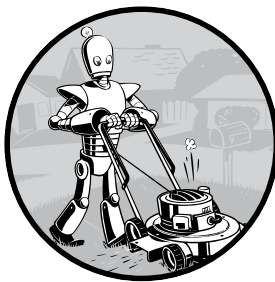


# A

## INSTALLING THIRD-PARTY MODULES



Beyond the standard library of modules packaged with Python, other developers have written their own modules to extend Python's capabilities even further. The primary way to install third-party modules is to use Python's **pip** tool. This tool securely downloads and installs Python modules onto your computer from <https://pypi.python.org/>, the website of the Python Software Foundation. PyPI, or the Python Package Index, is a sort of free app store for Python modules.

### The **pip** Tool

The executable file for the **pip** tool is called *pip* on Windows and *pip3* on OS X and Linux. On Windows, you can find **pip** at `C:\Python34\Scripts\pip.exe`. On OS X, it is in `/Library/Frameworks/Python.framework/Versions/3.4/bin/pip3`. On Linux, it is in `/usr/bin/pip3`.

While pip comes automatically installed with Python 3.4 on Windows and OS X, you must install it separately on Linux. To install pip3 on Ubuntu or Debian Linux, open a new Terminal window and enter `sudo apt-get install python3-pip`. To install pip3 on Fedora Linux, enter `sudo yum install python3-pip` into a Terminal window. You will need to enter the administrator password for your computer in order to install this software.

## Installing Third-Party Modules

The pip tool is meant to be run from the command line: You pass it the command `install` followed by the name of the module you want to install. For example, on Windows you would enter `pip install ModuleName`, where *ModuleName* is the name of the module. On OS X and Linux, you'll have to run pip3 with the `sudo` prefix to grant administrative privileges to install the module. You would need to type `sudo pip3 install ModuleName`.

If you already have the module installed but would like to upgrade it to the latest version available on PyPI, run `pip install -U ModuleName` (or `pip3 install -U ModuleName` on OS X and Linux).

After installing the module, you can test that it installed successfully by running `import ModuleName` in the interactive shell. If no error messages are displayed, you can assume the module was installed successfully.

You can install all of the modules covered in this book by running the commands listed next. (Remember to replace `pip` with `pip3` if you're on OS X or Linux.)

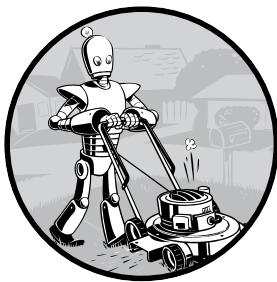
- `pip install send2trash`
- `pip install requests`
- `pip install beautifulsoup4`
- `pip install selenium`
- `pip install openpyxl`
- `pip install PyPDF2`
- `pip install python-docx` (install `python-docx`, not `docx`)
- `pip install imapclient`
- `pip install pyzmail`
- `pip install twilio`
- `pip install pillow`
- `pip install pyobjc-core` (on OS X only)
- `pip install pyobjc` (on OS X only)
- `pip install python3-xlib` (on Linux only)
- `pip install pyautogui`

### NOTE

*For OS X users: The `pyobjc` module can take 20 minutes or longer to install, so don't be alarmed if it takes a while. You should also install the `pyobjc-core` module first, which will reduce the overall installation time.*

# B

## RUNNING PROGRAMS



If you have a program open in IDLE's file editor, running it is a simple matter of pressing F5 or selecting the Run ► Run Module menu item. This is an easy way to run programs while writing them, but opening IDLE to run your finished programs can be a burden. There are more convenient ways to execute Python scripts.

### Shebang Line

The first line of all your Python programs should be a *shebang* line, which tells your computer that you want Python to execute this program. The shebang line begins with #!, but the rest depends on your operating system.

- On Windows, the shebang line is `#! python3`.
- On OS X, the shebang line is `#! /usr/bin/env python3`.
- On Linux, the shebang line is `#! /usr/bin/python3`.

You will be able to run Python scripts from IDLE without the shebang line, but the line is needed to run them from the command line.

## Running Python Programs on Windows

On Windows, the Python 3.4 interpreter is located at *C:\Python34\python.exe*. Alternatively, the convenient *py.exe* program will read the shebang line at the top of the *.py* file's source code and run the appropriate version of Python for that script. The *py.exe* program will make sure to run the Python program with the correct version of Python if multiple versions are installed on your computer.

To make it convenient to run your Python program, create a *.bat* batch file for running the Python program with *py.exe*. To make a batch file, make a new text file containing a single line like the following:

---

```
@py.exe C:\path\to\your\pythonScript.py %*
```

---

Replace this path with the absolute path to your own program, and save this file with a *.bat* file extension (for example, *pythonScript.bat*). This batch file will keep you from having to type the full absolute path for the Python program every time you want to run it. I recommend you place all your batch and *.py* files in a single folder, such as *C:\MyPythonScripts* or *C:\Users\YourName\PythonScripts*.

The *C:\MyPythonScripts* folder should be added to the system path on Windows so that you can run the batch files in it from the Run dialog. To do this, modify the PATH environment variable. Click the **Start** button and type **Edit environment variables for your account**. This option should auto-complete after you've begun to type it. The Environment Variables window that appears will look like Figure B-1.

From System variables, select the Path variable and click **Edit**. In the Value text field, append a semicolon, type **C:\MyPythonScripts**, and then click **OK**. Now you can run any Python script in the *C:\MyPythonScripts* folder by simply pressing WIN-R and entering the script's name. Running *pythonScript*, for instance, will run *pythonScript.bat*, which in turn will save you from having to run the whole command *py.exe C:\MyPythonScripts\pythonScript.py* from the Run dialog.

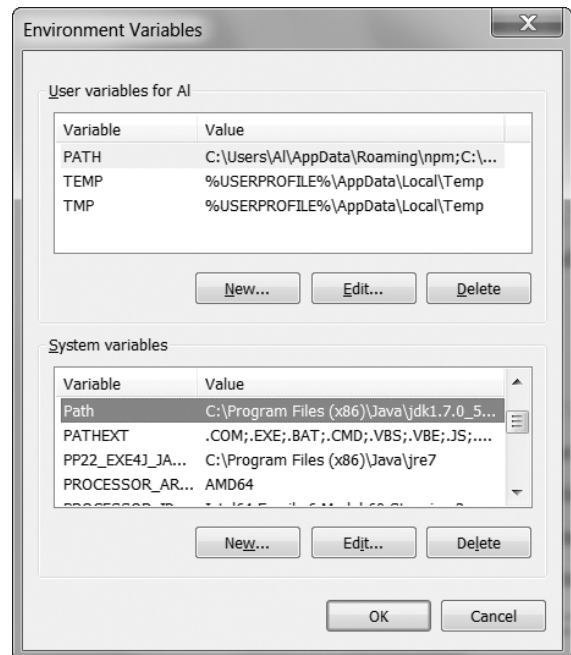


Figure B-1: The Environment Variables window on Windows

## Running Python Programs on OS X and Linux

On OS X, selecting Applications ► Utilities ► Terminal will bring up a *Terminal* window. A Terminal window is a way to enter commands on your computer using only text, rather than clicking through a graphic interface. To bring up the Terminal window on Ubuntu Linux, press the WIN (or SUPER) key to bring up Dash and type in **Terminal**.

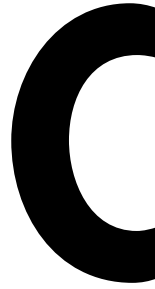
The Terminal window will begin in the home folder of your user account. If my username is *asweigart*, the home folder will be */Users/asweigart* on OS X and */home/asweigart* on Linux. The tilde (~) character is a shortcut for your home folder, so you can enter `cd ~` to change to your home folder. You can also use the `cd` command to change the current working directory to any other directory. On both OS X and Linux, the `pwd` command will print the current working directory.

To run your Python programs, save your *.py* file to your home folder. Then, change the *.py* file's permissions to make it executable by running `chmod +x pythonScript.py`. File permissions are beyond the scope of this book, but you will need to run this command on your Python file if you want to run the program from the Terminal window. Once you do so, you will be able to run your script whenever you want by opening a Terminal window and entering `./pythonScript.py`. The shebang line at the top of the script will tell the operating system where to locate the Python interpreter.

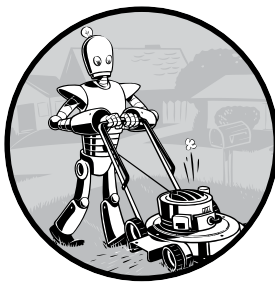
## Running Python Programs with Assertions Disabled

You can disable the `assert` statements in your Python programs for a slight performance improvement. When running Python from the terminal, include the `-O` switch after `python` or `python3` and before the name of the *.py* file. This will run an optimized version of your program that skips the assertion checks.





## **ANSWERS TO THE PRACTICE QUESTIONS**



This appendix contains the answers to the practice problems at the end of each chapter. I highly recommend that you take the time to work through these problems. Programming is more than memorizing syntax and a list of function names. As when learning a foreign language, the more practice you put into it, the more you will get out of it. There are many websites with practice programming problems as well. You can find a list of these at <http://nostarch.com/automatestuff/>.

## Chapter 1

1. The operators are +, -, \*, and /. The values are 'hello', -88.8, and 5.
2. The string is 'spam'; the variable is spam. Strings always start and end with quotes.
3. The three data types introduced in this chapter are integers, floating-point numbers, and strings.
4. An expression is a combination of values and operators. All expressions evaluate (that is, reduce) to a single value.
5. An expression evaluates to a single value. A statement does not.
6. The bacon variable is set to 20. The bacon + 1 expression does not reassign the value in bacon (that would need an assignment statement: bacon = bacon + 1).
7. Both expressions evaluate to the string 'spamspamspam'.
8. Variable names cannot begin with a number.
9. The int(), float(), and str() functions will evaluate to the integer, float-point number, and string versions of the value passed to them.
10. The expression causes an error because 99 is an integer, and only strings can be concatenated to other strings with the + operator. The correct way is I have eaten ' + str(99) + ' burritos.'.

## Chapter 2

1. True and False, using capital *T* and *F*, with the rest of the word in lowercase
2. and, or, and not
3. True and True is True.  
True and False is False.  
False and True is False.  
False and False is False.  
True or True is True.  
True or False is True.  
False or True is True.  
False or False is False.  
not True is False.  
not False is True.
4. False  
False  
True  
False  
False  
True



5. ==, !=, <, >, <=, and >=.
6. == is the equal to operator that compares two values and evaluates to a Boolean, while = is the assignment operator that stores a value in a variable.
7. A condition is an expression used in a flow control statement that evaluates to a Boolean value.
8. The three blocks are everything inside the if statement and the lines `print('bacon')` and `print('ham')`.

---

```
print('eggs')
if spam > 5:
    print('bacon')
else:
    print('ham')
print('spam')
```

---

9. The code:

---

```
if spam == 1:
    print('Hello')
elif spam == 2:
    print('Howdy')
else:
    print('Greetings!')
```

---

10. Press CTRL-C to stop a program stuck in an infinite loop.
11. The `break` statement will move the execution outside and just after a loop. The `continue` statement will move the execution to the start of the loop.
12. They all do the same thing. The `range(10)` call ranges from 0 up to (but not including) 10, `range(0, 10)` explicitly tells the loop to start at 0, and `range(0, 10, 1)` explicitly tells the loop to increase the variable by 1 on each iteration.
13. The code:

---

```
for i in range(1, 11):
    print(i)
```

---

and:

---

```
i = 1
while i <= 10:
    print(i)
    i = i + 1
```

---

14. This function can be called with `spam.bacon()`.

## Chapter 3

1. Functions reduce the need for duplicate code. This makes programs shorter, easier to read, and easier to update.
2. The code in a function executes when the function is called, not when the function is defined.
3. The `def` statement defines (that is, creates) a function.
4. A function consists of the `def` statement and the code in its `def` clause. A function call is what moves the program execution into the function, and the function call evaluates to the function's return value.
5. There is one global scope, and a local scope is created whenever a function is called.
6. When a function returns, the local scope is destroyed, and all the variables in it are forgotten.
7. A return value is the value that a function call evaluates to. Like any value, a return value can be used as part of an expression.
8. If there is no return statement for a function, its return value is `None`.
9. A `global` statement will force a variable in a function to refer to the global variable.
10. The data type of `None` is `NoneType`.
11. That `import` statement imports a module named `areallyourpetsnamederic`. (This isn't a real Python module, by the way.)
12. This function can be called with `spam.bacon()`.
13. Place the line of code that might cause an error in a `try` clause.
14. The code that could potentially cause an error goes in the `try` clause. The code that executes if an error happens goes in the `except` clause.

## Chapter 4

1. The empty list value, which is a list value that contains no items. This is similar to how `''` is the empty string value.
2. `spam[2] = 'hello'` (Notice that the third value in a list is at index 2 because the first index is 0.)
3. `'d'` (Note that `'3' * 2` is the string `'33'`, which is passed to `int()` before being divided by 11. This eventually evaluates to 3. Expressions can be used wherever values are used.)
4. `'d'` (Negative indexes count from the end.)
5. `['a', 'b']`
6. `1`
7. `[3.14, 'cat', 11, 'cat', True, 99]`
8. `[3.14, 11, 'cat', True]`

9. The operator for list concatenation is `+`, while the operator for replication is `*`. (This is the same as for strings.)
10. While `append()` will add values only to the end of a list, `insert()` can add them anywhere in the list.
11. The `del` statement and the `remove()` list method are two ways to remove values from a list.
12. Both lists and strings can be passed to `len()`, have indexes and slices, be used in `for` loops, be concatenated or replicated, and be used with the `in` and `not in` operators.
13. Lists are mutable; they can have values added, removed, or changed. Tuples are immutable; they cannot be changed at all. Also, tuples are written using parentheses, `(` and `)`, while lists use the square brackets, `[` and `]`.
14. `(42,)` (The trailing comma is mandatory.)
15. The `tuple()` and `list()` functions, respectively
16. They contain references to list values.
17. The `copy.copy()` function will do a shallow copy of a list, while the `copy.deepcopy()` function will do a deep copy of a list. That is, only `copy.deepcopy()` will duplicate any lists inside the list.

## Chapter 5

1. Two curly brackets: `{}`
2. `{'foo': 42}`
3. The items stored in a dictionary are unordered, while the items in a list are ordered.
4. You get a `KeyError` error.
5. There is no difference. The `in` operator checks whether a value exists as a key in the dictionary.
6. `'cat' in spam` checks whether there is a `'cat'` key in the dictionary, while `'cat' in spam.values()` checks whether there is a value `'cat'` for one of the keys in `spam`.
7. `spam.setdefault('color', 'black')`
8. `pprint.pprint()`

## Chapter 6

1. Escape characters represent characters in string values that would otherwise be difficult or impossible to type into code.
2. `\n` is a newline; `\t` is a tab.
3. The `\\` escape character will represent a backslash character.

4. The single quote in `Howl's` is fine because you've used double quotes to mark the beginning and end of the string.
5. Multiline strings allow you to use newlines in strings without the `\n` escape character.
6. The expressions evaluate to the following:
  - `'e'`
  - `'Hello'`
  - `'Hello'`
  - `'lo world!'`
7. The expressions evaluate to the following:
  - `'HELLO'`
  - `True`
  - `'hello'`
8. The expressions evaluate to the following:
  - `['Remember,', 'remember,', 'the', 'fifth', 'of', 'November.']`
  - `'There-can-be-only-one.'`
9. The `rjust()`, `ljust()`, and `center()` string methods, respectively
10. The `lstrip()` and `rstrip()` methods remove whitespace from the left and right ends of a string, respectively.

## Chapter 7

1. The `re.compile()` function returns `Regex` objects.
2. Raw strings are used so that backslashes do not have to be escaped.
3. The `search()` method returns `Match` objects.
4. The `group()` method returns strings of the matched text.
5. Group 0 is the entire match, group 1 covers the first set of parentheses, and group 2 covers the second set of parentheses.
6. Periods and parentheses can be escaped with a backslash: `\.`, `\(`, and `\)`.
7. If the regex has no groups, a list of strings is returned. If the regex has groups, a list of tuples of strings is returned.
8. The `|` character signifies matching “either, or” between two groups.
9. The `?` character can either mean “match zero or one of the preceding group” or be used to signify nongreedy matching.
10. The `+` matches one or more. The `*` matches zero or more.
11. The `{3}` matches exactly three instances of the preceding group. The `{3,5}` matches between three and five instances.
12. The `\d`, `\w`, and `\s` shorthand character classes match a single digit, word, or space character, respectively.
13. The `\D`, `\W`, and `\S` shorthand character classes match a single character that is not a digit, word, or space character, respectively.

14. Passing `re.I` or `re.IGNORECASE` as the second argument to `re.compile()` will make the matching case insensitive.
15. The `.` character normally matches any character except the newline character. If `re.DOTALL` is passed as the second argument to `re.compile()`, then the dot will also match newline characters.
16. The `.*` performs a greedy match, and the `.*?` performs a nongreedy match.
17. Either `[0-9a-z]` or `[a-z0-9]`
18. `'X drummers, X pipers, five rings, X hens'`
19. The `re.VERBOSE` argument allows you to add whitespace and comments to the string passed to `re.compile()`.
20. `re.compile(r'^\d{1,3}(\,\d{3})*$')` will create this regex, but other regex strings can produce a similar regular expression.
21. `re.compile(r'[A-Z][a-z]*\sNakamoto')`
22. `re.compile(r'(Alice|Bob|Carol)\s(eats|pets|throws)\s(apples|cats|baseballs)\. ', re.IGNORECASE)`

## Chapter 8

1. Relative paths are relative to the current working directory.
2. Absolute paths start with the root folder, such as `/` or `C:\`.
3. The `os.getcwd()` function returns the current working directory. The `os.chdir()` function *changes* the current working directory.
4. The `.` folder is the current folder, and `..` is the parent folder.
5. `C:\bacon\eggs` is the dir name, while `spam.txt` is the base name.
6. The string `'r'` for read mode, `'w'` for write mode, and `'a'` for append mode
7. An existing file opened in write mode is erased and completely overwritten.
8. The `read()` method returns the file's entire contents as a single string value. The `readlines()` method returns a list of strings, where each string is a line from the file's contents.
9. A shelf value resembles a dictionary value; it has keys and values, along with `keys()` and `values()` methods that work similarly to the dictionary methods of the same names.

## Chapter 9

1. The `shutil.copy()` function will copy a single file, while `shutil.copytree()` will copy an entire folder, along with all its contents.
2. The `shutil.move()` function is used for renaming files, as well as moving them.

3. The `send2trash` functions will move a file or folder to the recycle bin, while `shutil` functions will permanently delete files and folders.
4. The `zipfile.ZipFile()` function is equivalent to the `open()` function; the first argument is the filename, and the second argument is the mode to open the ZIP file in (read, write, or append).

## Chapter 10

1. `assert(spam >= 10, 'The spam variable is less than 10.')`
2. `assert(eggs.lower() != bacon.lower(), 'The eggs and bacon variables are the same!')` or `assert(eggs.upper() != bacon.upper(), 'The eggs and bacon variables are the same!')`
3. `assert(False, 'This assertion always triggers.')`
4. To be able to call `logging.debug()`, you must have these two lines at the start of your program:

---

```
import logging
logging.basicConfig(level=logging.DEBUG, format=' %(asctime)s -
%(levelname)s - %(message)s')
```

---

5. To be able to send logging messages to a file named *programLog.txt* with `logging.debug()`, you must have these two lines at the start of your program:

---

```
import logging
>>> logging.basicConfig(filename='programLog.txt', level=logging.DEBUG,
format=' %(asctime)s - %(levelname)s - %(message)s')
```

---

6. DEBUG, INFO, WARNING, ERROR, and CRITICAL
7. `logging.disable(logging.CRITICAL)`
8. You can disable logging messages without removing the logging function calls. You can selectively disable lower-level logging messages. You can create logging messages. Logging messages provides a timestamp.
9. The Step button will move the debugger into a function call. The Over button will quickly execute the function call without stepping into it. The Out button will quickly execute the rest of the code until it steps out of the function it currently is in.
10. After you click Go, the debugger will stop when it has reached the end of the program or a line with a breakpoint.
11. A breakpoint is a setting on a line of code that causes the debugger to pause when the program execution reaches the line.
12. To set a breakpoint in IDLE, right-click the line and select **Set Breakpoint** from the context menu.

## Chapter 11

1. The `webbrowser` module has an `open()` method that will launch a web browser to a specific URL, and that's it. The `requests` module can download files and pages from the Web. The `BeautifulSoup` module parses HTML. Finally, the `selenium` module can launch and control a browser.
2. The `requests.get()` function returns a `Response` object, which has a `text` attribute that contains the downloaded content as a string.
3. The `raise_for_status()` method raises an exception if the download had problems and does nothing if the download succeeded.
4. The `status_code` attribute of the `Response` object contains the HTTP status code.
5. After opening the new file on your computer in `'wb'` “write binary” mode, use a `for` loop that iterates over the `Response` object's `iter_content()` method to write out chunks to the file. Here's an example:

---

```
saveFile = open('filename.html', 'wb')
for chunk in res.iter_content(100000):
    saveFile.write(chunk)
```

---

6. F12 brings up the developer tools in Chrome. Pressing CTRL-SHIFT-C (on Windows and Linux) or ⌘-OPTION-C (on OS X) brings up the developer tools in Firefox.
7. Right-click the element in the page, and select **Inspect Element** from the menu.
8. `'#main'`
9. `'.highlight'`
10. `'div div'`
11. `'button[value="favorite"]'`
12. `spam.getText()`
13. `linkElem.attrs`
14. The `selenium` module is imported with `from selenium import webdriver`.
15. The `find_element_*` methods return the first matching element as a `WebElement` object. The `find_elements_*` methods return a list of all matching elements as `WebElement` objects.
16. The `click()` and `send_keys()` methods simulate mouse clicks and keyboard keys, respectively.
17. Calling the `submit()` method on any element within a form submits the form.
18. The `forward()`, `back()`, and `refresh()` `WebDriver` object methods simulate these browser buttons.

## Chapter 12

1. The `openpyxl.load_workbook()` function returns a `Workbook` object.
2. The `get_sheet_names()` method returns a `Worksheet` object.
3. Call `wb.get_sheet_by_name('Sheet1')`.
4. Call `wb.get_active_sheet()`.
5. `sheet['C5'].value` or `sheet.cell(row=5, column=3).value`
6. `sheet['C5'] = 'Hello'` or `sheet.cell(row=5, column=3).value = 'Hello'`
7. `cell.row` and `cell.column`
8. They return the highest column and row with values in the sheet, respectively, as integer values.
9. `openpyxl.cell.column_index_from_string('M')`
10. `openpyxl.cell.get_column_letter(14)`
11. `sheet['A1':'F1']`
12. `wb.save('example.xlsx')`
13. A formula is set the same way as any value. Set the cell's value attribute to a string of the formula text. Remember that formulas begin with the `=` sign.
14. When calling `load_workbook()`, pass `True` for the `data_only` keyword argument.
15. `sheet.row_dimensions[5].height = 100`
16. `sheet.column_dimensions['C'].hidden = True`
17. OpenPyXL 2.0.5 does not load freeze panes, print titles, images, or charts.
18. Freeze panes are rows and columns that will always appear on the screen. They are useful for headers.
19. `openpyxl.charts.Reference()`, `openpyxl.charts.Series()`, `openpyxl.charts.BarChart()`, `chartObj.append(seriesObj)`, and `add_chart()`

## Chapter 13

1. A `File` object returned from `open()`
2. Read-binary (`'rb'`) for `PdfFileReader()` and write-binary (`'wb'`) for `PdfFileWriter()`
3. Calling `getPage(4)` will return a `Page` object for page 5, since page 0 is the first page.
4. The `numPages` variable stores an integer of the number of pages in the `PdfFileReader` object.
5. Call `decrypt('swordfish')`.
6. The `rotateClockwise()` and `rotateCounterClockwise()` methods. The degrees to rotate is passed as an integer argument.



7. `docx.Document('demo.docx')`
8. A document contains multiple paragraphs. A paragraph begins on a new line and contains multiple runs. Runs are contiguous groups of characters within a paragraph.
9. Use `doc.paragraphs`.
10. A Run object has these variables (*not* a Paragraph).
11. `True` always makes the Run object bolded and `False` makes it always not bolded, no matter what the style's bold setting is. `None` will make the Run object just use the style's bold setting.
12. Call the `docx.Document()` function.
13. `doc.add_paragraph('Hello there!')`
14. The integers 0, 1, 2, 3, and 4

## Chapter 14

1. In Excel, spreadsheets can have values of data types other than strings; cells can have different fonts, sizes, or color settings; cells can have varying widths and heights; adjacent cells can be merged; and you can embed images and charts.
2. You pass a File object, obtained from a call to `open()`.
3. File objects need to be opened in read-binary (`'rb'`) for Reader objects and write-binary (`'wb'`) for Writer objects.
4. The `writerow()` method
5. The `delimiter` argument changes the string used to separate cells in a row. The `lineterminator` argument changes the string used to separate rows.
6. `json.loads()`
7. `json.dumps()`

## Chapter 15

1. A reference moment that many date and time programs use. The moment is January 1st, 1970, UTC.
2. `time.time()`
3. `time.sleep(5)`
4. It returns the closest integer to the argument passed. For example, `round(2.4)` returns 2.
5. A `datetime` object represents a specific moment in time. A `timedelta` object represents a duration of time.
6. `threadObj = threading.Thread(target=spam)`
7. `threadObj.start()`

8. Make sure that code running in one thread does not read or write the same variables as code running in another thread.
9. `subprocess.Popen('c:\\Windows\\System32\\calc.exe')`

## Chapter 16

1. SMTP and IMAP, respectively
2. `smtplib.SMTP()`, `smtpObj.ehlo()`, `smtpObj.starttls()`, and `smtpObj.login()`
3. `imapclient.IMAPClient()` and `imapObj.login()`
4. A list of strings of IMAP keywords, such as 'BEFORE <date>', 'FROM <string>', or 'SEEN'
5. Assign the variable `imaplib._MAXLINE` a large integer value, such as 10000000.
6. The `pyzmail` module reads downloaded emails.
7. You will need the Twilio account SID number, the authentication token number, and your Twilio phone number.

## Chapter 17

1. An RGBA value is a tuple of 4 integers, each ranging from 0 to 255. The four integers correspond to the amount of red, green, blue, and alpha (transparency) in the color.
2. A function call to `ImageColor.getcolor('CornflowerBlue', 'RGBA')` will return (100, 149, 237, 255), the RGBA value for that color.
3. A box tuple is a tuple value of four integers: the left edge x-coordinate, the top edge y-coordinate, the width, and the height, respectively.
4. `Image.open('zophie.png')`
5. `imageObj.size` is a tuple of two integers, the width and the height.
6. `imageObj.crop((0, 50, 50, 50))`. Notice that you are passing a box tuple to `crop()`, not four separate integer arguments.
7. Call the `imageObj.save('new_filename.png')` method of the Image object.
8. The `ImageDraw` module contains code to draw on images.
9. `ImageDraw` objects have shape-drawing methods such as `point()`, `line()`, or `rectangle()`. They are returned by passing the Image object to the `ImageDraw.Draw()` function.

## Chapter 18

1. Move the mouse to the top-left corner of the screen, that is, the (0, 0) coordinates.
2. `pyautogui.size()` returns a tuple with two integers for the width and height of the screen.

3. `pyautogui.position()` returns a tuple with two integers for the x- and y-coordinates of the mouse cursor.
4. The `moveTo()` function moves the mouse to absolute coordinates on the screen, while the `moveRel()` function moves the mouse relative to the mouse's current position.
5. `pyautogui.dragTo()` and `pyautogui.dragRel()`
6. `pyautogui.typewrite('Hello world!')`
7. Either pass a list of keyboard key strings to `pyautogui.typewrite()` (such as 'left') or pass a single keyboard key string to `pyautogui.press()`.
8. `pyautogui.screenshot('screenshot.png')`
9. `pyautogui.PAUSE = 2`



# INDEX

## Symbols

= (assignment) operator, 18, 34  
\  
  (backslash), 124, 151, 162, 174–175  
  line continuation character, 93  
^ (caret symbol), 162  
  matching beginning of string,  
  159–160  
  negative character classes, 159  
: (colon), 38, 45, 54, 82, 127  
{ } (curly brackets), 105, 162  
  greedy vs. nongreedy matching,  
  156–157  
  matching specific repetitions  
  with, 156  
\$ (dollar sign), 159–160, 162  
.  
  (dot character), 160–162  
  using in paths, 175–176  
  wildcard matches, 160–162  
" (double quotes), 124  
\*\* (exponent) operator, 15  
== (equal to) operator, 33, 34  
/  
  (forward slash), 174–175  
  division operator, 15, 88  
> (greater than) operator, 33  
>= (greater than or equal to)  
  operator, 33  
# (hash character), 126  
// (integer division/floored quotient)  
  operator, 15  
< (less than) operator, 33  
<= (less than or equal to) operator, 33  
% (modulus/remainder) operator,  
  15, 88  
\* (multiplication) operator, 15, 83, 88  
!= (not equal to) operator, 33  
( ) (parentheses), 96–97, 152–153  
| (pipe character), 153–154, 164–165  
+ (plus sign), 155–156, 162  
  addition operator, 15, 17, 83, 88  
? (question mark), 154–155, 162  
' (single quote), 124

[] (square brackets), 80, 162  
\* (star), 162  
  using with wildcard character, 161  
  zero or more matches with, 155  
- (subtraction) operator, 15, 88  
''' (triple quotes), 125, 164  
\_ (underscore), 20

## A

%A directive, 344  
%a directive, 344  
absolute paths, 175–179  
abspath() function, 177  
addition (+) operator, 15, 17, 83, 88  
additive color model, 389  
add\_heading() method, 314  
addPage() method, 299  
add\_paragraph() method, 313–314  
add\_picture() method, 315  
add\_run() method, 313–314  
algebraic chess notation, 112–113  
all\_caps attribute, 311  
ALL search key, 369  
alpha, defined, 388  
and operator, 35  
ANSWERED search key, 370  
API (application programming  
  interface), 327–328  
append() method, 89–90  
application-specific passwords, 365  
args keyword, 349  
arguments, function, 23, 63  
  keyword arguments, 65–66  
  passing to processes, 354  
  passing to threads, 348–349  
assertions, 219–221  
  disabling, 445  
assignment (=) operator, 18, 34  
AT&T mail, 363, 367  
attributes, HTML, 241, 248  
augmented assignment operators,  
  88–89

## B

- `\b` backspace escape character, 419
- `%B` directive, 344
- `%b` directive, 344
- `back()` method, 261
- backslash (`\`), 124, 151, 162, 174–175
- `BarChart()` function, 290
- `basename()` function, 178
- BCC search key, 370
- Beautiful Soup, 245. *See also* `bs4` module
- `BeautifulSoup` objects, 245–246
- BEFORE search key, 369
- binary files, 180–181, 184–185
- binary operators, 35–37
- bitwise or operator, 164–165
- blank strings, 17
- blocking execution, 337
- blocks of code, 37–38
- BODY search key, 369
- bold attribute, 311
- Boolean data type
  - binary operators, 35–36
  - flow control and, 32–33
  - in operator, 87
  - not in operator, 87
  - “truthy” and “falsey” values, 53
  - using binary and comparison operators together, 36–37
- box tuples, 390
- breakpoints, debugging using, 229–231
- break statements
  - overview, 49–50
  - using in for loop, 55
- browser, opening using `webbrowser` module, 234–236
- `bs4` module
  - creating object from HTML, 245–246
  - finding element with `select()` method, 246–247
  - getting attribute, 248
  - overview, 245
- built-in functions, 57
- bulleted list, creating in Wiki markup, 139–141
  - copying and pasting clipboard, 139–140
  - joining modified lines, 141
  - overview, 139
  - separating lines of text, 140

## C

- calling functions, 23
- call stack, defined, 217
- camelcase, 21
- caret symbol (`^`), 162
  - matching beginning of string, 159–160
  - negative character classes, 159
- Cascading Style Sheets (CSS)
  - matching with `selenium` module, 258
  - selectors, 246–247
- case sensitivity, 21, 163
- CC search key, 370
- Cell objects, 268–269
- cells, in Excel spreadsheets, 266
  - accessing Cell object by its name, 268–269
  - merging and unmerging, 286–287
  - writing values to, 278–279
- `center()` method, 133–134, 426
- chaining method calls, 398
- character classes, 158–159, 162
- character styles, 310
- charts, Excel, 288–290
- `chdir()` function, 175
- Chrome, developer tools in, 242–243
- `clear()` method, 258
- `click()` function, 420, 430, 431
- clicking mouse, 420
- `click()` method, 259
- clipboard, using string from, 236
- CMYK color model, 389
- colon (`:`), 38, 45, 54, 82, 127
- color values
  - CMYK vs. RGB color models, 389
  - RGBA values, 388–389
- `column_index_from_string()` function, 270
- columns, in Excel spreadsheets
  - setting height and width of, 285–286
  - slicing Worksheet objects to get Cell objects in, 270–272
- Comcast mail, 363, 367
- comma-delimited items, 80
- command line arguments, 235
- `commentAfterDelay()` function, 429
- comments
  - multiline, 126
  - overview, 23

- comparison operators
  - overview, 33–35
  - using binary operators with, 36–37
- compile() function, 151, 152, 164–165
- compressed files
  - backing up folder into, 209–212
  - creating ZIP files, 205–206
  - extracting ZIP files, 205
  - overview, 203–204
  - reading ZIP files, 204
- computer screen
  - coordinates of, 415
  - resolution of, 416
- concatenation
  - of lists, 83
  - string, 17–18
- concurrency issues, 349
- conditions, defined, 37
- continue statements
  - overview, 50–53
  - using in for loop, 55
- Coordinated Universal Time (UTC), 336
- coordinates
  - of computer screen, 415
  - of an image, 389–390
- copy() function, 100–101, 135, 198, 394
- copytree() function, 198–199
- countdown project, 357–358
  - counting down, 357
  - overview, 357
  - playing sound file, 357–358
- cProfile.run() function, 337
- crashes, program, 14
- create\_sheet() method, 278
- CRITICAL level, 224
- cron, 354
- cropping images, 393–394
- CSS (Cascading Style Sheets)
  - matching with selenium module, 258
  - selectors, 246–247
- CSV files
  - defined, 319
  - delimiter for, 324
  - format overview, 320
  - line terminator for, 324
  - Reader objects, 321
  - reading data in loop, 322
  - removing header from, 324–327
    - looping through CSV files, 325
    - overview, 324–325
    - reading in CSV file, 325–326
    - writing out CSV file, 326–327
  - Writer objects, 322–323

- curly brackets ({}), 105, 162
  - greedy vs. nongreedy matching, 156–157
  - matching specific repetitions with, 156
- current working directory, 175

## D

- \D character class, 158
- \d character class, 158
- %d directive, 344
- data structures
  - algebraic chess notation, 112–113
  - tic-tac-toe board, 113–117
- data types
  - Booleans, 32
  - defined, 16
  - dictionaries, 105–106
  - floating-point numbers, 17
  - integers, 17
  - list() function, 97
  - lists, 80
  - mutable vs. immutable, 94–96
  - None value, 65
  - strings, 17
  - tuple() function, 97
  - tuples, 96–97
- datetime module
  - arithmetic using, 343
  - converting objects to strings, 344–345
  - converting strings to objects, 345
  - fromtimestamp() function, 341
  - now() function, 341
  - overview, 341–342, 346
  - pausing program until time, 344
  - timedelta data type, 342–343
  - total\_seconds() method, 342
- datetime objects, 341–342
  - converting to strings, 344–345
  - converting from strings to, 345
- debug() function, 222
- debugging
  - assertions, 219–221
  - defined, 4
  - getting traceback as string, 217–218
  - in IDLE
    - overview, 225–227
    - stepping through program, 227–229
    - using breakpoints, 229–231

- debugging (*continued*)
    - logging
      - disabling, 224–225
      - to file, 225
      - levels of, 223–224
      - logging module, 221–223
      - print() function and, 223
    - raising exceptions, 216–217
  - DEBUG level, 223
  - decimal numbers. *See* floating-point numbers
  - decode() method, 374–375
  - decryption, of PDF files, 297–298
  - deduplicating code, 62
  - deepcopy() function, 100–101
  - def statements, 62
    - with parameters, 63
  - DELETED search key, 370
  - delete\_messages() method, 375
  - deleting files/folders
    - permanently, 200–201
    - using send2trash module, 201–202
  - del statements, 84
  - dictionaries
    - copy() function, 100–101
    - deepcopy() function, 100–101
    - get() method, 109
    - in operator, 109
    - items() method, 107–108
    - keys() method, 107–108
    - lists vs., 106–107
    - nesting, 117–119
    - not in operator, 109
    - overview, 105–106
    - setdefault() method, 110–111
    - values() method, 107–108
  - directories
    - absolute vs. relative paths, 175–176
    - backslash vs. forward slash, 174–175
    - copying, 198–199
    - creating, 176
    - current working directory, 175
    - defined, 173–174
    - deleting permanently, 200–201
    - deleting using send2trash module, 201–202
    - moving, 199–200
    - os.path module
      - absolute paths in, 177–179
      - file sizes, 179–180
      - folder contents, 179–180
      - overview, 177
      - path validity, 180
      - relative paths in, 177–179
      - renaming, 199–200
      - walking, 202–203
  - dirname() function, 178
  - disable() function, 224
  - division (/) operator, 15, 88
  - Document objects, 307–308
  - dollar sign (\$), 159–160, 162
  - dot character (.), 160–162
    - using in paths, 175–176
    - wildcard matches, 160–162
  - dot-star character (.\*), 161
  - doubleClick() function, 420, 430
  - double quotes ("), 124
  - double\_strike attribute, 311
  - downloading
    - files from web, 239–240
    - web pages, 237–238
    - XKCD comics, 251–256, 350–352
  - DRAFT search key, 370
  - dragging mouse, 420–422
  - dragRel() function, 420, 422, 430
  - dragTo() function, 420, 430
  - drawing on images
    - ellipses, 407
    - example program, 407–408
    - ImageDraw module, 406
    - lines, 406–407
    - points, 406
    - polygons, 407
    - rectangles, 407
    - text, 408–410
  - dumps() function, 329
  - duration keyword arguments, 416
- ## E
- ehlo() method, 364, 379
  - elements, HTML, 240
  - elif statements, 40–45
  - ellipse() method, 407
  - else statements, 39–40
  - email addresses, extracting, 165–169
    - creating regex, 166–167
    - finding matches on clipboard, 167–168
    - joining matches into a string, 168
    - overview, 165–166
  - emails
    - deleting, 375
    - disconnecting from server, 375–376
    - fetching



- folders, 368–369
    - getting message content, 372–373
    - logging into server, 368
    - overview, 366–367
    - raw messages, 373–375
  - gmail\_search() method, 372
  - IMAP, 366
  - marking message as read, 372–373
  - searching, 368–371
  - sending
    - connecting to SMTP server, 363–364
    - disconnecting from server, 366
    - logging into server, 364–365
    - overview, 362
    - reminder, 376–380
    - sending “hello” message, 364
    - sending message, 365
    - TLS encryption, 364
  - SMTP, 362
  - emboss attribute, 311
  - encryption, of PDF files, 302–303
  - endswith() method, 131
  - epoch timestamps, 336, 341, 346
  - equal to (==) operator, 33, 34
  - ERROR level, 224
  - errors
    - crashes and, 14
    - help for, 8–9
  - escape characters, 124–125
  - evaluation, defined, 14
  - Excel spreadsheets
    - application support, 265–266
    - charts in, 288–290
    - column width, 285–286
    - converting between column letters and numbers, 270
    - creating documents, 277
    - creating worksheets, 278
    - deleting worksheets, 278
    - font styles, 282–284
    - formulas in, 284–285
    - freezing panes, 287–288
    - getting cell values, 268–269
    - getting rows and columns, 270–272
    - getting worksheet names, 268
    - merging and unmerging cells, 286–287
    - opening documents, 267
    - openpyxl module, 266
    - overview, 266–267
    - reading files
      - overview, 272–273
      - populating data structure, 274–275
      - reading data, 273–274
      - writing results to file, 275–276
    - and reminder emails project, 376–380
    - row height, 285–286
    - saving workbooks, 277
    - updating, 279–281
      - overview, 279–280
      - setup, 280
    - workbooks vs., 266
    - writing values to cells, 278–279
  - Exception objects, 217
  - exceptions
    - assertions and, 219–221
    - getting traceback as string, 217–218
    - handling, 72–74
    - raising, 216–217
  - execution, program
    - defined, 31
    - overview, 38
    - pausing until specific time, 344
    - terminating program with sys.exit(), 58
  - exists() function, 180
  - exit codes, 353–354
  - expand keyword, 398
  - exponent (\*\*) operator, 15
  - expressions
    - conditions and, 37
    - in interactive shell, 14–16
  - expunge() method, 375
  - extensions, file, 173
  - extractall() method, 205
  - extracting ZIP files, 205
  - extract() method, 205
- ## F
- FailSafeException exception, 434
  - “falsey” values, 53
  - fetch() method, 371, 372–373
  - file editor, 21
  - file management
    - absolute vs. relative paths, 175–176
    - backslash vs. forward slash, 174–175
    - compressed files
      - backing up to, 209–212
      - creating ZIP files, 205–206

- file management (*continued*)
  - compressed files (*continued*)
    - extracting ZIP files, 205
    - overview, 203–204
    - reading ZIP files, 204
  - creating directories, 176
  - current working directory, 175
  - multiclipboard project, 191–193
  - opening files, 181–182
  - os.path module
    - absolute paths in, 177–179
    - file sizes, 179–180
    - folder contents, 179–180
    - overview, 177
    - path validity, 180
    - relative paths in, 177–179
  - overview, 173–174
  - paths, 173–174
  - plaintext vs. binary files, 180–181
  - reading files, 182–183
  - renaming files, date styles, 206–209
  - saving variables with pformat()
    - function, 185–186
  - send2trash module, 201–202
  - shelve module, 184–185
  - shutil module
    - copying files/folders, 198–199
    - deleting files/folders, 200–201
    - moving files/folders, 199–200
    - renaming files/folders, 199–200
  - walking directory trees, 202–203
  - writing files, 183–184
- filenames, defined, 173
- File objects, 182
- findall() method, 157–158
- find\_element\_by\_\* methods, 257–258
- find\_elements\_by\_\* methods, 257–258
- Firefox, developer tools in, 243
- FLAGGED search key, 370
- flipping images, 398–399
- float() function, 25–28
- floating-point numbers
  - integer equivalence, 27
  - overview, 17
  - rounding, 338
- flow control
  - binary operators, 35–36
  - blocks of code, 37–38
  - Boolean values and, 32–33
  - break statements, 49–50
  - comparison operators, 33–35
  - conditions, 37
  - continue statements, 50–53
  - elif statements, 40–45
  - else statements, 39–40
  - if statements, 38–39
  - overview, 31–32
  - using binary and comparison operators together, 36–37
  - while loops, 45–49
- folders
  - absolute vs. relative paths, 175–176
  - backing up to ZIP file, 209–212
    - creating new ZIP file, 211
    - figuring out ZIP filename, 210–211
  - walking directory tree, 211–212
- backslash vs. forward slash, 174–175
- copying, 198–199
- creating, 176
- current working directory, 175
- defined, 173–174
- deleting permanently, 200–201
- deleting using send2trash module, 201–202
- moving, 199–200
- os.path module
  - absolute paths in, 177–179
  - file sizes, 179–180
  - folder contents, 179–180
  - overview, 177
  - path validity, 180
  - relative paths in, 177–179
- renaming, 199–200
- walking directory trees, 202–203

- Font objects, 282–283
- font styles, in Excel spreadsheets, 282–284
- for loops
- overview, 53–56
- using dictionary items in, 108
- using lists with, 86
- format attribute, 392
- format\_description attribute, 392
- formData list, 434
- form filler project, 430–437
- overview, 430–431
- radio buttons, 435–436
- select lists, 435–436
- setting up coordinates, 432–434
- steps in process, 431
- submitting form, 436–437
- typing data, 434–435

- formulas, in Excel spreadsheets, 284–285
- forward() method, 261
- forward slash (/), 174–175
- FROM search key, 370
- fromtimestamp() function, 341, 346
- functions. *See also names of individual functions*
  - arguments, 23, 63
  - as “black box”, 72
  - built-in, 57
  - def statements, 63
  - exception handling, 72–74
  - keyword arguments, 65–66
  - None value and, 65
  - overview, 61–62
  - parameters, 63
  - return values, 63–65

## G

- get\_active\_sheet() method, 268
- get\_addresses() method, 374
- get\_attribute() method, 258
- getcolor() function, 388–389, 393
- get\_column\_letter() function, 270
- getcwd() function, 175
- get() function
  - overview, 109
  - requests module, 237
- get\_highest\_column() method, 269, 377
- get\_highest\_row() method, 269
- get\_payload() method, 374–375
- getpixel() function, 400, 423, 424
- get\_sheet\_by\_name() method, 268
- get\_sheet\_names() method, 268
- getsize() function, 179
- get\_subject() method, 374
- getText() function, 308–309
- GIF format, 392
- global scope, 70–71
- Gmail, 363, 365, 367
- gmail\_search() method, 372
- Google Maps, 234–236
- graphical user interface automation. *See* GUI (graphical user interface) automation
- greater than (>) operator, 33
- greater than or equal to (>=) operator, 33
- greedy matching
  - dot-star for, 161
  - in regular expressions, 156–157

- group() method, 151, 152–153
- groups, regular expression
  - matching
    - greedy, 156–157
    - nongreedy, 157
    - one or more, 155–156
    - optional, 154–155
    - specific repetitions, 156
    - zero or more, 155
  - using parentheses, 152–153
  - using pipe character in, 153–154
- Guess the Number program, 74–76
- GUI (graphical user interface)
  - automation. *See also* form filler project
  - controlling keyboard, 426–429
    - hotkey combinations, 429
    - key names, 427–428
    - pressing and releasing, 428–429
    - sending string from keyboard, 426–427
  - controlling mouse, 415–417, 419–423
    - clicking mouse, 420
    - dragging mouse, 420–422
    - scrolling mouse, 422–423
  - determining mouse position, 417–419
  - image recognition, 425–426
  - installing pyautogui module, 414
  - logging out of program, 414
  - overview, 413–414
  - screenshots, 423–424
  - stopping program, 414–415

## H

- %H directive, 344
- hash character (#), 126
- headings, Word document, 314–315
- help
  - asking online, 9–10
  - for error messages, 8–9
- hotkey combinations, 429
- hotkey() function, 429, 430
- Hotmail.com, 363, 367
- HTML (Hypertext Markup Language)
  - browser developer tools and, 242–243
  - finding elements, 244
  - learning resources, 240
  - overview, 240–241
  - viewing page source, 241–242

## I

- %I directive, 344
- id attribute, 241
- IDLE (interactive development environment)
  - creating programs, 21–22
  - debugging in
    - overview, 225–227
    - stepping through program, 227–229
    - using breakpoints, 229–231
  - expressions in, 14–16
  - overview, 8
  - running scripts outside of, 136
  - starting, 7–8
- if statements
  - overview, 38–39
  - using in while loop, 46–47
- imageDraw module, 406
- imageDraw objects, 406–408
- ImageFont objects, 408–410
- Image objects, 391–399
- images
  - adding logo to, 401–405
  - attributes for, 392–393
  - box tuples, 390
  - color values in, 388–389
  - coordinates in, 389–390
  - copying and pasting in, 394–396
  - cropping, 393–394
  - drawing on
    - example program, 407–408
    - ellipses, 407
    - ImageDraw module, 406
    - lines, 406–407
    - points, 406
    - polygons, 407
    - rectangles, 407
    - text, 408–410
  - flipping, 398–399
  - opening with Pillow, 390–391
  - pixel manipulation, 400
  - recognition of, 425–426
  - resizing, 397
  - RGBA values, 388–389
  - rotating, 398–399
  - transparent pixels, 397
- IMAP (Internet Message Access Protocol)
  - defined, 366
  - deleting messages, 375
  - disconnecting from server, 375–376
  - fetching messages, 372–375
  - folders, 368–369
  - logging into server, 368
  - searching messages, 368–371
- imapclient module, 366
- IMAPClient objects, 367–368
- immutable data types, 94–96
- importing modules
  - overview, 57–58
  - pyautogui module, 417
- imprint attribute, 311
- im variable, 423
- indentation, 93
- indexes
  - for dictionaries. *See* keys, dictionary
  - for lists
    - changing values using, 83
    - getting value using, 80–81
    - negative, 82
    - removing values from list using, 84
  - for strings, 126–127
- IndexError, 106
- index() method, 89
- infinite loops, 49, 51, 418
- INFO level, 223
- in operator
  - using with dictionaries, 109
  - using with lists, 87
  - using with strings, 127
- input() function
  - overview, 23–24, 89–90
  - using for sensitive information, 365
- installing
  - openpyxl module, 266
  - pyautogui module, 414
  - Python, 6–7
  - selenium module, 256
  - third-party modules, 441–442
- int, 17. *See* also integers
- integer division/floored quotient (//)
  - operator, 15
- integers
  - floating-point equivalence, 27
  - overview, 17
- interactive development environment. *See* IDLE (interactive development environment)
- interactive shell. *See* IDLE
- Internet Explorer, developer tools in, 242–243

Internet Message Access Protocol. *See*  
IMAP (Internet Message  
Access Protocol)

interpreter, Python, 7

int() function, 25–28

isabs() function, 177

isalnum() method, 129–131

isalpha() method, 129–130

isdecimal() method, 129–131

isdir() function, 180

is\_displayed() method, 258

is\_enabled() method, 258

isfile() function, 180

islower() method, 128–129

is\_selected() method, 258

isspace() method, 130

istitle() method, 130

isupper() method, 128–129

italic attribute, 311

items() method, 107–108

iter\_content() method, 239–240

## J

%j directive, 344

join() method, 131–132, 174–175,  
177, 352

JPEG format, 392

JSON files

APIs for, 327–328

defined, 319–320

format overview, 327–328

reading, 328–329

and weather data project, 329–332

writing, 329

justifying text, 133–134

## K

keyboard

controlling, with PyAutoGUI

hotkey combinations, 429

pressing and releasing keys,  
428–429

sending string from keyboard,  
426–427

key names, 427–428

KeyboardInterrupt exception, 340,  
417, 418

keyDown() function, 428, 429, 430

keys, dictionary, 105

keys() method, 107–108

keyUp() function, 428, 429, 430

keyword arguments, 65–66

## L

LARGER search key, 370

launchd, 354–355

launching programs

and countdown project, 357–358

opening files with default

applications, 355–356

opening websites, 355

overview, 352–354

passing command line arguments

to processes, 354

poll() method, 353

running Python scripts, 355

scheduling, 354–355

sleep() function, 355

wait() method, 354

len() function, 307–308

finding number of values in list, 83

overview, 24–25

less than (<) operator, 33

less than or equal to (<=) operator, 33

LibreOffice, 265, 306

line breaks, Word document, 315

LineChart() function, 290

line continuation character (\), 93

line() method, 406–407

linked styles, 310

Linux

backslash vs. forward slash, 174–175

cron, 354

installing Python, 7

installing third-party modules, 442

launching processes from  
Python, 353

logging out of automation  
program, 414

opening files with default  
applications, 355

pip tool on, 441–442

Python support, 4

running Python programs on, 445

starting IDLE, 8

Unix philosophy, 356

listdir() function, 179

list\_folders() method, 368–369

list() function, 321, 426

- lists
  - append() method, 89–90
  - augmented assignment operators, 88–89
  - changing values using index, 83
  - concatenation of, 83
  - copy() function, 100–101
  - deepcopy() function, 100–101
  - dictionaries vs., 106–107
  - finding number of values using len(), 83
  - getting sublists with slices, 82–83
  - getting value using index, 80–81
  - index() method, 89
  - in operator, 87
  - insert() method, 89–90
  - list() function, 97
  - Magic 8 Ball example program using, 92–93
  - multiple assignment trick, 87–88
  - mutable vs. immutable data types, 94–96
  - negative indexes, 82
  - nesting, 117–119
  - not in operator, 87
  - overview, 80
  - remove() method, 90–91
  - removing values from, 84
  - replication of, 83
  - sort() method, 91–92
  - storing variables as, 84–85
  - using with for loops, 86
- ljust() method, 133–134
- load\_workbook() function, 267
- loads() function, 328–329, 331
- local scope, 67–70
- locateAllOnScreen() function, 426
- locateOnScreen() function, 425
- location attribute, 258
- logging
  - disabling, 224–225
  - to file, 225
  - levels of, 223–224
  - print() function and, 223
- logging module, 221–223
- logging out, of automation
  - program, 414
- login() method, 364, 368, 379
- logo, adding to an image, 401–406
  - looping over files, 402–403
  - opening logo image, 401–402
  - overview, 404
  - resizing image, 403–404

- logout() method, 375–376
- LogRecord objects, 221
- loops
  - break statements, 49–50
  - continue statements, 50–53
  - for loop, 53–56
  - range() function for, 56–57
  - reading data from CSV file, 322
  - using lists with, 86
  - while loop, 45–49
- lower() method, 128–129
- lstrip() method, 134–135

## M

- %M directive, 344
- %m directive, 344
- Mac OS X. *See* OS X
- Magic 8 Ball example program, 92–93
- makedirs() function, 176, 403
- maps, open when location is copied, 234–236
  - figuring out URL, 234–235
  - handling clipboard content, 236
  - handling command line argument, 235–236
  - launching browser, 236
  - overview, 234
- Match objects, 151
- math
  - operators for, 15
  - programming and, 4
- mergePage() method, 302
- Message objects, 381–382
- methods
  - chaining calls, 398
  - defined, 89
  - dictionary
    - get() method, 109
    - items() method, 107–108
    - keys() method, 107–108
    - setdefault() method, 110–111
    - values() method, 107–108
  - list
    - append() method, 89–90
    - index() method, 89
    - insert() method, 89–90
    - remove() method, 90–91
    - sort() method, 91–92
  - string
    - center() method, 133–134
    - copy() method, 135
    - endswith() method, 131

- isalnum() method, 129–131
- isalpha() method, 129–130
- isdecimal() method, 129–131
- islower() method, 128–129
- isspace() method, 130
- istitle() method, 130
- isupper() method, 128–129
- join() method, 131–132
- ljust() method, 133–134
- lower() method, 128–129
- lstrip() method, 134–135
- paste() method, 135
- rjust() method, 133–134
- rstrip() method, 134–135
- split() method, 131–133
- startswith() method, 131
- strip() method, 134–135
- upper() method, 128–129
- Microsoft Windows. *See* Windows OS
- middleClick() function, 420, 430
- modules
  - importing, 57–58
  - third-party, installing, 442
- modulus/remainder (%) operator, 15, 88
- Monty Python, 4
- mouse
  - controlling, 415–417, 419–423
    - clicking mouse, 420
    - dragging mouse, 420–422
    - scrolling mouse, 422–423
  - determining position of, 417–419
  - locating, 417–419
    - getting coordinates, 418–419
    - handling KeyboardInterrupt exception, 418
    - importing pyautogui module, 418
    - infinite loop, 418
    - overview, 417
  - and pixels, identifying colors of, 424–425
- mouseDown() function, 420, 430
- mouse.position() function, 418
- mouseUp() function, 430
- move() function, 199–200
- moveRel() function, 416, 417, 420, 430
- moveTo() function, 416, 420, 430
- moving files/folders, 199–200
- multiclipboard project, 191–193
  - listing keywords, 193
  - loading keyword content, 193
  - overview, 191

- saving clipboard content, 192
  - setting up shelf file, 192
- multiline comments, 126
- multiline strings, 125–126
- multiple assignment trick, 87–88
- multiplication (\*) operator, 15, 83, 88
- multithreading
  - concurrency issues, 349
  - downloading multiple images, , 350–352
    - creating and starting threads, 351–352
    - using downloadXkcd() function, 350–351
    - waiting for threads to end, 352
  - join() method, 352
  - overview, 347–348
  - passing arguments to threads, 348–349
  - start() method, 348, 349
  - Thread() function, 347–348
- mutable data types, 94–96

## N

- NameError, 84
- namelist() method, 204
- negative character classes, 159
- negative indexes, 82
- nested lists and dictionaries, 117–119
- newline keyword argument, 322
- None value, 65
- nongreedy matching
  - dot, star, and question mark for, 161
  - in regular expressions, 157
- not equal to (!=) operator, 33
- not in operator
  - using with dictionaries, 109
  - using with lists, 87
  - using with strings, 127
- not operator, 36
- NOT search key, 370
- now() function, 341, 346

## O

- ON search key, 369
- open() function, 181–182, 234, 355–356, 391
- opening files, 181–182
- OpenOffice, 265, 306
- open program, 355

- openpyxl module, installing, 266
- operators
  - augmented assignment, 88–89
  - binary, 35–36
  - comparison, 33–35
  - defined, 14
  - math, 15
  - using binary and comparison
    - operators together, 36–37
- order of operations, 15
- or operator, 36
- OR search key, 370
- OS X
  - backslash vs. forward slash, 174–175
  - installing Python, 7
  - installing third-party modules, 442
  - launchd, 354
  - launching processes from
    - Python, 356
  - logging out of automation
    - program, 414
  - opening files with default
    - applications, 355–356
  - pip tool on, 441–442
  - Python support, 4
  - running Python programs on, 445
  - starting IDLE, 8
  - Unix philosophy, 356
- outline attribute, 311
- Outlook.com, 363, 367

## P

- %p directive, 344
- page breaks, Word document, 315
- Page objects, 297
- Paragraph objects, 307
- paragraphs, Word document, 309–310
- parameters, function, 63
- parentheses (), 96–97, 152–153
- parsing, defined, 245
- passing arguments, 23
- passing references, 100
- passwords
  - application-specific, 365
  - managing project, 136–138
    - command-line arguments, 137
    - copying password, 137–138
    - data structures, 136–137
    - overview, 136
- pastebin.com, 10
- paste() method, 135, 394, 395

- paths
  - absolute vs. relative, 175–176
  - backslash vs. forward slash, 174–175
  - current working directory, 175
  - overview, 173–174
  - os.path module
    - absolute paths in, 177–179
    - file sizes, 179–180
    - folder contents, 179–180
    - overview, 177
    - path validity, 180
    - relative paths in, 177–179
- PAUSE variable, 415, 434
- PdfFileReader objects, 297–298
- PDF files
  - combining pages from multiple
    - files, 303–306
    - adding pages, 303
    - finding PDF files, 304
    - opening PDFs, 304–305
    - overview, 303
    - saving results, 305–306
  - creating, 298–299
  - decrypting, 297–298
  - encrypting, 302–303
  - extracting text from, 296–297
  - format overview, 295–296
  - pages in
    - copying, 299–300
    - overlying, 301–302
    - rotating, 300–301
- PdfFileWriter objects, 298–299
- pformat() function
  - overview, 111–112
  - saving variables in text files using, 185–186
- phone numbers, extracting, 165–169
  - creating regex, 166
  - finding matches on clipboard, 167–168
  - joining matches into a
    - string, 168
  - overview, 165–166
- Pillow
  - copying and pasting in images, 394–396
  - cropping images, 393–394
  - drawing on images
    - ellipses, 407
    - example program, 407–408
    - ImageDraw module, 406
    - lines, 406–407



- points, 406
  - polygons, 407
  - rectangles, 407
  - text, 408–410
- flipping images, 398–399
- image attributes, 392–393
- module, 388
- opening images, 390–391
- pixel manipulation, 400
- resizing images, 397
- rotating images, 398–399
- transparent pixels, 397
- pipe character (`|`), 153–154, 164–165
- pip tool, 441–442
- `pixelMatchesColor()` function, 424, 435
- pixels, 388, 400
- plaintext files, 180–181
- plus sign (+), 155–156, 162
- PNG format, 392
- `point()` method, 406
- `poll()` method, 353
- `polygon()` method, 407
- `Popen()` function, 352–353
  - opening files with default applications, 355–356
  - passing command line arguments to, 354
- `position()` function, 417, 419
- `pprint()` function, 111–112
- precedence of math operators, 15
- `press()` function, 429, 430, 436
- `print()` function, 435
  - logging and, 223
  - overview, 23
  - passing multiple arguments to, 66
  - using variables with, 24
- processes
  - and countdown project, 357–358
  - defined, 352
  - opening files with default applications, 355–356
  - opening websites, 355
  - passing command line arguments to, 354
  - `poll()` method, 353
  - `Popen()` function, 352–353
  - `wait()` method, 354
- profiling code, 336–337
- programming
  - blocks of code, 37–38
  - comments, 23
  - creativity needed for, 5
  - deduplicating code, 62
  - defined, 3
  - exception handling, 72–74
  - execution, program, 38
  - functions as “black boxes”, 72
  - global scope, 70–71
  - indentation, 93
  - local scope, 67–70
  - math and, 4
  - Python, 4
  - terminating program with `sys.exit()`, 58
- projects
  - Adding Bullets to Wiki Markup, 139–141
  - Adding a Logo, 401–406
  - Automatic Form Filler, 430–437
  - Backing Up a Folder into a ZIP File, 209–212
  - Combining Select Pages from Many PDFs, 303–306
  - Downloading All XKCD Comics, 251–256
  - Extending the mouseNow Program, 424–425
  - Fetching Current Weather Data, 329–332
  - Generating Random Quiz Files, 186–191
  - “I’m Feeling Lucky” Google Search, 248–251
  - “Just Text Me” Module, 383
  - mapIt.py* with the webbrowser Module, 234–236
  - Multiclipboard, 191–193
  - Multithreaded XKCD Downloader, 350–352
  - Password Locker, 136–138
  - Phone Number and Email Address Extractor, 165–169
  - Reading Data from a Spreadsheet, 272–276
  - Removing the Header from CSV Files, 324–327
  - Renaming Files with American-Style Dates to European-Style Dates, 206–209
  - Sending Member Dues Reminder Emails, 376–379
  - Simple Countdown Program, 357–358
  - Super Stopwatch, 338–340

- projects (*continued*)
  - Updating a Spreadsheet, 279–281
  - “Where Is the Mouse Right Now?”, 417–419
- putpixel() method, 400
- pyautogui.click() function, 431
- pyautogui.click() method, 420
- pyautogui.doubleClick() function, 420
- pyautogui.dragTo() function, 420
- pyautogui.FailSafeException
  - exception, 415
- pyautogui.hotkey() function, 429
- pyautogui.keyDown() function, 428
- pyautogui.keyUp() function, 428
- pyautogui.middleClick() function, 420
- pyautogui module
  - form filler project, 430–437
    - controlling keyboard, 426–429
      - hotkey combinations, 429
      - key names, 427–428
      - pressing and releasing keys, 428–429
    - sending string from keyboard, 426–427
  - controlling mouse, 415–417, 419–423
    - clicking mouse, 420
    - dragging mouse, 420–422
    - scrolling mouse, 422–423
  - documentation for, 414
  - fail-safe feature, 415
  - functions, 430
  - image recognition, 425–426
  - importing, 417
  - installing, 414
  - pausing function calls, 415
  - screenshots, 423–424
- pyautogui.mouseDown() function, 420
- pyautogui.moveRel() function, 416, 417
- pyautogui.moveTo() function, 416
- pyautogui.PAUSE variable, 415
- pyautogui.position() function, 419
- pyautogui.press() function, 436
- pyautogui.rightClick() function, 420
- pyautogui.screenshot() function, 423
- pyautogui.size() function, 416
- pyautogui.typewrite() function, 426, 427, 431
- py.exe program, 444–445
- pyobjc module, 442

- PyPDF2 module
  - combining pages from multiple PDFs, 303–306
  - creating PDFs, 298–299
  - decrypting PDFs, 297–298
  - encrypting PDFs, 302–303
  - extracting text from PDFs, 296–297
  - format overview, 295–296
  - pages in PDFs
    - copying, 299–300
    - overlying, 301–302
    - rotating, 300–301
- pyperclip module, 135
- Python
  - data types, 16–17
  - downloading, 6
  - example program, 21–22
  - help, 8–9
  - installing, 6–7
  - interactive shell, 8
  - interpreter, defined, 7
  - math and, 4
  - overview, 4
  - programming overview, 3
  - starting IDLE, 7–8
- python-docx module, 306
- pyzmail module, 366, 373–375
- PyzMessage objects, 373–375

## Q

- question mark (?), 154–155, 162
- quit() method, 261, 366, 379
- quiz generator, 186–190
  - creating quiz file, 188
  - creating answer options, 189
  - overview, 186
  - shuffling question order, 188
  - storing quiz data in dictionary, 187
  - writing content to files, 189–191

## R

- radio buttons, 435–436
- raise\_for\_status() method, 238
- raise keyword, 216–217
- range() function, 56–57
- raw strings, 125, 151
- Reader objects, 321–322
- reading files, 182–183, 204
- readlines() method, 182

- read() method, 182
- rectangle() method, 407
- Reddit, 9
- Reference objects, 289
- references
  - overview, 97–99
  - passing, 100
- refresh() method, 261
- Regex objects
  - creating, 150
  - matching, 151
- regular expressions
  - beginning of string matches, 159–160
  - case sensitivity, 163
  - character classes, 158–159
  - creating Regex objects, 150–151
  - defined, 147–148
  - end of string matches, 159–160
  - extracting phone numbers and emails addresses, 165–169
  - findall() method, 157–158
  - finding text without, 148–150
  - greedy matching, 156–157
  - grouping
    - matching specific repetitions, 156
    - one or more matches, 155–156
    - optional matching, 154–155
    - using parentheses, 152–153
    - using pipe character in, 153–154
    - zero or more matches, 155
  - HTML and, 243
  - matching with, 151–152
  - multiple arguments for compile() function, 164–165
  - nongreedy matching, 157
  - patterns for, 150
  - spreading over multiple lines, 164
  - substituting strings using, 163–164
  - symbol reference, 162
  - wildcard character, 160–162
- relative paths, 175–179
- relpath() function, 177, 178
- remainder/modulus (%) operator, 15, 88
- remove() method, 90–91
- remove\_sheet() method, 278
- renaming files/folders, 199–200
  - date styles, 206–209
    - creating regex for dates, 206
    - identifying dates in filenames, 207–208
    - overview, 206
    - renaming files, 209
- replication
  - of lists, 83
  - string, 18
- requests module
  - downloading files, 239–240
  - downloading pages, 237–238
- resolution of computer screen, 416
- Response objects, 237–238
- return values, function, 63–65
- reverse keyword, 91
- RGBA values, 388–389
- RGB color model, 389
- rightClick() function, 420, 430
- rjust() method, 133–134, 419
- rmdir() function, 200
- rmtree() function, 200
- rotateClockwise() method, 300
- rotateCounterClockwise() method, 300
- rotating images, 398–399
- rounding numbers, 338
- rows, in Excel spreadsheets
  - setting height and width of, 285–286
  - slicing Worksheet objects to get Cell objects in, 270–272
- rstrip() method, 134–135
- rtl attribute, 311
- Run objects, 310, 311–312
- running programs
  - on Linux, 445
  - on OS X, 445
  - overview, 443
  - on Windows, 444–445
  - shebang line, 443–444

## S

- \s character class, 158
- \s character class, 158
- %s directive, 344
- Safari, developer tools in, 243
- save() method, 391
- scope
  - global, 70–71
  - local, 67–70
- screenshot() function, 423, 430
- screenshots
  - analyzing, 424
  - getting, 423
- scripts
  - running from Python program, 355
  - running outside of IDLE, 136

- scroll() function, 422, 423, 430
- scrolling mouse, 422–423
- searching
  - email, 368–371
  - the Web, 248–251
    - finding results, 249–250
    - getting command line arguments, 249
    - opening web browser for results, 250–251
    - overview, 248
    - requesting search page, 249
- search() method, 151
- SEEN search key, 370
- see program, 355
- select\_folder() method, 369
- select lists, 435–436
- select() method, bs4 module, 246–247
- selectors, CSS, 246–247, 258
- selenium module
  - clicking buttons, 261
  - finding elements, 257–259
  - following links, 259
  - installing, 256
  - sending special keystrokes, 260–261
  - submitting forms, 259–260
  - using Firefox with, 256–257
- send2trash module, 201–202
- sending reminder emails, 376–379
  - finding unpaid members, 378
  - opening Excel file, 376–377
  - overview, 376
  - sending emails, 378–379
- send\_keys() method, 259–260
- sendmail() method, 365, 379
- sequence numbers, 373
- sequences, 86
- setdefault() method, 110–111
- shadow attribute, 311
- shebang line, 443–444
- shelve module, 184–185
- Short Message Service (SMS)
  - sending messages, 381–382
  - Twilio service, 380
- shutil module
  - deleting files/folders, 200–201
  - moving files/folders, 199–200
  - renaming files/folders, 199–200
- SID (string ID), 382
- Simple Mail Transfer Protocol. *See* SMTP (Simple Mail Transfer Protocol)
- SINCE search key, 369
- single quote ('), 124
- single-threaded programs, 347
- size() function, 416
- sleep() function, 337–338, 344, 346, 355
- slices
  - getting sublists with, 82–83
  - for strings, 126–127
- small\_caps attribute, 311
- SMALLER search key, 370
- SMS (Short Message Service)
  - sending messages, 381–382
  - Twilio service, 380
- SMTP (Simple Mail Transfer Protocol)
  - connecting to server, 363–364
  - defined, 362
  - disconnecting from server, 366
  - logging into server, 364–365
  - sending “hello” message, 364
  - sending message, 365
  - TLS encryption, 364
- SMTP objects, 363–364
- sort() method, 91–92
- sound files, playing, 357–358
- source code, defined, 3
- split() method, 131–133, 178, 320
- spreadsheets. *See* Excel spreadsheets
- square brackets [], 80
- Stack Overflow, 9
- standard library, 57
- star (\*), 161, 162
  - using with wildcard character, 161
  - zero or more matches with, 155
- start() method, 348, 349, 351
- start program, 355
- startswith() method, 131
- starttls() method, 364, 379
- step argument, 56
- stopwatch project, 338–340
  - overview, 338–339
  - set up, 339
  - tracking lap times, 339–340
- strftime() function, 344–345, 346
- str() function, 25–28, 97, 419
- strike attribute, 311
- string ID (SID), 382
- strings
  - center() method, 133–134
  - concatenation, 17–18
  - converting datetime objects to, 344–345
  - converting to datetime objects, 345

- copying and pasting, 135
- double quotes for, 124
- endswith() method, 131
- escape characters, 124–125
- extracting PDF text as, 296–297
- getting traceback as, 217–218
- indexes for, 126–127
- in operator, 127
- isalnum() method, 129–131
- isalpha() method, 129–130
- isdecimal() method, 129–131
- islower() method, 128–129
- isspace() method, 130
- istitle() method, 130
- isupper() method, 128–129
- join() method, 131–132
- literals, 124
- ljust() method, 133–134
- lower() method, 128–129
- lstrip() method, 134–135
- multiline, 125–126
- mutable vs. immutable data types, 94–96
- not in operator, 127
- overview, 17
- raw, 125
- replication of, 18
- rjust() method, 133–134
- rstrip() method, 134–135
- slicing, 126–127
- split() method, 131–133
- startswith() method, 131
- strip() method, 134–135
- substituting using regular expressions, 163–164
- upper() method, 128–129
- strip() method, 134–135
- strptime() function, 345, 346
- strs, 17, *See also* strings
- Style objects, 282–283
- SUBJECT search key, 369
- sublists, getting with slices, 82–83
- sub() method, 163–164
- submitButtonColor variable, 432, 435
- submitButton variable, 432
- submit() method, 260
- subprocess module, 335, 352–354
- subtraction (-) operator, 15, 88
- subtractive color model, 389
- Sudoku puzzles, 4
- sys.exit() function, 58

## T

- tag\_name attribute, 258
- Tag objects, 246–247
- tags, HTML, 240
- Task Scheduler, 354
- termination, program, 22, 58
- text attribute, 308, 311
- text messaging
  - automatic notifications, 383
  - sending messages, 381–382
  - Twilio service, 380
- text() method, 408–410
- TEXT search key, 369
- textsize() method, 409
- third-party modules, installing, 441–442
- Thread() function, 347–348, 351
- threading module, 335, 347
- Thread objects, 347–348
- threads
  - concurrency issues, 349
  - join() method, 352
  - multithreading, 347–348
    - image downloader, 350–352
  - passing arguments to, 348–349
  - processes vs., 352
- tic-tac-toe board, 113–117
- timedelta data type, 342–343, 346
- timedelta objects, 342–343
- time module
  - overview, 346
  - sleep() function, 337–338, 344
  - stopwatch project, 338–340
  - time() function, 336–337
- TLS encryption, 364
- top-level domains, 167
- T0 search key, 370
- total\_seconds() method, 342, 346
- traceback, getting from error, 217–218
- transparency, 388, 397
- transpose() method, 399
- triple quotes (""), 125, 164
- truetype() function, 409
- truth tables, 35–36
- “truthy” values, 53
- tuple data type
  - overview, 96–97
  - tuple() function, 97
- twilio module, 380
- TwilioRestClient objects, 381

- Twilio service
  - automatic text messages, 383
  - overview, 380
  - sending text messages, 381–382
- `TypeError`, 81, 94
- `typewrite()` function, 426, 427, 430, 431, 435, 436

## U

- Ubuntu, 7
  - cron, 354–355
  - launching processes from Python, 353
  - opening files with default applications, 355
  - Unix philosophy, 356
- UNANSWERED search key, 370
- UNDELETED search key, 370
- underline attribute, 311
- underscore (`_`), 20
- UNDRAFT search key, 370
- UNFLAGGED search key, 370
- Unicode encodings, 239
- Unix epoch, 336, 341, 346
- Unix philosophy, 356
- `unlink()` function, 200
- UNSEEN search key, 370
- `upper()` method, 128–129
- UTC (Coordinated Universal Time), 336

## V

- `ValueError`, 88, 345
- values, defined, 14, 150
- `values()` method, 107–108
- variables. *See also* lists
  - assignment statements, 18–19
  - defined, 18
  - global, 70–71
  - initializing, 19
  - local, 67–70
  - naming, 20–21
  - `None` value and, 65
  - overwriting, 19–20
  - references, 97–99
  - saving with `shelve` module, 184–185
  - storing as list, 84–85
- Verizon mail, 363, 367
- volumes, defined, 174

## W

- `\w` character class, 158
- `\W` character class, 158
- `%w` directive, 344
- `walk()` function, 202–203, 354
- WARNING level, 223
- weather data, fetching, 329–332
  - downloading JSON data, 330–331
  - getting location, 330
  - loading JSON data, 331–332
  - overview, 329
- webbrowser module
  - `open()` function, 355
  - opening browser using, 234–236
- `WebDriver` objects, 257
- `WebElement` objects, 257–258
- web scraping
  - `bs4` module
    - creating object from HTML, 245–246
    - finding element with `select()` method, 246–247
    - getting attribute, 248
    - overview, 245
  - downloading
    - files, 239–240
    - images, 251–256
    - pages, 237–238
  - and Google maps project, 234–236
  - and Google search project, 248–251
  - HTML
    - browser developer tools and, 242–243
    - finding elements, 244
    - learning resources, 240
    - overview, 240–241
    - viewing page source, 241–242
- overview, 233–234
- requests module, 237–238
- selenium module
  - clicking buttons, 261
  - finding elements, 257–259
  - following links, 259
  - installing, 256
  - sending special keystrokes, 260–261
  - submitting forms, 259–260
  - using Firefox with, 256–257
- websites, opening from script, 355

- while loops
  - getting and printing mouse coordinates using, 418
  - infinite, 418
  - overview, 45–49
- whitespace, removing, 134–135
- wildcard character (.), 160–162
- Windows OS
  - backslash vs. forward slash, 174–175
  - installing Python, 6–7
  - installing third-party modules, 442
  - launching processes from Python, 353
  - logging out of automation program, 414
  - opening files with default applications, 355–356
  - pip tool on, 441–442
  - Python support, 4
  - running Python programs on, 444–445
  - starting IDLE, 7
  - Task Scheduler, 354
- Word documents
  - adding headings, 314–315
  - creating documents with nondefault styles, 310–311
  - format overview, 306–307
  - getting text from, 308–309
  - line/page breaks, 315
  - pictures in, 315–316
  - python-docx module, 306
  - reading, 307–308
  - Run object attributes, 311–312
  - styling paragraphs, 309–310
  - writing to file, 312–314
- Workbook objects, 267
- workbooks, Excel, 266
  - creating worksheets, 278
  - deleting worksheets, 278
  - opening, 267
  - saving, 277
- Worksheet objects, 268
- write() method, 183–184
- Writer objects, 322–323
- writerow() method, 323

## X

- XKCD comics
  - downloading project, 251–256
  - designing program, 252–253
  - downloading web page, 253–254
  - overview, 251–252
  - saving image, 255–256
- multithreaded downloading
  - project, 350–352
  - creating and starting threads, 351–352
  - using downloadXkcd() function, 350–351
  - waiting for threads to end, 352

## Y

- %Y directive, 344
- %y directive, 344
- Yahoo! Mail, 363, 367

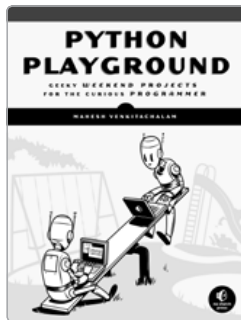
## Z

- zipfile module
  - creating ZIP files, 205–206
  - extracting ZIP files, 205
  - and folders, 209–212
  - overview, 203–204
  - reading ZIP files, 204
- ZipFile objects, 204–205
- ZipInfo objects, 204

# RESOURCES

Visit <http://nostarch.com/automatestuff/> for resources, errata, and more information.

More no-nonsense books from  **NO STARCH PRESS**



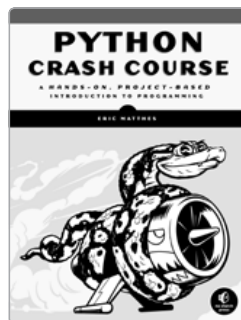
## **PYTHON PLAYGROUND**

**Geeky Weekend Projects  
for the Curious Programmer**

by MAHESH VENKITACHALAM

OCTOBER 2015, 352 PP., \$29.95

ISBN 978-1-59327-604-1



## **PYTHON CRASH COURSE**

**A Hands-On, Project-Based  
Introduction to Programming**

by ERIC MATTHES

NOVEMBER 2015, 560 PP., \$39.95

ISBN 978-1-59327-603-4



## **THE LINUX COMMAND LINE**

**A Complete Introduction**

by WILLIAM E. SHOTTS, JR.

JANUARY 2012, 480 PP., \$39.95

ISBN 978-1-59327-389-7



## **JAVASCRIPT FOR KIDS**

**A Playful Introduction to Programming**

by NICK MORGAN

DECEMBER 2014, 336 PP., \$34.95

ISBN 978-1-59327-408-5

*full color*



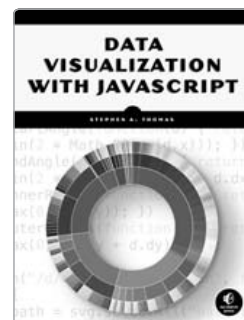
## **STATISTICS DONE WRONG**

**The Woefully Complete Guide**

by ALEX REINHART

MARCH 2015, 176 PP., \$24.95

ISBN 978-1-59327-620-1



## **DATA VISUALIZATION WITH JAVASCRIPT**

by STEPHEN A. THOMAS

MARCH 2015, 384 PP., \$39.95

ISBN 978-1-59327-605-8

*full color*

**PHONE:**  
800.420.7240 OR  
415.863.9900

**EMAIL:**  
SALES@NOSTARCH.COM  
**WEB:**  
WWW.NOSTARCH.COM