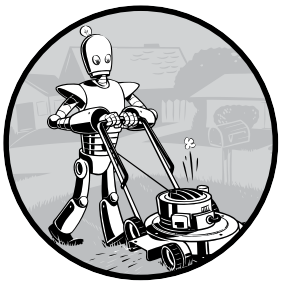
A

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.

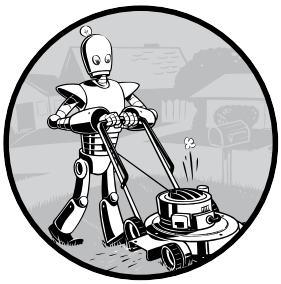
You can install all of the modules covered in this book by running the commands listed next. (Remember to replace `pip` with `pip3` if you're on OS X or Linux.)

- `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 autocomplete after you've begun to type it. The Environment Variables window that appears will look like Figure B-1.

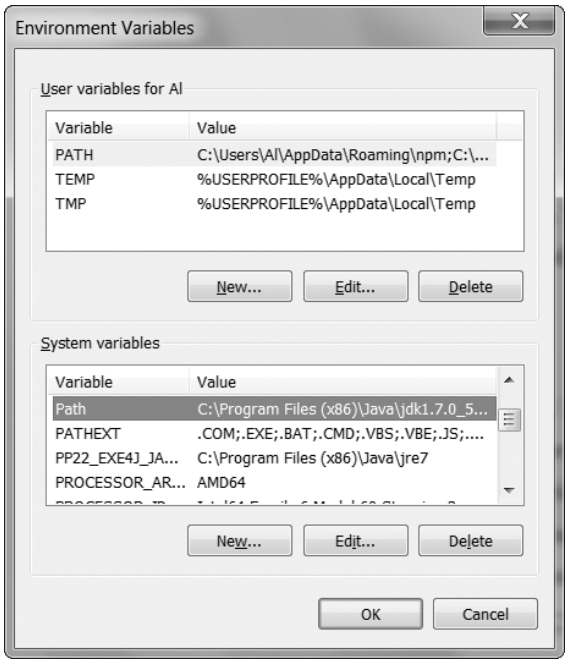


Figure B-1: The Environment Variables window on Windows

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.

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.

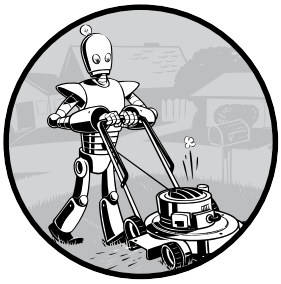
To run your Python programs, save your *.py* file to your home folder. Then, change the *.py* file's permissions to make it executable by running `chmod +x pythonScript.py`. File permissions are beyond the scope of this book, but you will need to run this command on your Python file if you want to run 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.

C

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, floating-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()`, `smptObj.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

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

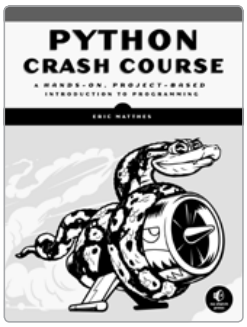
Z

RESOURCES

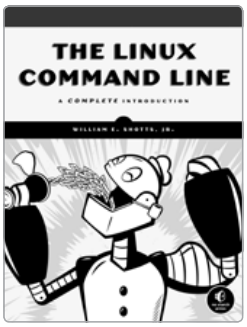
Visit *http://nostarch.com/automatestuff/* for resources, errata, and more information.

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

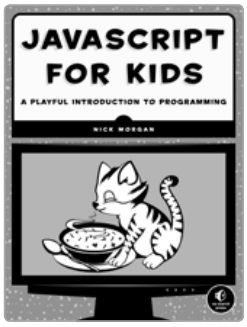
PYTHON PLAYGROUND
Geeky Weekend Projects for the Curious Programmer
by MAHESH VENKITACHALAM
OCTOBER 2015, 352 PP., $29.95
ISBN 978-1-59327-604-1



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

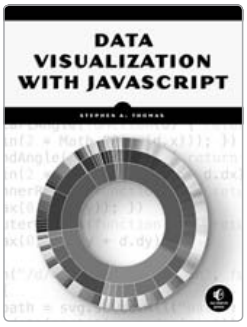


THE LINUX COMMAND LINE
A Complete Introduction
by WILLIAM E. SHOTTS, JR.
JANUARY 2012, 480 PP., $39.95
ISBN 978-1-59327-389-7



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