- plus sign (+) in, 344
- POSIX BRE engine, 332
- POSIX ERE engine, 332
- question mark (?) in, 343
- search() function, 333-334
- special characters, 337
- types of, 332
- using, 346-348
servers and network programming

socket programming, linking programs using

servers and network programming

email servers, 428-429
Gmail security, 436
Linux modular email environment, 429-431
Postfix, 430
remote email servers, 432
sending email messages, 431-436
sendmail, 430
smtplib module, 430-433
server programs (socket programming), 444-446, 448-449
web servers, 436
example of, 427-441
LXML module, 437-441
parsing webpage data, 437-442
relocation of webpages, 442
retrieving webpages, 436-437
urllib module, 436-437

setting up

keyboards for Python, 51-52
MicroSD cards, 21
Raspberry Pi
installation software, 19
NOOBS installation software, 19-21
OS installation, 22-24
plugging in peripherals, 21-22
researching possible setups, 19

shortcuts (math operator) in Python scripts, 105

SimpleXMLRPCServer module (network programming), 427

single quotes (') in Python scripts, 74-77

slices

strings, 210
tuples, 161-162

smtplib module (network programming), 427

smtplib module (network programming), 427, 430-431
class methods of, 431
classes of, 431
text email servers, 428-429
sending email messages, 431-433

sets (Python)
creating, 193
defining, 192-193
deleting elements from, 198-199
differences, 196-197
intersections, 195
memberships, 194
obtaining data from, 194-197
populating, 193-194
programming, 199-202
symmetric set differences, 196-197
traversing, 197
unions, 195
updating, 197-198

SoC (System on a Chip), 9

socket programming, linking programs using, 442

client programs, 446-449
client/server communication process, 442-443
client programs, 444-449
running client/server demo, 448-449
server programs, 444-446

Scrollbar widget (GUI programming), 374

SD cards

MicroSD cards
formatting, 562-566
SD cards versus, 10

Raspberry Pi 2 Model B, 10
Raspbian OS, loading on SD cards via NOOBS, 557-558
downloading NOOBS, 558-559
formatting MicroSD cards, 562-566
unpacking zip files, 561-562
verifying checksums, 559-561

SDL and game programming, 399

search() function (regular expressions), 333-334

security
databases, 461, 485
Gmail, 436
webpages, 485
self-powered USB hubs, 18
sendmail, 430
Separator widget (GUI programming), 374
594  socket programming, linking programs using

 closing sockets, 449
defining, 442-443
server programs, 444-446, 448-449
socket module, 443-444
software packages, Raspbian OS, 30
sound design in game programming, 413-414
Spinbox widget (GUI programming), 374
sprites (PyGame game library), 402
startx command, 35
statements
  escape sequences in, 77-79
exception handling
  exception groups, 362-364
  try except statements, 356-358, 361-370
  grouping, 119-121
if statements, 117-119
testing, 68
static electricity
cases, 17
circuit boards, 17
statistical math functions (math module), 111-112
storing
  HD images on removable drives, 502-505
  music on removable disks, 518-519
string files, 225
strings (Python), 207
  comparisons, 127-128
  creating, 208-209
  formats of, 207-208
  formatting for output, 217
    format() function, 217-218
    named placeholders, 218-219
numbers, 219-222
  positional formatting, 222
  positional placeholders, 218
joining, 213
manipulation functions, 210-212
searching, 215-217
splitting, 212-213
testing, 213-214
values
  altering, 210-212
  assigning, 209-210
  slices, 210
subclasses (OOP classes)
  creating, 311-312
  inheritance and, 308-311, 316-327
  adding additional subclasses to object module files, 313-315
  adding subclasses to object module files, 312-313
  creating subclasses, 311-312
  putting a subclass in its own object module file, 315-316
object module files
  adding additional subclasses to, 313-315
  adding subclasses to, 312-313
  putting a subclass in its own object module file, 315-316
sudo command
  booting straight to GUI, 37
  rebooting Raspberry Pi, 33-35
switch bounce, GPIO interface input, 553
symmetric set differences, 196-197
synchronous events, GPIO interface input, 551
syntactical error exceptions, 351-353

T
tables, creating in
  MySQL database, 457-459
  PostgreSQL database, 467-469
telnetlib module (network programming), 427
test directories, custom modules in, 278-279
testing
  functions in scripts, 129
  GPIO interface output, 543-544
  modules, 279-280, 284
text editors (Python), 50, 53, 66-68
  Text widget (GUI programming), 374, 388-390
title screens (HD images), 513-514
tkinter GUI package, 375-376, 384
  Button widget, 384-385
  Checkbutton widget, 385-387
  Entry widget, 387-388
  Label widget, 384
  Listbox widget, 390-391
  Menu widget, 391-392
  Text widget, 388-390
  window interface
    adding widgets to, 378-382
trigonometric functions (math module), 110-111
triple quotes ("""") in Python scripts, 76-77
troubleshooting directories, 231
files, opening, 231
for loops, 140-141
MicroSD cards, 25
NOOBS installation software, 22, 25
output displays, 25
peripherals, 26
Raspberry Pi cable connections, 24
microSD cards, 25
output displays, 25
peripherals, 26
try except statement and exception handling, 356-358
statement blocks, 361-370
statement options, 364-365
tuples (Python scripts), 159, 162
accessing data in, 161
ranges of value, 161-162
concatenating, 164
creating, 159-160
iterating through, 172
slices, 161-162
values checking, 162-163
finding minimum/maximum values, 163
finding the number of, 163

Unassigned variables in Python scripts, 86-87
Unicode escape sequences, 78-80
Unions (sets), 195
Upton, Eben, 5-6, 19
urlib module (network programming), 427, 436-437
urllibparse module (network programming), 427
USB hubs
bus-powered USB hubs, 18
self-powered USB hubs, 18
USB keyboards and power consumption, 15
USB mouses (mice) and power consumption, 15
user accounts, creating in MySQL database, 456-457
PostgreSQL database, 466-467

van Rossum, Guido, 47
variables
functions and global variables, 260-263
local variables, 260-261
Python scripts, 83
assigning expression results to, 88
assigning long string values to, 87
assigning numeric values to, 88
assigning value to, 85

Creating variable names, 84
data types and, 89-90
formatting output, 85-86
reassigning values to, 88-89
unassigned variables, 86-87

VGA output displays, 14

Web programming, 475

Watson (IBM), 29
web forms, 488
cgi module, 491-493
creating, 488-490
HTML elements, 488-489
web programming, 475

Apache web server, 475-476
cgi programming, 478-480
files and folders, 476
installing, 476-477
publishing webpages, 478
serving HTML files, 477-478
web forms, 488-493
cgi module, 491-493
CGI programming
creating Python programs, 479-480
debugging Python programs, 486-488
defining, 479
running Python programs, 479
web forms, 491-493
lighttp, 475
Monkey HTTP, 475
web programming

Nginx web server, 475
web forms, 488
creating, 488-490
HTML elements, 488-489
webpages
dynamic webpages, 482-485
formatting data, 480-482
publishing, 478
security, 485
web resources
Debian-related resources, 30
IDE, 57
Komodo IDE development environment shell, 57
modules, 273
NOOBS installation software, 558
PyCharm development environment shell, 57
PyDev Open Source Plug-In for Eclipse, 57
PyGame game library, 400
Python games, 399
Raspberry Pi Foundation, 19
Raspberry Pi wiki page, 11-12
retailers, buying from, 9-10
web servers and web programming, 436
example of, 427-441
LXML module, 437-438
finding data via CSS, 439-440
parsing HTML via etree method, 438-439
urllib module, 436-437
webpages
parsing data, 437-442
relocation of, 442
retrieving, 436-437
web servers and web programming, 475
Apache web server, 475-476
CGI programming, 478-480
files and folders, 476
installing, 476-477
publishing webpages, 478
serving HTML files, 477-478
web forms, 488-493
cgi module, 491-493
CGI programming
creating Python programs, 479-480
debugging Python programs, 486-488
defining, 479
running Python programs, 479
web forms, 491-493
lighttp, 475
Monkey HTTP, 475
Nginx web server, 475
web forms, 488
creating, 488-490
HTML elements, 488-489
widgets (GUI programming), 374
Button widget, 374, 384-385
Checkbutton widget, 374, 385-387
defining, 380-382
Entry widget, 374, 387-388
Frame widget, 374
Label widget, 374, 384
Listbox widget, 374, 390-391
Menu widget, 374, 391-392
Progressbar widget, 374
Radiobutton widget, 374
Scrollbar widget, 374
Separator widget, 374
Spinbox widget, 374
Text widget, 374, 388-390
window interface
adding widgets to, 378-382
frame templates, 378-379
positioning widgets in, 379-380
Wi-Fi adapters, buying, 15
window interface (GUI programming), tkinter GUI package, 374
creating, 376-377
event handlers, 382-384
widgets, adding, 378-382
Windows and NOOBS installation software
checksums, verifying, 560
MicroSD cards, formatting, 564-565
zip files, unpacking, 561-562
writing files, 240-245
numbers as strings, 242
preexisting files, 243-244
write mode removals, 241
wxPython GUI package, 375

X

Xfce GUI, 36
XML files, 225
xmlrpc module (network programming), 427
.xz files, 225

Y - Z

.zip files, 225
Linux and NOOB unpackaging, 561
OS X and NOOB unpackaging, 562
Windows and NOOB unpackaging, 561-562