FREE

Exploring Python Basics

Chapters selected by Naomi Ceder

Exploring Python Basics

Selected by Naomi Ceder

Manning Author Picks

For online information and ordering of these and other Manning books, please visit www.manning.com. The publisher offers discounts on these books when ordered in quantity.

For more information, please contact

Special Sales Department
Manning Publications Co.
20 Baldwin Road
PO Box 761
Shelter Island, NY 11964
Email: orders@manning.com

Manning Publications Co.
20 Baldwin Road Technical
PO Box 761
Shelter Island, NY 11964

Cover designer: Leslie Haimes

ISBN: 9781617296581
Printed in the United States of America
1 2 3 4 5 6 7 8 9 10 - EBM - 23 22 21 20 19 18

contents

Exploring Python – the basics

Getting started in programming is confusing—what language should you learn? What approach makes learning easiest? Where do you start?

There are many answers to these questions, and none of them is right for every person in every situation. There are, however, some choices that are highly recommended and that have past successes to back them up.

This ebook will let you explore the basics of programming with one of the most popular languages in the world right now, Python. Python is popular in a number of areas—it powers the Django web framework, which makes it easy to create database powered web sites; it's an all purpose work horse on the back end, gluing programs together and connecting data pipelines; with tools like pandas, numpy, and tensorflow (and many, many others) it's a star in data science; and combined with the tiny (but mighty) raspberry pi and other boards it puts the power of computing literally in the palm of your hand.

This sampler brings to together chapters from three Manning books to help you explore the basics of this powerful language and decide if Python is the language for you. It starts with picking up the basics of coding and how to think like a coder from Ana Bell's *Get Programming*.

After that we move to the world of makers with *Hello Raspberry Pi* by Ryan Heitz. These chapters introduce the truly inexpensive raspberry pi computer and show how to wire up LED's to it and control them with simple Python code. Finally, for a more in depth look at "The Absolute Basics" of Python there are two chapters from my book, *The Quick Python Book*.

Each of these three books represents a different, but tried and tested, approach to learning to code with Python. The selections that follow will help you decide which of them is right for you.

Learning how to program

One of the problems people have in learning to code is figuring out how to start and how to even think about the process. This ebook starts with some chapters from Ana Bell's *Get Programming* which walk you through how to think about programming, how to install Python, and then how to write your first program.

Those topics actually cover quite a bit of ground, and Ana's book is very thoughtfully constructed to help beginners not only write their first bits of code, but also to guide them in how think like a programmer.

LESSON 2

Lesson 2 from *Get Programming* by Ana Bell

BASIC PRINCIPLES OF LEARNING A PROGRAMMING LANGUAGE

After reading lesson 2, you'll be able to

- Understand the process of writing a computer program
- Get a big-picture view of the think-code-test-debug-repeat paradigm
- Understand how to approach a programming problem
- Understand what it means to write readable code

2.1 Programming as a skill

Like reading, counting, playing piano, or playing tennis, programming is a skill. As with any skill, you have to nurture it through lots of practice. Practice requires dedication, perseverance, and self-discipline on your part. At the beginning of your programming career, I highly recommend that you write out as much code as possible. Open your code editor and type up every piece of code that you see. Try to type it out instead of relying on copying and pasting. At this point, the goal is to make programming become second nature, not to program quickly.

This lesson serves as motivation to get you in the mindset of a programmer. The first lesson introduced you to the "Thinking like a programmer" boxes that will be scattered throughout this book. The following sections offer a big-picture view encapsulating the main ideas of those boxes.

Consider this You want to teach a cave dweller how to get dressed to go to a job interview. Assume the clothes are already laid out and that the dweller is familiar with clothes, just not the process of dressing up. What steps do you tell him to take? Be as specific as possible.

Answer:

1. Pick up underwear, put left foot in one hole, put right foot in the other hole, and pull them up.
2. Pick up shirt, put one arm through one sleeve, and then put your other arm in the other sleeve. The buttons should be on your chest. Close shirt by inserting the little buttons into the holes.
3. Pick up pants, put one foot on one pant leg, and the other foot in the other pant leg. The pant opening should be in the front. Pull zipper up and button the pants.
4. Take one sock and put it on one foot. Then put a shoe on. Pull the laces of the shoes and tie them. Repeat with the other sock and shoe.

2.2 A parallel with baking

Suppose I ask you to bake me a loaf of bread. What is the process you go through—from the time I give you the task, to when you give me the finished loaf?

2.2.1 Understand the task "bake a loaf of bread"

The first step is to make sure you understand the given task. "Bake a loaf of bread" is a bit vague. Here are some questions you may want to ask to clarify the task:

- What size loaf of bread?
- Should it be a simple bread or flavored bread? Are there any specific ingredients you have to use or not use? Are there any ingredients you don't have?
- What equipment do you need? Is the equipment supplied to you, or do you need to get your own?
- Is there a time limit?
- Are there any recipes that you can look up and use, or do you have to make one up on your own?

It's important that you get these details right in order to avoid having to start over on the task. If no further details on the task are provided, the solution you come up with should be as simple as possible and should be as little work for you as possible. For example, you should look up a simple recipe instead of trying to come up with the

correct combination of ingredients on your own. As another example, first try to bake a small loaf of bread, don't add any flavors or spices, and use a bread machine (if you have one) to save time.

2.2.2 Find a recipe

After you clarify any questions or misconceptions about the task, you can look up a recipe or come up with one on your own. The recipe tells you how to do the task. Coming up with a recipe on your own is the hardest part of doing the task. When you have a recipe to follow, putting everything together shouldn't be difficult.

Take a quick look at any recipe right now. Figure 2.1 shows a sample recipe. A recipe may include the following:

- The steps you should take and in what order
- Specific measurements
- Instructions on when and how many times to repeat a task
- The substitutions you can make for certain ingredients
- Any finishing touches on the dish and how to serve it

Quantity	Ingredient
1/4 ounce	Active dry yeast
1 tablespoon	Salt
2 tablespoons	Butter (or canola oil)
3 tablespoons	Sugar
2-1/4 cups	Warm water
6-1/2 cups	All-purpose flour

1. In a large bowl, dissolve yeast in warm water. Add the sugar, salt, oil and 3 cups flour. Beat until smooth.
2. Stir in enough remaining flour to form a soft dough.
3. Turn onto a floured surface.
4. Knead until smooth and elastic.
5. Put in a greased bowl, turning once to grease the top.
6. Cover and let rise in a warm place until doubled, about 1-1/2 hours.
7. Punch dough down. Turn onto a lightly floured surface.
8. Divide dough in half. Shape each into a loaf. Place in two greased 9x5-inch loaf pans.
9. Cover and let rise until doubled, about 30–45 minutes.
10. Bake at 375° for 30 minutes or until golden brown.
11. Move breads from pans to wire racks to cool.
12. Slice and enjoy!

Figure 2.1 A sample recipe for bread

The recipe is a sequence of steps that you must follow to bake bread. The steps are sequential; for example, you can't take the loaf out of the oven without first putting the dough in the pan. At certain steps, you can choose to put in one item instead of another; for example, you can put in either butter or canola oil, but not both. And some steps may be repeated, such as occasionally checking for the crust color before declaring that the bread is done.

2.2.3 Visualize the recipe with flowcharts

When you read a recipe, the sequence of steps is likely outlined in words. To prepare you for understanding how to be a programmer, you should start to think about visualizing recipes with flowcharts, as discussed briefly in lesson 1. Figure 2.2 shows how to represent baking bread with a flowchart. In this scenario, you're using a bread machine, and the ingredients differ slightly than the ones shown in figure 2.1. In the flowchart, steps are entered in rectangular boxes. If a recipe allows for a possible substitution, represent that with a diamond box. If a recipe has you repeating a task, draw an arrow going back up to the first step in the repeated sequence.

2.2.4 Use an existing recipe or make one up?

There are many recipes for bread out there. How do you know which one to use? With a vague problem statement such as "Bake a loaf of bread," all are good to use because they all accomplish the task. In that sense, a more general problem statement is easier for you when you have a set of recipes to pick from, because any of the recipes will work.

But if you have a picky eater and are asked to bake something for which there's no recipe, you'll have a hard time accomplishing the task. You'll have to experiment with various ingredient combinations and quantities, and with various temperatures and baking times. Likely, you'll have to start over a few times.

The most common type of problem you'll be given is a specific task for which you have some information, such as "Bake me a two-pound rosemary loaf of bread using 4 cups of flour, 1 tablespoon of sugar, 1 tablespoon of butter, 1 teaspoon of salt, and 1 teaspoon of yeast." You may not find a recipe that accomplishes this task exactly, but you're given a lot of critical information already in the task; in this example, you have all the ingredient measurements except for the rosemary amount. The hardest part is experimenting with ways of putting the ingredients together and determining how much rosemary to add. If this isn't your first time baking, you come to the task with some intuition for how much rosemary is just right. The more practice you have, the easier it'll be.

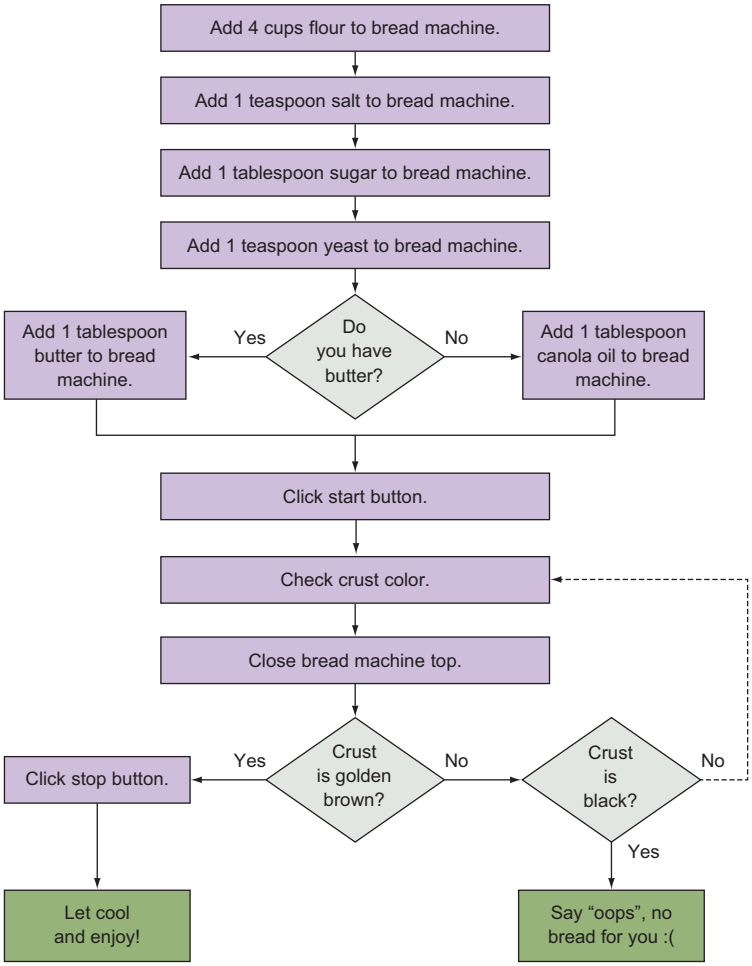


Figure 2.2 A flowchart of a simple recipe for baking bread. Rectangular boxes represent taking an action. Diamonds represent a decision point. The line going back up to a previous step represents a sequence repetition. Follow the arrows to trace paths through various implementations of the recipe.

The main idea to come away with from this baking example is that there's more to baking than following a recipe. First you have to understand what you're being asked to bake. Then you must determine whether you have any existing recipes that you can follow. If not, you have to come up with your own recipe and experiment until you have a

final product that matches what is being asked of you. In the next section, you'll see how the baking example translates into programming.

2.3 Think, code, test, debug, repeat

In this book, you'll write both simple and complicated programs. No matter what the complexity of the program, it's important to approach every problem in an organized and structured manner. I suggest using the think-code-test-debug-repeat paradigm shown in figure 2.3 until you're satisfied that your code works according to the problem specification.

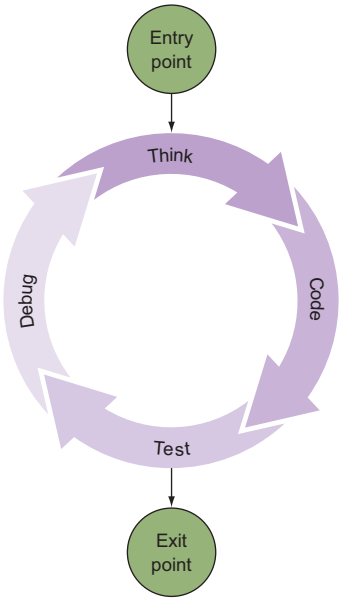


Figure 2.3 This is the ideal way to approach solving a problem with programming. Understand the problem before you write any code. Then test the code that you write and debug it as necessary. This process repeats until your code passes all tests.

The Think step is equivalent to making sure you understand what kind of baked good you're being asked to make. Think about the problem asked of you and decide whether you have any recipes that might work or if you need to come up with one on your own. In programming, recipes are *algorithms*.

The Code step is equivalent to getting your hands dirty and experimenting with possible combinations of ingredients, any substitutions, and any repetitive parts (for example, check the crust every five minutes). In programming, you're coding up an *implementation* of an algorithm.

The Test step is equivalent to determining whether the final product matches what the task was expecting you to produce. For example, is the baked good that came out of the oven a loaf of bread? In programming, you run a program with different inputs and check whether the *actual output* matches the *expected output*.

The Debug step is equivalent to tweaking your recipe. For example, if it's too salty, reduce the amount of salt you add. In programming, you *debug* a program to figure out which lines of code are causing incorrect behavior. This is a rough process if you don't follow best practices. Some are outlined later in this lesson, and unit 7 also contains some debugging techniques.

These four steps repeat as many times as necessary until your code passes all tests.

2.3.1 Understanding the task

When you're given a problem that you need to solve using programming, you should never begin to write code right away. If you start by writing code, you enter the cycle directly at the Code leg shown in figure 2.3. It's unlikely that you'll write your code correctly the first time. You'll have to cycle through until you think about the given problem, because you didn't correctly solve it the first time. By thinking about the problem at the start, you minimize the number of times you'll go through the programming cycle.

As you tackle harder and harder problems, it's also important that you try to break them into smaller problems with a simpler and smaller set of steps. You can focus on solving the smaller problems first. For example, instead of baking a loaf of bread with exotic ingredients, try a few small rolls to get the proportions just right without wasting too many resources or too much time.

When you're given a problem, you should ask yourself the following:

- What is this program supposed to accomplish? For example, "Find the area of a circle."
- Are there any interactions with the user? For example, "The user will enter a number" and "You show the user the area of a circle with that radius."
- What type of input is the user giving you? For example, "The user will give you a number representing the radius of a circle."
- What does the user want from the program and in what form? For example, you might show the user "12.57," you might be more verbose and show "The area of a circle with radius 2 is 12.57," or you might draw a picture for the user.

I suggest that you organize your thoughts on the problem by redescribing the problem in two ways:

- Visualize the problem.
- Write down a few sample inputs and then the outputs that you expect.

Quick check 2.1 Find any recipe (in a box or look one up on the internet). Write a problem statement for what the recipe is trying to achieve. Write a vague problem statement. Write a more specific problem statement.

2.3.2 Visualizing the task

When you're given a task to solve using programming, think of the task as a black box. At first, don't worry about the *implementation*.

DEFINITION An implementation is the way you write the code to solve a task.

Instead of worrying about the details of the implementation, think about what's being asked: Are there any interactions with the user? Is there any input your program might need? Is there any output your program might show? Is your program just supposed to be doing calculations behind the scenes?

It's helpful to draw a diagram showing possible interactions between your program and the user of the program. Go back to the bread example. A possible black box visualization is shown in figure 2.4. The inputs are represented to the left of the black box and the outputs to the right.

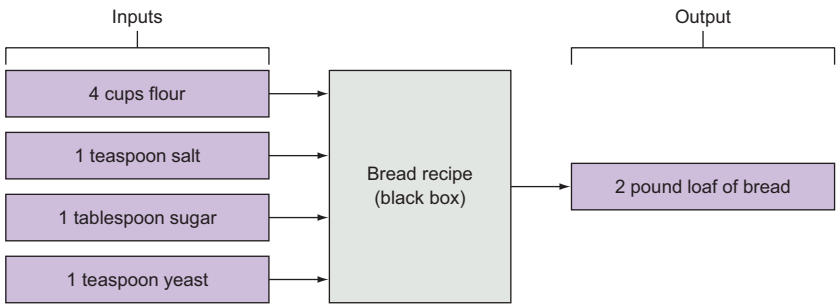


Figure 2.4 A black box visualization of baking a loaf of bread with a given set of ingredients.

When you have an idea of the inputs and outputs of your black box, think about any special behaviors that you may need to take into account. Will the program behave differently in different situations? In the bread example, can you substitute something for sugar if you don't have any? Will you get a different type of bread if you add sugar instead of salt? You should write out what the program will do in these situations.

All these specific interactions can be visualized in a flowchart. You can trace many routes through your flowchart, each route representing a different possible implementation and outcome, as in figure 2.2.

Quick check 2.2 You need to clean up after your baking adventure. You need to do two things: wash the dishes and take out the trash, in that order. Organize the following steps and decision points into a flowchart as in figure 2.2. Use as many steps/decisions as possible, but you don't have to use them all.

Step: Rinse dish
Step: Sing a song
Step: Tie up trash bag
Step: Take trash outside
Step: Pick up a dirty dish
Step: Scrub dirty dish with soap
Step: Put clean dish in drying rack
Step: Dump out trash bag on the floor
Step: Put a piece of trash in the trash bag

Decision: Anything else to put in the trash bag?
Decision: Am I happy with my baking skills?
Decision: Any more dirty dishes left?
Decision: Should I watch a movie tonight?

2.3.3 Writing pseudocode

At this point, you've come up with test cases, special behaviors you might have to be careful of, and a visual representation of a sequence of steps that you believe will accomplish the task given. If you drew out your sequence of steps, now is the time to convert your drawing into words, using programming concepts. To solve the problem, you must come up with a sequence of steps to follow so that they achieve the task outlined in the problem.

Pseudocode is a mix of English and programming on paper or in your code editor. It helps you get the structure of the program looking right at various points: when you get input from the user, when you show output, when you need to make a decision, and when you need to repeat a set of steps.

Putting the sequence of steps into words is like writing down and trying out your recipe. You must use what you know about how ingredients taste and what they're used

for to decide the best way to arrange them together. In programming, you must use everything you know about various techniques and constructs to put the code together, and this is the hardest part of programming.

In pseudocode, finding the area of a circle might look like this:√

1. Get a radius from the user.
2. Apply a formula.
3. Show the result.
4. Repeat steps 1–3 until the user says to stop.

Throughout this book, you'll see examples where certain programming concepts are useful. The only way to know which concept to use and when to use it is through intuition, which comes with a lot of practice.

Of course, there are many ways to achieve a task with programming. It's not a bad thing to go down one path and find yourself stuck; then you'll have a better understanding of why a particular method doesn't work in that case. With time and experience, you'll develop intuition for when one concept is better to use than another.

Quick check 2.3 Here's a problem statement. The Pythagorean theorem is $a^2 + b^2 = c^2$. Solve for c. Write a sequence of steps, in pseudocode, that you might take to solve for c.
Hint: $\sqrt{(x^2)} = x$

2.4 Writing readable code

As you learn more about programming, and specifically Python programming in this book, you'll see language specifics that Python offers to help you achieve this principle. I don't discuss those in this lesson. What's important to remember at this point, before you even start writing code, is that any code you write should be with the intent that someone else will read it, including yourself in a few weeks' time!

2.4.1 Using descriptive and meaningful names

Here's a short snippet of code in Python, which you don't need to understand right now. It consists of three lines of code, evaluated in order, top to bottom. Notice that it looks similar to something you may write in a math class:

```
a = 3.1
b = 2.2
c = a * b * b
```

Can you tell, at a high level, what the code is supposed to calculate? Not really. Suppose I rewrite the code:

```
pi = 3.1
radius = 2.2
# use the formula to calculate the area of a circle
circle_area = pi * radius * radius
```

Now can you tell, at a high level, what the code is supposed to do? Yes! It calculates—or rather, estimates—the area of a circle with radius 2.2. As in math, programming languages use *variables* to store data. A key idea behind writing readable code when programming is using descriptive and meaningful variable names. In the preceding code, `pi` is a variable name, and you can use it to refer to the value 3.1. Similarly, `radius` and `circle_area` are variable names.

2.4.2 Commenting your code

Also notice that the preceding code includes a line that starts with the `#` character. That line is called a *comment*. In Python, a comment starts with the `#` character, but in other languages, it can start with different special characters. A comment line isn't part of the code that runs when the program runs. Instead, comments are used in code to describe important parts of the code.

> **DEFINITION** A comment is a line in a Python program that starts with a `#`. These lines are ignored by Python when running a program.

Comments should help others, and yourself, understand why you wrote code in that way. They shouldn't just put into words what the code implements. A comment that says "Use the formula to calculate the area of a circle" is much better than one that says "Multiply pi times the radius times the radius." Notice that the former explains why the code is correct to use, but the latter simply puts into words what the code is implementing. In this example, someone else reading the code already knows that you're multiplying the three values (because they know how to read code!), but they might not know why you're doing the multiplication.

Comments are useful when they describe the rationale behind a larger chunk of code, particularly when you come up with a unique way to compute or implement something. A comment should describe the big idea behind the implementation of that particular chunk of code, because it might not be obvious to others. When someone reading the code understands the big picture, they can then go into the specifics of the code by reading each line to see exactly what calculations you're doing.

Quick check 2.4 Here's a short piece of Python code implementing a solution to the following problem. Fill in the comments. "You're filling a pool and have two hoses. The green hose fills it in 1.5 hours, and the blue hose fills it in 1.2 hours. You want to speed up the process by using both hoses. How long will it take using both hoses, in minutes?"

```
# Your comment here
time_green = 1.5
time_blue = 1.2

# Your comment here
minutes_green = 60 * time_green
minutes_blue = 60 * time_blue

# Your comment here
rate_hose_green = 1 / minutes_green
rate_hose_blue = 1 / minutes_blue

# Your comment here
rate_host_combined = rate_hose_green + rate_hose_blue

# Your comment here
time = 1 / rate_host_combined
```

Summary

In this lesson, my objective was to teach you

- The think-code-test-debug-repeat cycle of events that a good programmer should follow.
- To think about the problem you're given and understand what's being asked.
- To draw out what the inputs and outputs are based on the problem description before beginning to code.
- That a problem statement won't necessarily outline the series of steps you should take to solve the task. It may be up to you to come up with a recipe—a series of steps to achieve the task.
- To write code with the intent of it being read. You should use descriptive and meaningful names, and write comments that describe in words the problem and coded solution.

LESSON 3

Lesson 3 from *Get Programming* by Ana Bell

INTRODUCING PYTHON: A PROGRAMMING LANGUAGE

After reading lesson 3, you'll be able to

- Understand Python,the programming language you'll be using
- Use a program to write your programs
- Understand the components of a programming development environment

3.1 Installing Python

The Python programming language is, at the time of this writing, the most popular language for teaching introductory computer science. The language is used by top universities to expose students to programming, and many students are citing Python as a language they're familiar with upon entering college. Broadly, Python is used to build applications and websites, and is being used behind the scenes by companies such as NASA, Google, Facebook, and Pinterest to maintain features and analyze collected data.

Python is a great general-purpose language that can be used to write quick and simple programs. After you set up a working environment, writing a program in Python doesn't require much setup.

3.1.1 What is Python?

Python is a programming language created by Guido van Rossum at Centrum Wiskunde & Informatica in the Netherlands. But the name *Python* is also used to refer to the interpreter.

> **DEFINITION** A Python interpreter is a program used to run programs written in the Python programming language.

In the Python programming language, every *thing*, called an *object*, has characteristics (data) associated with it and ways to interact with it. For example, any word is a thing, or object, in Python. The data associated with the word *summer* is the letter characters in that sequence. One way that you can interact with the word is to change every letter to be uppercase. An example of a more complicated object is a bicycle. The data associated with a bicycle could be the number of wheels, its height, its length, and its color. The actions that a bike can do might be that it can fall over, a person can ride it, and you can repaint it.

In this book, you'll be writing programs in the latest Python version at the time of this writing, version 3.5.

3.1.2 Downloading Python version 3.5

You can download Python version 3.5 in various ways; you can get it from the official Python website, www.python.org, or through any third-party programs that offer the Python language as well as extra packages preinstalled. In this book, I recommend that you download a specific third-party program called the *Anaconda Python Distribution*.

3.1.3 Anaconda Python Distribution

You can download the Anaconda Python Distribution from www.anaconda.com. This free Python distribution offers various versions of Python and includes more than 400 of the most popular packages for science, math, engineering, and data analysis. There's also a lighter version, without any of the extra packages, called *Miniconda*.

Go to the downloads page, www.anaconda.com/downloads, and choose the Python 3.5 download link for your appropriate operating system. Follow the installation instructions with the default values to install the distribution on your computer. Note that the latest version might be different than Python 3.5, and that's fine. For our purposes, changes between subversions of Python 3 won't make a difference.

3.1.4 Integrated development environments

After the installation is complete, open Spyder, a program part of Anaconda. Spyder is an integrated development environment (IDE) that you'll use to write and run your programs in this book.

> **DEFINITION** An *integrated development environment* (IDE) is a complete programming environment that helps make your program writing experience a lot nicer.

Open Spyder

In Windows only, you can open Spyder from the Anaconda folder in the Start menu, shown in figure 3.1.

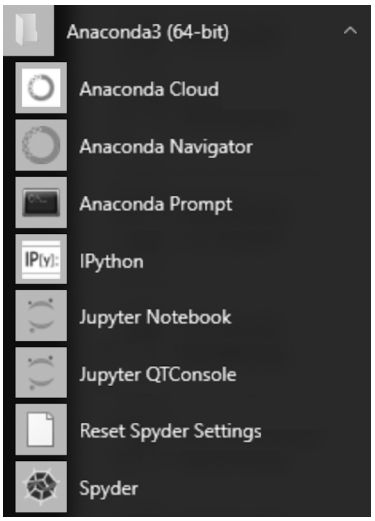


Figure 3.1 Anaconda folder in the Start menu

Some of the important features that the Spyder IDE offers, shown in figure 3.2, are as follows:

- An editor to write your Python programs
- A way to see lines of code, before running your program, that may contain potential errors or inefficiencies
- A console to interact with the user of your programs, through input and output
- A way to see values of variables in your program
- A way to step through your code line by line

Figure 3.2 shows the entire Spyder IDE and some code written in the code editor. You don't have to understand the code.

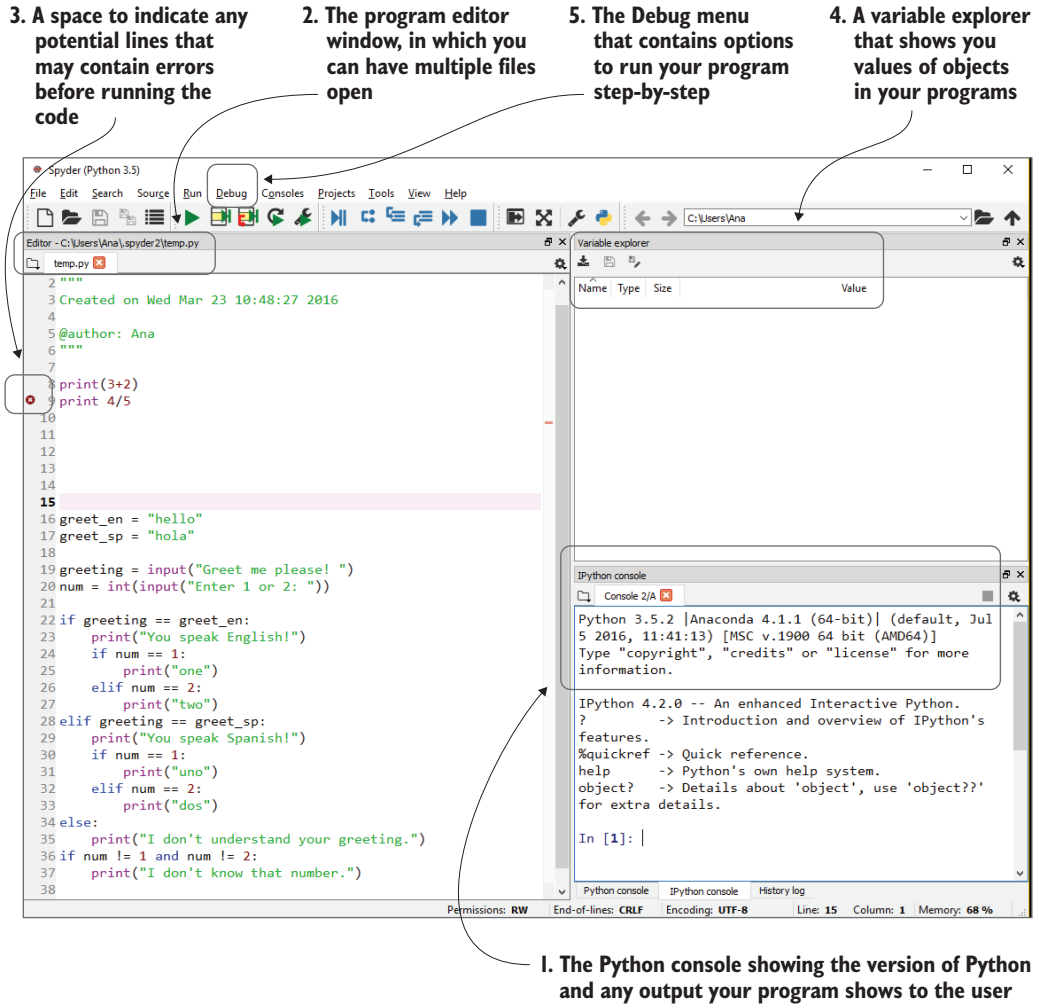


Figure 3.2 The Spyder IDE with the code editor, console, and variable explorer windows

3.2 Setting up your workspace

When you open Spyder, as shown in figure 3.2, you see that the program window is separated into three window panes:

- The left pane is the editor, originally containing no code, only a few lines of text. You'll notice that this text is green, meaning that this is a multi-line comment—not code that will be run.
- The top-right pane might contain the object inspector, variable explorer, or file explorer. You won't be using this window pane, but the variable explorer, for example, shows you the values for each variable in your program after the program is finished.
- The bottom-right pane is, by default, the IPython console. In this lesson, you'll see some basics regarding the IPython console and the file editor.

The next two sections will guide you through simple computations in Spyder. You'll see how to enter the computations directly in the console and how to write more-complicated programs in the code editor. At the end of the next two sections, your Spyder session should look like figure 3.3.

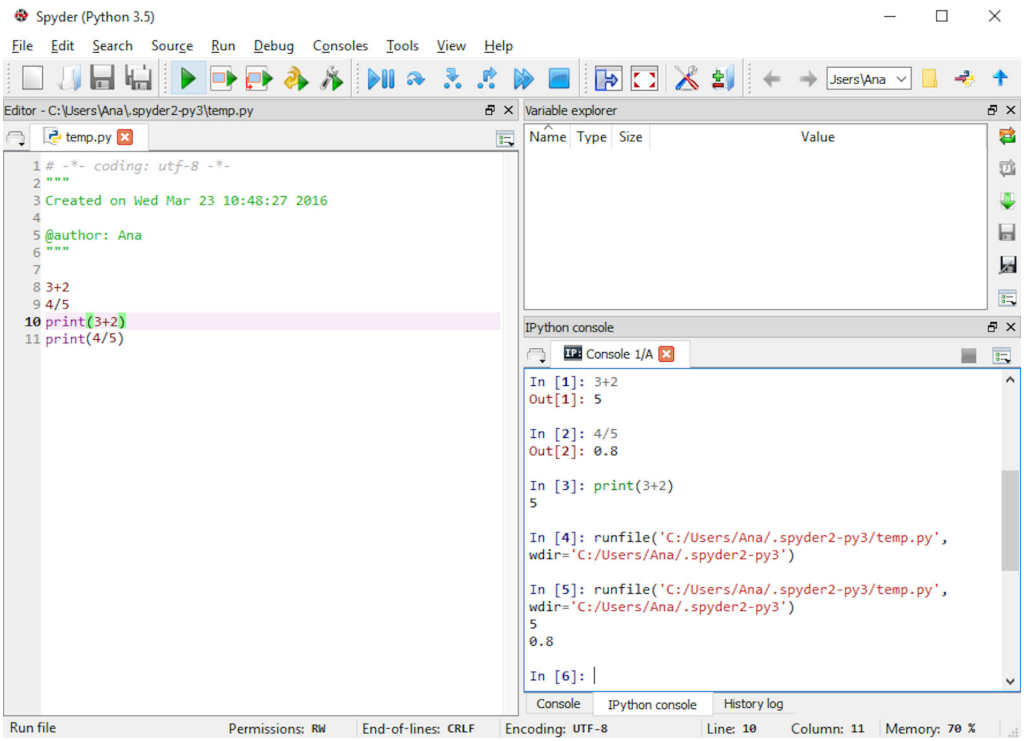


Figure 3.3 Spyder session after entering expressions in the IPython console and the code editor

3.2.1 The IPython console

The IPython console is the primary way that you can quickly test commands to see what they do. More important, users will be using the console to interact with your programs. The *I* in *IPython* stands for *interactive*. The IPython console is an advanced console that gives its users neat features including autocompletion, a history of previous commands typed in, and color highlighting of special words.

Writing commands directly into the console

You can write single commands directly in the IPython console to try things and see what they do. If you're just beginning to program, you should be trying things out a lot. That's the best way to start gaining intuition about what statements do and what expressions evaluate to.

Type `3 + 2` in the console and hit Enter to perform this addition. You'll see the result `5` preceded by the text `Out[]`. Now type `4 / 5` to perform this division and you'll see the result `0.8` preceded by the text `Out[]`.

You can think of this console as something that lets you peek into the values of the expressions that you type in. Why do I say *peek*? Because the results of these expressions aren't visible to a user. To make them visible to a user, you must explicitly print their values to the console. Type `print(3 + 2)` in the console. The number `5` is printed again, except that there's no `Out[]` right before it.

Both `3 + 2` and `4 / 5` are called Python *expressions*. In general, anything in Python that can be evaluated to a value is called an expression. You'll see more examples of expressions in lesson 4. In the next section, you'll see how to enter commands in the file editor to write more-complicated programs.

Quick check 3.1 Will the following expressions show output to the user, or are they just letting you peek into their value? Type the expressions in the console to check yourself!

```
1 6 < 7
2 print(0)
3 7 * 0 + 4
4 print("hello")
```

Primary uses of the console

Few programmers can write a perfect program on the first go. Even experienced programmers make mistakes. Your first try to write a program will be a little unsteady, and you'll have *bugs* (errors) that will show up when you try to run your program.

DEFINITION A *bug* is an error in a program.

If a program has bugs, big or small, you have to try to fix them. You can learn a lot from the debugging process. As you start to write more-complicated programs, you can think of using the console from the point of view of two roles: you as a programmer, and as a person interacting with your program (the user). Figure 3.4 shows the dual role the console lets you play. A programmer primarily uses the console to test commands and debug programs. A user uses the console to interact with a program that's running by typing in input and seeing what the program outputs.

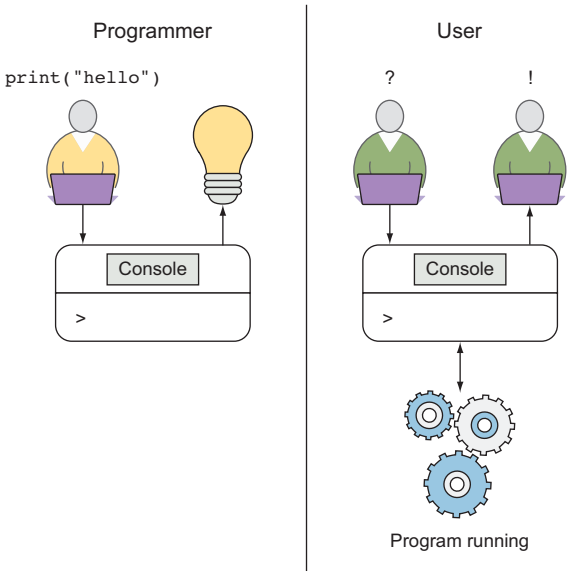


Figure 3.4 Programmers use the console for their own testing and debugging. They type commands directly in the console and look at the output. Users interact with a program via the console. They type input to a program and view the output from a program in the console.

The majority of the programs you'll see in this book don't have a visual interface. Instead, you'll write programs that interact with users via text in the console; users will be given the opportunity to enter text/numbers/symbols when prompted in the console, and your program will display results in the console. Figure 3.5 shows an example of how the user may interact with the programs you write.

As a programmer, you'll be using the console to take on the role of a user of your program. This is most useful when debugging programs (when you're trying to figure out why your program isn't working as expected). When you use the file editor to write more-complicated programs, it's often useful to have the console print values of any computations or objects in your programs, not just the final value. Doing so can help

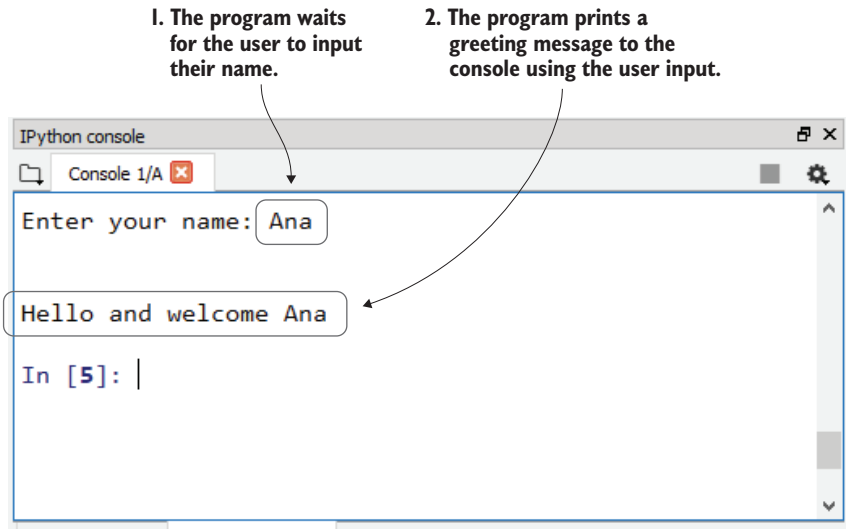


Figure 3.5 An example of a user interacting with a program

you determine intermediary values in your programs and help you debug. If running your program is like trying out a recipe, printing intermediary values is like tasting items in your recipe to make sure everything is going well. Debugging is covered in a lot more detail in unit 7.

The console is useful for trying out single expressions and seeing their values. You can retype expressions in the console if you want to run them again, or you can use the up arrow to see expressions you previously typed and hit Enter to run them again. A file editor saves your expressions to a file so you don't need to retype them. This saves a lot of time when you want to write programs that are longer than one line.

3.2.2 The file editor

When you write more-complicated Python programs containing more than just a couple of lines, you should use the file editor pane. Here, you can type the commands (in programming, called *statements*), one on each line, as in figure 3.3. After you finish writing a set of commands, you can run the program by clicking the green arrow in the toolbar at the top of Spyder, shown in figure 3.6. Editing and running files is the same for all operating systems that Anaconda supports: PC, Mac, and Linux. This book shows screenshots from a Windows operating system.

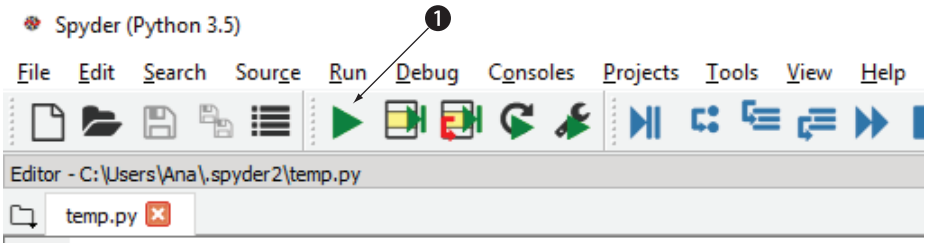


Figure 3.6 Click the green arrow button to run the program.

Not all lines of code produce output visible to the user

In the empty file, type `3 + 2` on line 8, as shown previously in figure 3.3. On the next line, type `4 / 5`. Don't type anything else yet. Now click the green arrow to run the program. The first time you click the arrow, you may get a pop-up that asks you for the working directory; it's OK to accept the default values. What happens? Your console at the bottom right shows some red text, similar to the following:

```
runfile('C:/Users/Ana/.spyder2-py3/temp.py',
➥wdir='C:/Users/Ana/.spyder2-py3')
```

That line indicates that your program ran, but nothing was shown to the user.

Now make the following additions. On line 10, type `print(3 + 2)`. And on the following line, type `print(4 / 5)`. Run the program again. Now what happens? You should see the same thing as in figure 3.3. The console shows the results of the calculations to the user, each on a different line.

How does that work? The Python interpreter executes each line in the file. It first runs the statement `3 + 2` and internally calculates the result of this expression. Then it internally calculates `4 / 5`. Because these two statements don't tell Python to show the output of the calculations, their values don't show up on the console.

A keyword in Python, `print`, is reserved for when you want to output the value of whatever is in the parentheses following `print` to the console. In this case, you show the result of evaluating the expressions `3 + 2` and `4 / 5`, in that order.

Quick check 3.2 Which of these expressions will the user see on the console? Type the expressions in the file editor and click Run to check!

```
1 print(4 - 4 * 4)
2 print(19)
3 19 - 10
```

Saving files

You should save every program you write in a separate file to keep you organized. The file in which you wrote the previous code is a temporary file, saved in some location in the Anaconda installation folder. Open a new file from the Spyder menu bar, as shown in figure 3.7. Type the previous two `print` statements again in the new file.

> **TIP** I strongly encourage you to type the commands again instead of copying and pasting. Repetition is a great way to help you get the hang of programming. Forcing yourself, especially at the beginning of your programming career, to type commands will help speed up your learning process and make writing code second nature.

Now save the file in a directory of your choosing. You must save it with a .py extension. If you don't save it with this extension, you won't be able to run the program (the green Run button will be gray). After you save the file, click the green Run button. The same output as before should show up in the console.

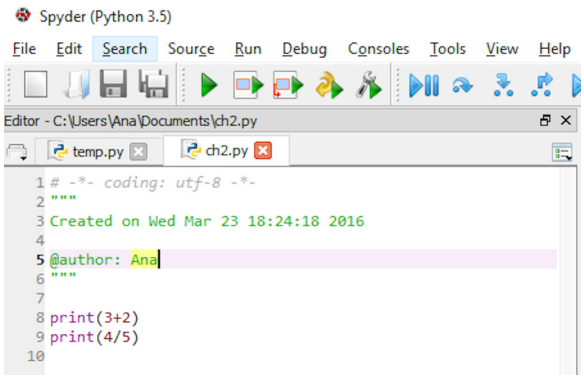


Figure 3.7 Multiple files open in Spyder; each one has its own tab in the file editor pane.

If you close the file you just saved, your program isn't lost. You can reopen the file from the File menu. All the code is still there, and you can run the program as if you just wrote it.

Summary

In this lesson, my objective was to teach you

- How to install a Python distribution called Anaconda, using Python version 3.5 and an IDE called Spyder

- How to open a new file, write a simple program in the file, save the file, and run a program
- How to write code in the file editor and open many files in the editor pane
- That the console allows you to peek into values of variables or to show output to the user
- How to use `print` statements to print expression values to the console

LESSON 4

Lesson 4 from *Get Programming* by Ana Bell

VARIABLES AND EXPRESSIONS: GIVING NAMES AND VALUES TO THINGS

After reading lesson 4, you'll be able to

- Write code that creates Python objects
- Write code that assigns objects to variables

In your everyday life, you encounter many physical objects, or *things*. Each of these things has a name. They have names because it's much easier to refer to them using a name rather than a description.

Using names is a great help when you're always manipulating things, or objects. Some things are simple, such as the number 9. Some are more complicated, such as a dictionary. I can give the name *Nana* to the number 9, and the name *Bill* to my dictionary. You can give things (almost) any name you want. You can even give names to combinations of things. For example, if I glue a banana to my laptop cover to create a new thing, I can name that new trendy creation Banalaptop. Individual things can be named as well; if I have two apples, I can name one Allie and the other one Ollie.

After you name things, you can refer to them later without any confusion. The benefit of using names is that you don't have to re-create (in programming, *recalculate*) values. When you name a thing, you inherently remember every detail about it.

Consider this

Scan the room you're in right now for a few things. Then take these steps:

1. Write the items down. (I saw my phone, a chair, a carpet, papers, and a water bottle.)
2. Write a sentence using some or all of these objects. You can use an object more than once. (My water bottle spilled on my phone and papers, and now my phone is broken and my papers are ruined.)
3. Write a description of each object without using its name.
4. Now rewrite the sentence you came up with using only the descriptions.
5. Is the sentence you wrote easier to read using the item names or descriptions?

Answers:

1. A phone, papers, and a water bottle.
2. My water bottle spilled on my phone and papers, and now my phone is broken and my papers are ruined.
3. Descriptions:
   - *Water bottle*—Thing that holds clear liquid
   - *Phone*—Rectangular device I use to call/text/watch cat videos
   - *Papers*—Stack of thin, white, flimsy things with black-and-white text on them
4. A thing that holds clear liquid spilled on a rectangular device I use to call/text/watch cat videos and on a stack of thin, white, flimsy things with black-and-white text on them, and now my rectangular device is broken, and my things with black-and-white text are ruined.
5. The sentence is easier to read with the item names, not the descriptions.

4.1 Giving names to things

Everything you use has a name, which makes it easier to reference in conversation. Writing a computer program is like writing a detailed description of the events you want to happen and the things involved. In programming, you reference things by using variables, which are discussed in the context of programming in section 4.2.

4.1.1 Math vs. programming

When you hear the word *variable*, it may remind you of math class, where you did calculations with equations and were asked to "solve for x." Programming also uses variables, but in a different way.

In math, lines of equations state an equivalence. For example, "$x = 1$" stands for "x is equivalent to 1," and "$2 * x = 3 * y$" stands for "2 times x is equivalent to 3 times y."

In programming, lines of code with an equal sign stand for an *assignment*. Figure 4.1 shows an assignment in Python.

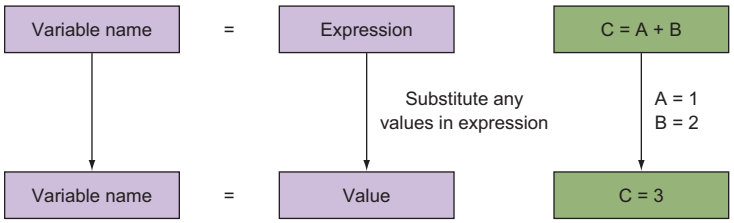


Figure 4.1 Assignment to a name in Python. Any expression on the right side gets converted to a single value and given a name.

You use the equal sign to assign variables to values. For example, `a = 1` or `c = a + b`. The thing to the right of the equal sign is an *expression* with a *value*.

DEFINITION An expression is a line of code that can be reduced to a value.

To get the value, you substitute the values for all other known variables in the expression and do the calculation. For example, if `a = 1` and `b = 2`, then `c = a + b = 1 + 2 = 3`. In programming, the only thing you're allowed to have to the left of the equal sign is the name of a variable.

4.1.2 What the computer can and can't do

An important point is worth coming back to: a computer needs to be told what to do. A computer can't spontaneously solve an equation on its own. If you tell the computer that `a = 2`, `b = 2`, and that `a + x = b`, it doesn't know what to do with this information or how to solve for `x` on its own. The line `a + x = b` doesn't tell the computer how to calculate anything; it just states an equivalence.

The computer needs to be told a recipe for solving something. Recall that when you're following a recipe for baking bread, you need to know the steps. You, as the programmer, have to come up with the recipe and tell the computer what to do. To come up with the recipe, you need to go off-computer and on-paper and do the calculation on your own. Then you can tell the computer what steps it needs to calculate a value.

Quick check 4.1 Decide whether the computer is allowed to do the following assignments. Assume that every *thing* on the right side of the equal sign has a value:

```
1 3 + 3 = 4
2 stuff + things = junk
3 stack = 1000 * papers + 40 * envelopes
4 seasons = spring + summer + fall + winter
```

4.2 Introducing variables

With that bit of intuition for how variables work in programming, you can now dive in and start learning about how variables work.

4.2.1 Objects are things that can be manipulated

In the previous section, we talked about things. In Python, everything is an object. This means that every *thing* that you can create in Python has the following:

- A type
- A set of operations

The *type* of an object tells you the data/values/attributes/properties associated with it. The *operations* are commands that you can tell the object to do; these commands might work only on the object itself, or they might be ways that the object can interact with other objects.

Quick check 4.2 For the following items, write some attributes (describe their color, size, and so forth) and some operations (what it can do, how it can interact with something else, and so forth):

1. Phone
2. Dog
3. Mirror
4. Credit card

4.2.2 Objects have names

Every *thing* that you create in a program can be given a name so you can refer to it later. The names are *variables* and are used to refer to objects.

DEFINITION A variable is used to bind a name to an object. A variable name refers to a particular object.

For example:

- If `a = 1`, then the object named `a` has the value `1`, and you can do mathematical operations with it.
- If `greeting = "hello"`, then the object named `greeting` has a value of `"hello"` (a sequence of characters). Operations you can do on this object include "tell me how many characters it has" or "tell me if it has the letter *a* in it" or "tell me at which position the first *e* occurs."

In both of these examples, the item to the left of the equal sign is a variable name that you can use to refer to an object, and the thing to the right of the equal sign is the object itself, which has a value and some operations you can do on it. In Python, you bind a variable name to an object.

The object to the right of the equal sign doesn't have to be only one object. It can be a calculation that can be simplified to give a value. With that final value, you get an object. For example, `a = 1 + 2` is a calculation on two objects (`1` and `2`) that can be simplified to one object with a value of `3`.

4.2.3 What object names are allowed?

You write code with variable names that make the code readable by other people. Many programming languages, Python included, have restrictions on the names you can use for variables:

- Must begin with a letter (`a` to `z` or `A` to `Z`) or an underscore (_).
- Other characters in the variable name can be letters, numbers, or an underscore.
- Names are case-sensitive.
- Names can be any length.

Thinking like a programmer

If you want, you can have a variable name that is 1,000,000,000,000,000 characters long. But don't! It makes your code unreadable. Limit lines of code to at most 80 characters and try to make your names as concise as possible while maintaining readability.

Quick check 4.3 Are the following variable names allowed?

```
1 A
2 a-number
3 1
4 %score
5 num_people
6 num_people_who_have_visited_boston_in_2016
```

Programming languages have a few reserved words that you can't use as variable names. For Python, Spyder has *syntax highlighting*, which changes the color of words that are special reserved Python *keywords*.

DEFINITION A keyword is a special word. It's reserved because it has special meaning in a programming language.

Figure 4.2 shows an example of syntax highlighting. A good general rule is that if the variable you want to use turns a different color, you shouldn't use it as a variable name.

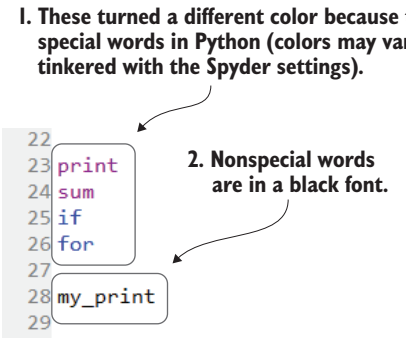


Figure 4.2 Special words that have a meaning in Python change color in the code editor. As a general rule, you shouldn't name your variables using any words that turn a color other than black.

In addition to the preceding rules for naming variables, here are some guidelines to help you write programs that are more readable:

- Choose descriptive and meaningful names instead of short, single-character names.
- Use underscores to add a pretend space between variable words.
- Don't use variable names that are too long.
- Be consistent throughout your code.

Quick check 4.4 Are the following allowed and good variable names?

1. `customer_list`
2. `print` (where this word is a color other than black in Spyder)
3. `rainbow_sparkly_unicorn`
4. `list_of_things_I_need_to_pick_up_from_the_store`

4.2.4 Creating a variable

Before you can work with a variable, you have to set it to a value. You *initialize* the variable by assigning it to an object, using the equal sign. The initialization binds the object to the variable name.

DEFINITION A variable initialization binds a variable name to an object.

After you initialize a variable, you can refer to a particular object by using its variable name. In Spyder, type the following lines in the console to initialize three variables:

```
a = 1
b = 2
c = a + b
```

You can use the variable explorer to see the names of variables, their type, and size (you'll see what this means in following lessons), and their value. Figure 4.3 shows how your screen should look.

You should see that the variable explorer is populated with the variables you create and their values. If you type in the name of a variable in the console and hit Enter, this allows you to *peek* into its value. The variable explorer also tells you an additional bit of information in the second column: the type of the variable. The next section goes into more detail on what this means.

4.2.5 Updating a variable

After you create a variable name, you can update the name to be any object. You saw that these lines initialize three variables:

```
a = 1
b = 2
c = a + b
```

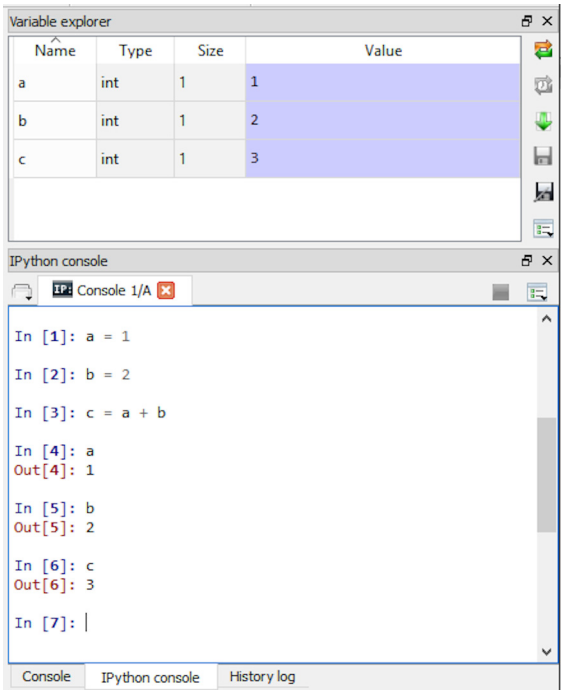


Figure 4.3 How to create variables in the console. The variable explorer shows you what variables you have set up and initialized in this session.

You can update the value of `c` to be something else. Now you can type `c = a - b` to reassign the variable `c` to have a new value. In the variable explorer, you should see that the variable `c` now has a different value. Figure 4.4 shows how Spyder looks now.

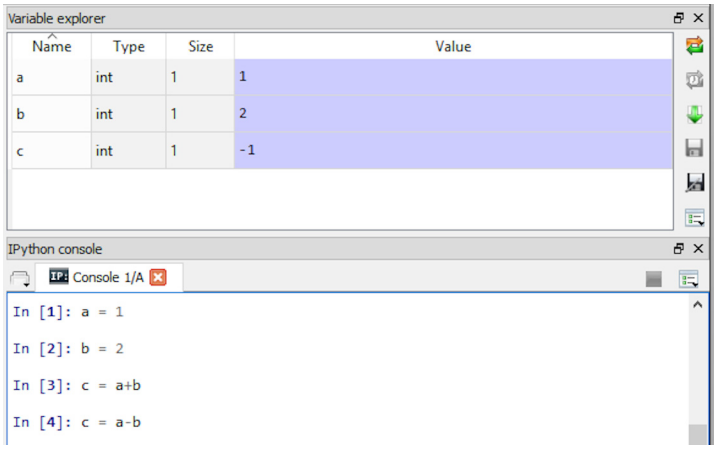


Figure 4.4 The variable explorer has the same variable `c`, except with a new value.

Variable names merely bind names to objects. The same name can be reassigned to a different object. A Python operation, named `id`, shows the identity of an object in the form of a sequence of digits. The identity is unique for every object and won't change while the object exists. Type the following lines in the console:

```
c = 1
id(c)
c = 2
id(c)
```

After the first `id(c)` command, my console printed out `1426714384`. After the second `id(c)` command, I got `1426714416`. These are two numbers for the same variable name because the numbers 1 and 2 are different objects.

Quick check 4.5 Assume you're doing the following actions in order. Write a line of code for each:

1. Initialize a variable named `apples` to the value 5.
2. Initialize a variable named `oranges` to the value 10.
3. Initialize a variable named `fruits` to the sum of `apples` and `oranges`.
4. Reassign the variable `apples` to be 20.
5. Recalculate the variable `fruits` just as before.

Summary

In this lesson, my objective was to teach you

- To create and initialize variables
- That not all names are allowed for variable names and that there are general rules for naming your variables
- That an object has a value
- That expressions are lines of code that can be reduced to a value
- That an object has operations you can do on it
- That a variable is a name that is bound to an object

Let's see if you got this …

Q4.1 You're given the following problem. Solve the equation for x. Write x in terms of an expression and then find its value.

```
a = 2
b = 2
a + x = b
```

Q4.2 Type `a + x = b` in the Spyder console and hit Enter. You get an error. Maybe the error happened because you didn't tell the computer what `a` and `b` were. Type the following lines in the console, each followed by pressing Enter. Do you still get an error?

```
a = 2
b = 2
a + x = b
```

LESSON 5

Lesson 5 from *Get Programming* by Ana Bell

OBJECT TYPES AND STATEMENTS OF CODE

After reading lesson 5, you'll be able to

- Write code that creates various types of objects
- Write simple lines of code to manipulate Python variables

Suppose you have a family as follows:

- *Four people*—Alice, Bob, Charlotte, and David
- *Three cats*—Priss, Mint, and Jinx
- *Two dogs*—Rover and Zap

Every person, cat, and dog is a separate object. You named each object something different so you can easily refer to them and so that everyone else knows which object you're talking about. In this family, you have three types of objects: people, cats, and dogs.

Each type of object has characteristics that are different from another type. People have hands and feet, whereas cats and dogs have only feet. Cats and dogs have whiskers, but people don't. The characteristics of a type of object uniquely identify all individual objects in that type. In programming, characteristics are called data *attributes*, or *values*, for the type.

Each type of object also has actions or behavior. People can drive a car, but dogs and cats can't. Cats can climb trees, but dogs can't. The actions that a type of object can do are specific to that object only. In programming, actions are called *operations* on the type.

Consider this You have a sphere and a cube. Write some characteristics of each (choose characteristics that uniquely identify them) and some actions you can do with each.

Answer:

Sphere—Round, has a radius/diameter, rolls, bounces

Cube—All sides equal length, stays flat, has points, can stand on it

5.1 Types of things

So far, you've created variables to store objects. Variables are names given to individual objects. In reality, you can classify objects into groups. All objects in the same group are going to be of the same type; they'll all have the same basic properties, and they'll all have the same basic operations for interacting with them.

5.2 Basic type of objects in programming

Objects have

- A type, which dictates the values they can have
- Operations you can do with them

DEFINITION An object type tells you what kinds of values the object can have.

In most programming languages, a few types of objects are the basic building blocks for each language. These types of objects might be called *primitives*, or *scalars*. These basic types are built in to the language, and every other type of object can be made up of combinations of these primitives. This is similar to how the 26 letters in the alphabet are the building blocks of the English language; from the 26 letters, you can make words, sentences, paragraphs, and novels.

Python has five basic *types* of objects: integers, floating point, Booleans, strings, and a special type that represents the absence of a value. In Python, these five types are called *primitives*, and every other type of object in the language can be constructed with these five types.

5.2.1 Integers as whole numbers

An object of type *integer* (in Python, the `int` type) is an object whose values are real whole numbers. For example, 0, 1, 2, 5, 1234, -4, -1000 are all integers.

The kinds of operations you can do on these numbers are, as you might expect, operations that you would do on numbers in math class.

You can add, subtract, multiply, and divide two or more numbers. You may have to surround a negative number with parentheses to avoid confusing the negative number with the subtraction operation. For example,

- `a = 1 + 2` adds an integer object with value `1` and an integer object with value `2` together and binds the resulting object with a value of `3` to the variable named `a`.
- `b = a + 2` adds the value of the integer object named `a` and the integer object with value `2` together and binds the resulting object's value to the variable named `b`.

You can increment a number by a certain value. For example,

- `x = x + 1` means that you add `1` to the value of `x` and rebind that value to the variable named `x`. Notice that this is different than in math, where you would solve this equation by moving `x` from the right to the left of the equal sign (or subtracting `x` from both sides of the equation) and simplify the expression to `0 = 1`.
- `x += 1` is programming shorthand notation for `x = x + 1`. It's alternate, valid syntax. You can replace the `+=` with `*=`, `-=`, or `/=` to stand for `x = x * 1`, `x = x - 1`, or `x = x / 1`, respectively. The `1` on the right-hand side of the equal sign can also be replaced with any other value.

Quick check 5.1 Write a line of code that achieves each of the following:

1. Add 2 and 2 and 2 and store the result in a variable named `six`.
2. Multiply the variable `six` with -6 and store the result in a variable named `neg`.
3. Use shorthand notation to divide the variable `neg` by 10 and store the result in the same variable, `neg`.

5.2.2 Floating point as decimal numbers

An object of type *floating point* (in Python, the `float` type) is an object whose values are decimal numbers. For example, 0.0, 2.0, 3.1415927, and -22.7 are all floats. If you've played around with integers, you may have noticed that when you divided two numbers, the result was a float type. The kinds of operations you can do on these numbers are the same as those for integers.

It's important to understand that the following two lines lead to two variables representing objects of two types. The variable `a` is an integer, but the variable `b` is a float:

```
a = 1
b = 1.0
```

Quick check 5.2 Write a line of code that achieves each of the following:

1. Multiply 0.25 by 2 and store the result in a variable named `half`.
2. Subtract the variable `half` from 1.0 and store the result in a variable named `other_half`.

5.2.3 Booleans as true/false data

In programming, you often work with more than just numbers. A type of object that's even simpler than a number is the *Boolean* (in Python, the `bool` type); it has only two possible values, `True` or `False`. They replace expressions with one of these two values; for example, the expression `4 < 5` is replaced by `False`. The kinds of operations you can do on Booleans involve the logic operations `and` and `or` and were briefly introduced in lesson 1.

Quick check 5.3 Write a line of code that achieves each of the following:

1. Store the value `True` in a variable named `cold`.
2. Store the value `False` in a variable named `rain`.
3. Store the result of the expression `cold and rain` in a variable named `day`.

5.2.4 Strings as sequences of characters

A useful data type is the string (in Python, the `str` type), which is covered in a lot more detail in lessons 7 and 8. Briefly, a *string* is a sequence of characters surrounded by quotation marks.

One character is anything you can enter by hitting one key on the keyboard. The quotations surrounding characters can either be single or double quotation marks (`'` or `"`) as long as you're consistent. For example, `'hello'`, `"we're # 1!"`, `"m.ss.ng c.ns.n.nts??"`, and `"'"` (the single quote inside two double quotes) are possible values for a string.

You can do many operations on strings, and they are detailed in lesson 7.

Quick check 5.4 Write a line of code that achieves each of the following:

1. Create a variable named `one` with the value `"one"`.
2. Create a variable named `another_one` with the value `"1.0"`.
3. Create a variable named `last_one` with the value `"one 1"`.

5.2.5 The absence of a value

You might want to designate the absence of a value in a program. For example, if you get a new pet and haven't named it yet, the pet doesn't have a value for its name. Programming languages allow you to designate a special value in this situation. In many programming languages, this is referred to as `null`. In Python, the value is `None`. Because in Python everything is an object, this `None` also has a type, `NoneType`.

Quick check 5.5 What is the type of each object?

```
1 2.7
2 27
3 False
4 "False"
5 "0.0"
6 -21
7 99999999
8 "None"
9 None
```

5.3 Working with basic types of data values

Now that you understand a little bit about the types of objects you'll be working with, you can start to write code consisting of more than one line of code. When you're writing a program, each line of code is called a *statement*. A statement may or may not contain an *expression*.

DEFINITION A statement is any line of code.

5.3.1 Building blocks of expressions

An *expression* is an operation between objects that can be reduced to a single value. The following are examples of expressions (and statements):

- `3 + 2`
- `b - c` (if you know the values of `b` and `c`)
- `1 / x`

A line of code that prints something is a statement but not an expression, because the act of printing can't be reduced to a value. Similarly, a variable assignment is an example of a Python statement but not an expression, because the act of doing the assignment doesn't have a value.

Quick check 5.6 Write down whether each of the following is a statement, expression, or both:

```
1 2.7 - 1
2 0 * 5
3 a = 5 + 6
4 print(21)
```

5.3.2 Converting between different types

If you're not sure of the type of an object, you can use Spyder to check for yourself. In the console, you can use a special command, `type()`, to get the data type of an object. For example,

- Type in the console `type(3)` and hit Enter to see that the type of `3` is an integer.
- Type in the console `type("wicked")` and hit Enter to see that the type of `"wicked"` is a string.

You can also convert objects from one type to another. To do this in Python, you surround the object you want to convert with parentheses and the name of the type that you want to convert to. For example,

- `float(4)` gives `4.0` by converting the int `4` to the float `4.0`.
- `int("4")` gives `4` by converting the string `"4"` to the int `4`. Notice that you can't convert a string that isn't a number to an int or a float. If you try to convert `int("a")`, you'll get an error.
- `str(3.5)` gives `"3.5"` by converting the float `3.5` to the string `"3.5"`.
- `int(3.94)` gives `3` by converting the float `3.94` to the int `3`. Notice this truncates the number to keep only the whole number before the decimal point.
- `int(False)` gives `0` by converting the bool `False` to the int `0`. Notice that `int(True)` gives `1`.

Quick check 5.7 Write an expression that converts the following objects to the desired types and then predict the converted value. Remember that you can check by typing the expressions in the Python console:

1. `True` to a `str`
2. `3` to a `float`
3. `3.8` to a `str`
4. `0.5` to an `int`
5. `"4"` to an `int`

5.3.3 How arithmetic impacts object types

Mathematical operations are one example of Python expressions. When you do an operation between numbers in Python, you get a value, so all mathematical operations are expressions in Python.

Many of the operators between numbers in math work between numbers (ints or floats) in Python. You're allowed to mix and match integers and floats when you do the mathematical operations. Table 5.1 shows what happens when you do mathematical operations on all possible combinations of ints and floats. The first row of the table says that when you add an int to another int, you get an int result. One example of this is adding 3 and 2 to get 5. When at least one of the operands is a float, the result will be a float. Adding 3.0 + 2, or 3 + 2.0, or 3.0 + 2.0 all result in the float 5.0. The exception to this is division. When you divide two numbers, you always get a float, no matter what the operand types are.

Table 5.1 shows two operations you haven't seen before—the power and the remainder:

- The *power* (**) is a base raised to the power of an exponent. For example, 3^2 is written as `3 ** 2` in Python.
- The *remainder* (%) gives you the remainder when the first object is divided by the second object. For example, `3 % 2` finds out how many times 2 goes into 3 (only one time) and then tells you how much is left (1, because you have to add 1 more to `1 * 2` to get 3).

Table 5.1 Mathematical operations on ints and floats and the resulting types

Type of first object	Operation(s)	Type of second object	Type of result	Example	
int	+ - * ** %	int	int	3 + 2 3 - 2 3 * 2 3 ** 2 3 % 2	gives 5 gives 1 gives 6 gives 9 gives 1
int	/	int	float	3 / 2	gives 1.5
int	+ - * / ** %	float	float	3 + 2.0 3 - 2.0 3 * 2.0 3 / 2.0 3 ** 2.0 3 % 2.0	gives 5.0 gives 1.0 gives 6.0 gives 1.5 gives 9.0 gives 1.0
float	+ - * / ** %	int	float	3.0 + 2 3.0 - 2 3.0 * 2 3.0 / 2 3.0 ** 2 3.0 % 2	gives 5.0 gives 1.0 gives 6.0 gives 1.5 gives 9.0 gives 1.0
float	+ - * / ** %	float	float	3.0 + 2.0 3.0 - 2.0 3.0 * 2.0 3.0 / 2.0 3.0 ** 2.0 3.0 % 2.0	gives 5.0 gives 1.0 gives 6.0 gives 1.5 gives 9.0 gives 1.0

Another operation you can do on numbers is to round them by using the `round()` command; for example, `round(3.1)` gives you the int `3`, and `round(3.6)` gives you the int `4`.

Quick check 5.8 What is the value and type of the resulting value of each expression? Recall that you can use the `type()` command to check yourself. You can even put an expression inside the parentheses of `type`; for example, `type(3 + 2)`.

```
1 2.25 - 1
2 3.0 * 3
3 2 * 4
4 round(2.01 * 100)
5 2.0 ** 4
6 2 / 2.0
7 6 / 4
8 6 % 4
9 4 % 2
```

Summary

In this lesson, my objective was to teach you about variables with a few basic types in Python: integers, floats, Booleans, strings, and a special `NoneType`. You wrote code to work with object types, and to do specific operations on ints and floats. You also wrote statements and expressions. Here are the major takeaways:

- An object has a value and operations you can do on it.
- All expressions are statements, but not all statements are expressions.
- Basic data types are ints, floats, bools, strings, and a special type to represent the absence of a value.

LESSON 6

Lesson 6 from *Get Programming* by Ana Bell

CAPSTONE PROJECT: YOUR FIRST PYTHON PROGRAM—CONVERT HOURS TO MINUTES

After reading lesson 6, you'll be able to

- Read your first programming problem
- Walk through two possible solutions
- Write your first Python program

Here are some of the main ideas you should be familiar with so far:

- Programs are made up of a sequence of statements.
- Some statements initialize variables.
- Some statements can be expressions to do calculations.
- Variables should be given descriptive and meaningful names, especially to help future programmers who might be looking at the code.
- Some calculations you've seen so far are addition, subtraction, multiplication, division, remainder, and power.
- You can convert an object to a different type.
- The `print` command can be used to show output to the console.
- You should write comments in the code to document what the code is doing.

THE PROBLEM The first programming task you'll see is to write a program in Python that converts minutes to hours. You'll start with a variable that contains the number of minutes. Your program will take that number, do some calculations, and print out the conversion to hours and minutes.

Your program should print the result in the following way. If the number of minutes is 121, the program should print this:

```
Hours
2
Minutes
1
```

6.1 Think-code-test-debug-repeat

Recall that before you begin to code, you should make sure to understand the problem. You can get the big picture by drawing your program as a black box. Figure 6.1 shows your program as a black box, any inputs, and any outputs you must generate.

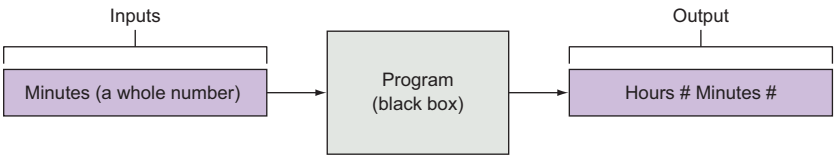


Figure 6.1 The input to the program is any whole number representing minutes. The program does some calculations and prints out how many hours that is and any leftover minutes.

After you understand the inputs and outputs, come up with a few inputs and write down what you expect the outputs to be for each. Here are some other possible inputs for the number of minutes and their conversions to hours:

- "60 minutes" is converted to "1 hour and 0 minutes".
- "30 minutes" is converted to "0 hours and 30 minutes".
- "123 minutes" is converted to "2 hours and 3 minutes".

These input-output pairs are called *sample test cases*. You'll be able to use these inputs and expected outputs to test your program after you write it.

Quick check 6.1 What's the expected output given the following input for the number of minutes?

```
1 456
2 0
3 9999
```

6.2 Divide your task

Now that you understand what the problem is asking, you have to figure out whether you can break it into smaller tasks.

You need to have input to convert, so this can be one task. You're showing the user a result, and this can be another task. These two tasks are going to be easy to implement with at most a couple of lines of code.

Code to set up the input

To set up the input, you need to initialize a variable with a value. Your variable name should be descriptive, and the number of minutes should be an integer. For example,

```
minutes_to_convert = 123
```

Code to set up the output

To show the output to the user, the format required is as follows, where `<some number>` is calculated by your program:

```
Hours
<some number>
Minutes
<some number>
```

You show the user output by using the `print` command. Here's the code:

```
print("Hours")
print(hours_part)
print("Minutes")
print(minutes_part)
```

Here, `hours_part` and `minutes_part` are variables you'll calculate in your program.

Now the only thing left to do is to come up with a way to do the conversion from minutes to hours and minutes. This is going to be the most involved part of the overall task.

6.3 Implement the conversion formula

When you're dealing with time units, you know that 1 hour has 60 minutes. Your first instinct may be to divide the number of minutes you're given by 60. But the division gives you a decimal number: at a first pass, given 123 minutes, your result will be 2.05, not 2 hours and 3 minutes.

To do the conversion properly, you must divide the problem into two parts: find out the number of hours and then find out the number of minutes.

6.3.1 How many hours?

Recall that given 123 minutes, dividing 123/60 gives 2.05. Notice that the whole number part, 2, represents the number of hours.

Quick check 6.2 Divide the following numbers by 60 and determine the whole number part of the result. You can do the division in Spyder to check yourself:

1. 800
2. 0
3. 777

Recall that in Python, you can convert one type to another type. For example, you can covert the integer 3 to a float by using `float(3)` to give you `3.0`. When you convert a float to an int, you remove the decimal point and everything after it. To get the whole number part of a division, you can convert the float result to an int.

Quick check 6.3 Write a line of code for each of the following points and then answer the questions at the end:

1. Initialize a variable named `stars` with the value 50.
2. Initialize another variable named `stripes` with the value 13.
3. Initialize another variable named `ratio` with the value `stars` divided by `stripes`. Question: what is the type of `ratio`?
4. Convert `ratio` to an int and save the result in a variable named `ratio_truncated`. Question: what is the type of `ratio_truncated`?

In the given task, you'll divide the minutes by 60 and convert the result to an integer to give you the number of whole hours, like so:

```
minutes_to_convert = 123
hours_decimal = minutes_to_convert/60
hours_part = int(hours_decimal)
```

At this point, your `hours_part` variable holds the number of hours converted from the input.

6.3.2 How many minutes?

Finding out the number of minutes is a little bit trickier. In this section, you'll see two ways of doing this:

- *Method 1*—Use the decimal portion of the result from the division. If you use the 123 minutes example, how can you convert the decimal part 0.05 into minutes? You should multiply 0.05 by 60 to give you 3.
- *Method 2*—Use the remainder operator, `%`. Again, use the 123 minutes example. The remainder when 123 is divided by 60 is 3.

6.4 Your first Python program: one solution

The code for the final program using method 1 is shown in listing 6.1. The code is separated into four parts. The first part initializes the variable to hold the given number of minutes to convert. The second converts the given input into a whole number of hours. The third converts the given input into the whole number of minutes. The last part prints the results.

Listing 6.1 Convert minutes to hours and minutes using the decimal part

```
minutes_to_convert = 123
hours_decimal = minutes_to_convert/60
hours_part = int(hours_decimal)

minutes_decimal = hours_decimal-hours_part
minutes_part = round(minutes_decimal*60)

print("Hours")
print(hours_part)
print("Minutes")
print(minutes_part)
```

Finds the decimal version of the number of hours and gets the whole number of hours by converting to an int type

Gets the part after the decimal point and converts it to whole minutes

Prints the results

The part where you calculate the number of minutes from the decimal number may look a bit intimidating, but you can break it down to understand what's going on. The following line gets the part after the decimal point from the division:

```
minutes_decimal = hours_decimal-hours_part
```

For the example, if `minutes_to_convert` is 123, this calculates as `minutes_decimal = hours_decimal -hours_part = 2.05 - 2 = 0.05`. You now have to convert the 0.05 into minutes.

The following line is made up of two separate operations, as shown in figure 6.2:

```
minutes_part = round(minutes_decimal * 60)
```

First it multiplies `minutes_decimal * 60`:

Then it rounds that result with `round(minutes_decimal * 60)`.

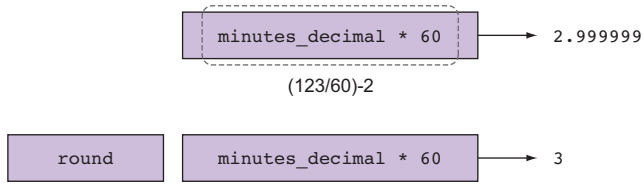


Figure 6.2 The two calculations and their evaluations, in order, on the variable `minutes_decimal`

Why do you need to do all these operations? If you run the program with the line

```
minutes_part = minutes_decimal * 60
```

instead of

```
minutes_part = round(minutes_decimal * 60)
```

you'll notice something interesting. The output is

```
Hours
2
Minutes
2.9999999999999893
```

You expected to see 3 but see 2.999999999893. What's going on? This behavior occurs because of the way that Python stores floats. Computers can't store decimal numbers precisely because they can't represent fractions exactly. When they represent 0.05 in memory, they approximate this number. When you multiply floats, the small differences between their exact value and how they're represented in memory are amplified.

When you multiply 0.05 by 60, the result is off by 0.0000000000000107. You can solve this by rounding your final answer to an integer with `round(minutes_decimal * 60)`.

Quick check 6.4 Change the program in listing 6.1 to find the hours and minutes when starting with 789 minutes. What's the output?

6.5 Your first Python program: another solution

The code for the final program using method 2 is shown in the next listing. The code is separated into the same four parts as the previous solution: initializing, getting the whole number of hours, getting the whole number of minutes, and printing the result.

Listing 6.2 Convert minutes to hours and minutes using the remainder

```
minutes_to_convert = 123
hours_decimal = minutes_to_convert/60
hours_part = int(hours_decimal)
minutes_part = minutes_to_convert%60
print("Hours")
print(hours_part)
print("Minutes")
print(minutes_part)
```

The given number of minutes

Finds the decimal version of the number of hours

Gets the whole number of hours by converting to an int type

Uses the remainder when you divide the number of minutes by 60 to get the whole minutes

The output of this program is as follows:

```
Hours
2
Minutes
3
```

This version of the program uses the remainder idea to give a more concise program in which you don't need to do any "post-processing" to round or convert to integers, as you had to do with the previous method. But good style would be to leave a comment right above the line `minutes_part = minutes_to_convert % 60` to remind yourself that the remainder when divided by 60 gives you the whole number of minutes. An appropriate comment is shown here:

```
# the remainder gives the number of minutes remaining
```

Summary

In this lesson, my objective was to teach you how to put together many ideas to write your first Python program. The program incorporated the following main ideas:

- Thinking about the given task and dividing it into a few smaller tasks
- Creating variables and initializing them to a value
- Performing operations on variables
- Converting variable types to other types
- Printing output to the user

Let's see if you got this …

Q6.1 Write a program that initializes a variable with the value 75 to represent the temperature in Fahrenheit. Then convert that value into Celsius by using the formula c = (f - 32) / 1.8. Print the Celsius value.

Q6.2 Write a program that initializes a variable with the value 5 to represent a number of miles. Then convert this value into kilometers and then meters by using km = miles / 0.62137 and meters = 1000 * km. Print the result in the following form:

```
miles
5
km
8.04672
meters
8046.72
```

Meet Raspberry Pi

One of the great satisfactions of coding is when you can write some code that actually **does** something, particularly something in the physical world. Seeing code that you wrote controlling physical objects is so powerful that people get hooked, going to create more projects that interact with the world to make their lives easier or to just give them joy.

Many people think that in order to connect a computer to objects in the physical world you need expensive equipment or electronic wizardry, but these days that's no longer true. What you need is an inexpensive small single board computer, like the raspberry pi, some simple components, and a power, but friendly, language like Python.

So if you're interested in being a "maker", even on a small scale, the following chapters from *Hello Raspberry Pi* by Ryan Heitz should give you a taste of how much fun this can be. The first chapter introduces you to the pi and how to get it up and running, and the second selection walks you through connecting some LED's and controlling them with a Python script. But be warned – making can be addictive!

1

Chapter 1 from *Hello Raspberry Pi*
by Ryan Heitz

Meet Raspberry Pi

In this chapter, you'll learn how to

- *Set up your Raspberry Pi*
- *Install an operating system—Raspbian—on your Pi*
- *Find and open applications*
- *Write your first bit of code in Python*

What kinds of things do you think you can do with a Raspberry Pi?

1. Play games.
2. Watch videos.
3. Create a video game.
4. Listen to music.
5. Make a sound mixer for a dance party.
6. Build a robot.

Believe it or not, these are all projects you can do yourself, and if you learn to program in Python, the sky is the limit. You can achieve quite a lot on your Pi, as long as you can write a program to do it. But before we talk about that, let's take a look at a Raspberry Pi and discover what makes it tick.

What is the Raspberry Pi?

The *Raspberry Pi*, sometimes referred to as the *Pi*, is a small, low-cost computer invented in the U.K. by the Raspberry Pi Foundation. It provides an easy-to-use tool to help you learn to code in Python (the *Pi* part of its name came from the focus on using it to code in Python).

About the size of a deck of cards, it isn't as powerful as a laptop or desktop computer; its computing power is more similar to that of a smart phone. But what it lacks in processing power, it makes up for in its many features:

- Its readiness for programming in Python
- The many ways you can use it
- Its small size and cost

The Pi, with its companion memory card, is preloaded with all the software you need to jump into programming in Python. Type in commands, and see what happens. Enter a program you find on the internet or in a magazine, run it, and see how it works. The Pi is made for you to learn to code by playing with it, using it, and interacting with it.

Once you learn to program in Python, you can use your Pi as a base for all sorts of projects—with your imagination, the possibilities are endless! The Pi's small size makes it easy to carry around and include in projects. Hide it on a shelf or mount it on a wall with a camera to make a security system; power it with a rechargeable battery pack if you need it to be portable; or even attach it to a remote-controlled car or helicopter. And if you happen to mess something up, it's simple to recover. Even if you manage to break the Pi, it's pretty cheap to replace.

At its core, the Raspberry Pi is a circuit board that has all the components found in many computers. The next section checks out the components of the Pi and explores what they do. Let's go!

Exploring your Raspberry Pi's parts: hardware

Ever look closely at an insect under a magnifying glass, or take apart a toy? Humans are naturally curious about what makes things work. What are the

different parts, and what do they do? What parts are unique? Let's treat the Raspberry Pi the same way, explore its parts, and learn how to set it up.

Luckily, you don't have to break it open to see its parts. You can see the Raspberry Pi's components displayed before you on the green circuit board in your hand (see figure 1.1). Let's walk through the parts of the Raspberry Pi and see what they do. We'll be focusing on the Raspberry Pi 2 Model B; if you have a Raspberry Pi 1 Model B+ or B, see appendix B for more information.

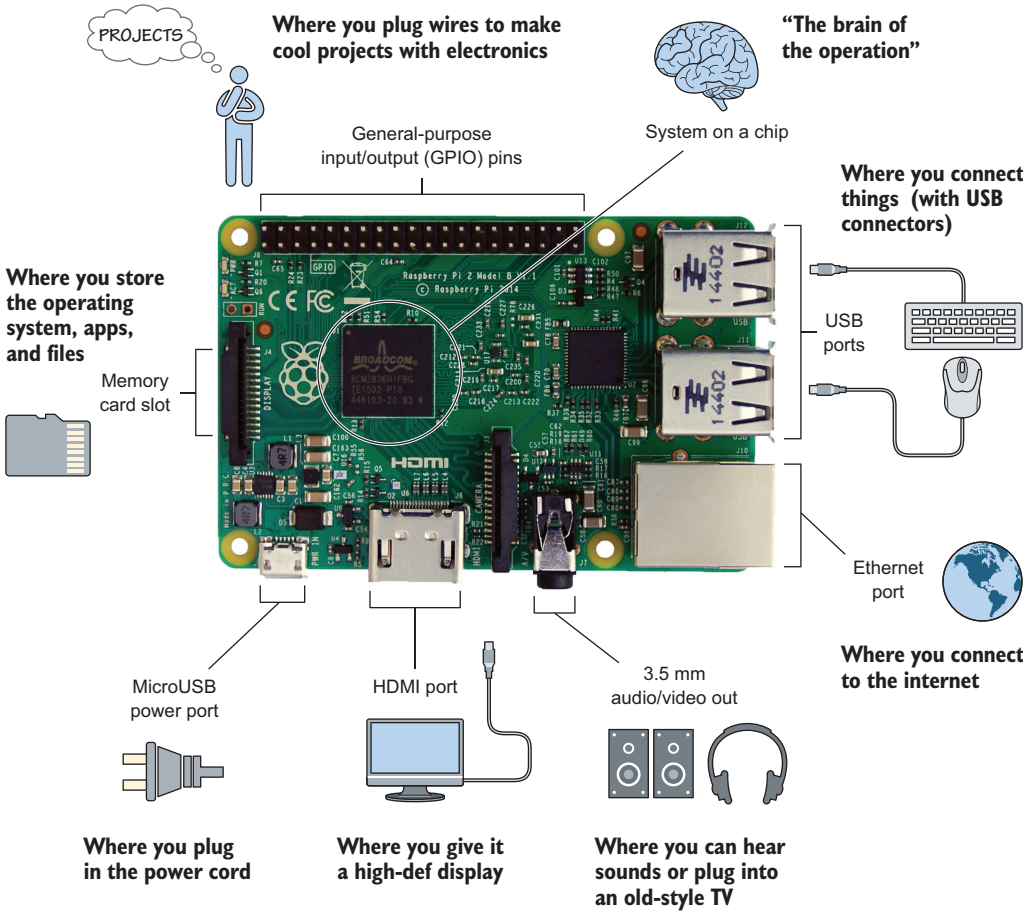


Figure 1.1 The Raspberry Pi provides an excellent platform for learning to program in Python. It includes many input and output ports to give you flexibility in how you connect it. As you would with a desktop computer, you need to connect a keyboard, mouse, monitor, and power cable before you can start using your Pi.

Defining some tech terms

Input and *output* are terms used for communication to and from a computer.

USB refers to a common connector found on computers. It's used to plug in a keyboard, a mouse, flash drives, and many other computer peripherals.

HDMI is a standard way to connect devices to high-definition TVs or monitors. We'll talk about this more later, when we discuss connecting a TV or monitor to your Raspberry Pi.

Ethernet is a technology used to connect computers together into a network. This port provides a way to plug in and connect to the internet or your home network if a wireless connection isn't available.

Giving your Pi a cozy home: Pi cases

We all like to be warm and cozy in our homes. A Raspberry Pi is no different. Do the right thing and protect your Pi by putting it in a case (see figure 1.2). If your Pi didn't come with a case, you have a lot of options. You can buy one or make your own. My favorite approach is to make my own case from wood, cardboard, a plastic container, or even LEGOs. The key is making sure your Pi is protected from accidental drops and, ideally, spills. But before you close up your Pi in a case, let's take a closer look at some of its features.

Figure 1.2 A case protects your Raspberry Pi from damage while making it easy to access the ports. Some people use a case to give their Pi a unique personality. You can purchase a case or, better yet, make your own. Plastic cases are the most common, but these pictures show examples of cases made from paper, plastic, and aluminum. You could even try using LEGOs to make one.

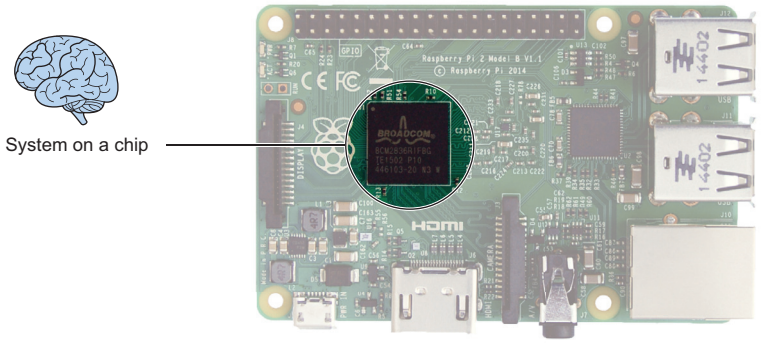


Figure 1.3 The Raspberry Pi's system on a chip (SoC) contains its computing and graphics processing power and working memory. The Pi uses the ARM11 microprocessor as its CPU and the VideoCore IV for its GPU. The ARM11 microprocessor is found in handheld electronics such as smart phones and gaming systems. The SoC in the Raspberry Pi 2 Model B comes with 1 GB of RAM.

The brain of your Pi: system on a chip

Meet the brain of your Raspberry Pi. The *system on a chip (SoC)* is the black square in the middle of the Pi circuit board in figure 1.3. This incredible chip is a package of many parts: the central processing unit (CPU), the graphics processing unit (GPU), the digital signal processor, and the Pi's working memory. The chip provides the computing power, graphics power, and memory to run apps and play videos.

The Pi's CPU handles running applications and executing instructions. The same processor is also found in smart phones and e-readers. Think of it as the part of your brain that allows you to follow instructions and calculate the answer to math problems.

The GPU is like the visual part of your brain that allows you to visualize a 3D object in your mind or track a ball thrown to you. It handles the Pi's multimedia tasks, like processing digital images, drawing graphics, and playing videos. The GPU gives your Pi surprisingly good high-definition video-playback capabilities. Both the central processor and the graphics processor share the Pi's working memory, or RAM, which is part of the SoC.

Working memory: RAM

Question: Can you remember the following grocery store list? *Bananas, milk, peanut butter, jam, bread.* Read the list once more, and then look away from the book and try to recite the list from memory.

To remember it, you need to hold the names of the items in your memory. You only have to store them for a short time. Once you go to the store and buy the items, you can forget them.

When a computer is working, it does much the same thing. It may have to remember and process millions of instructions and bits of information each second, but it can often forget them once it's done processing them. The computer does this using working memory or *random access memory (RAM).* It's packed in the SoC, and it gives your Raspberry Pi the ability to process instructions quickly by remembering pieces of information as it's working and forgetting them when they're no longer needed—much like how the neurons in your brain work together to remember a grocery list. Later, we'll talk about storing information for the long term and where that happens.

Connecting a keyboard and mouse: USB ports

Meet the *USB* ports on your Raspberry Pi. The two metal, rectangular boxes each contain two USB ports, shown in figure 1.4. USB stands for *Universal Serial Bus*.[1] The Pi provides USB ports to allow you to connect a keyboard, a mouse, flash drives, and other USB peripherals.

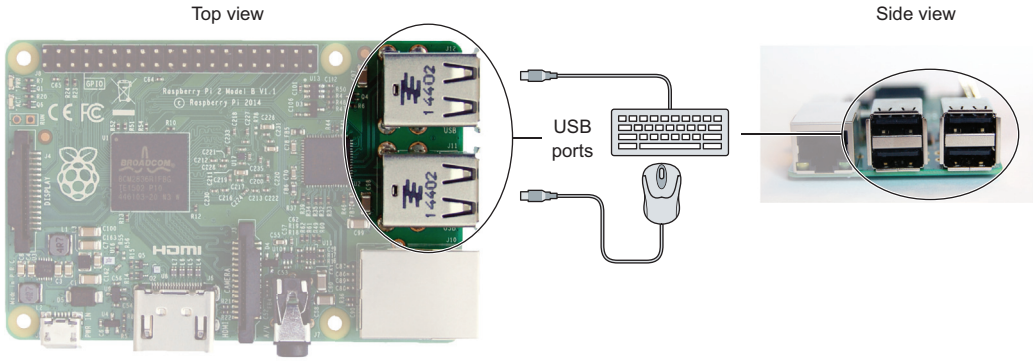


Figure 1.4 The Raspberry Pi 2 Model B has four USB ports. They're on the board in two sets of two, side by side. The USB ports are useful for connecting a keyboard and mouse to your Pi. A USB hub can also be plugged in to allow for even more peripherals.

[1] The *U* for *Universal* is because it provides computer makers and computer equipment makers with a standard way to connect things to computers. Things connected to a computer are often called *peripherals*.

Why are they called ports?

Back in ancient times, when Romans walked around and spoke Latin to each other, the word for a gate or door was *porta*. Although computers don't have doors or gates, they have places where you plug things in, called *ports*.

Ports allow electrical signals to go in and out of your computer. Without ports, you wouldn't be able to view your computer's screen, download web pages, or move a mouse.

Let's pretend you could shrink and that you had special glasses so you could see these electrical signals. What would you see when I pressed the E key on the keyboard? You'd see an electrical signal flying from the keyboard through the keyboard's wire, through the port on the computer, and into the computer. The port acts like a gate, allowing signals to go into or out of your computer.

Get your keyboard and mouse. Let's plug them into your Pi.

CONNECTING A KEYBOARD

You'll need a keyboard that plugs into a USB port. Figure 1.5 shows an example of a keyboard with a USB connector.[2]

To attach your keyboard to your Pi, plug the wire from your keyboard into your Raspberry Pi's USB port. There are four USB ports on your Pi. It doesn't matter which one you choose.

Figure 1.5 You need a USB keyboard to type and enter commands on your Raspberry Pi. The keyboard plugs into one of the four available USB ports on the Raspberry Pi 2 Model B.

[2] If you don't have a keyboard with a USB connector, have no fear. You can find one for under $15 online or at your local computer or electronics store.

> TIP If the keyboard's USB connector doesn't fit into the Raspberry Pi's USB connector, flip over the connector and try again. USB connectors only fit in one way.

Fantastic! Your keyboard is connected to your Pi. It's time to move on to adding a mouse.

CONNECTING A MOUSE

For this step, you need a mouse that plugs into a USB port. The keyboard is using one of your Raspberry Pi's four USB ports. Plug your mouse into one of the other ports.

ANOTHER OPTION: WIRELESS KEYBOARD AND MOUSE COMBINATION

If you own a wireless keyboard and mouse combination, instead of using wires, you can plug the USB dongle into one of the USB ports on the Pi. This frees up one of your USB ports, which can be handy should you decide to attach multiple USB devices such as a USB Wi-Fi adapter or USB flash drives, or if you want fewer wires on your desk.

Excellent! Giving your Pi the ability to store and retrieve information is your next task.

Storing memories: your Pi gets a memory card

We all like to remember things that are important to us. Birthdays, vacations, and holidays are wonderful times, and we've invented ways to help us recall them. You might use a scrapbook or a photo album to store memories. Even after many years, you can open these books and remember these past events.

In addition to working memory (RAM), computers also need a way to remember things, even if they're turned off for long periods of time. The Raspberry Pi, like all computers, has this capability for memory storage, letting it save and retrieve data, files, and applications. Much like a photo album lets you recall holidays, the Pi's memory storage allows you to store important applications and information. You'll use this capability when you learn how to save sets of Python instructions or programs.

SD MEMORY CARD

A Raspberry Pi is different from most computers because its memory storage is contained on an SD memory card, whereas most laptops and desktops use a hard drive. Files, applications, and even the Pi's operating system are all stored on the SD memory card, whether it's a Python game you're creating or a new music player app for your Pi. If you purchase a Raspberry Pi kit, it will come with an SD card (see figure 1.6).[3]

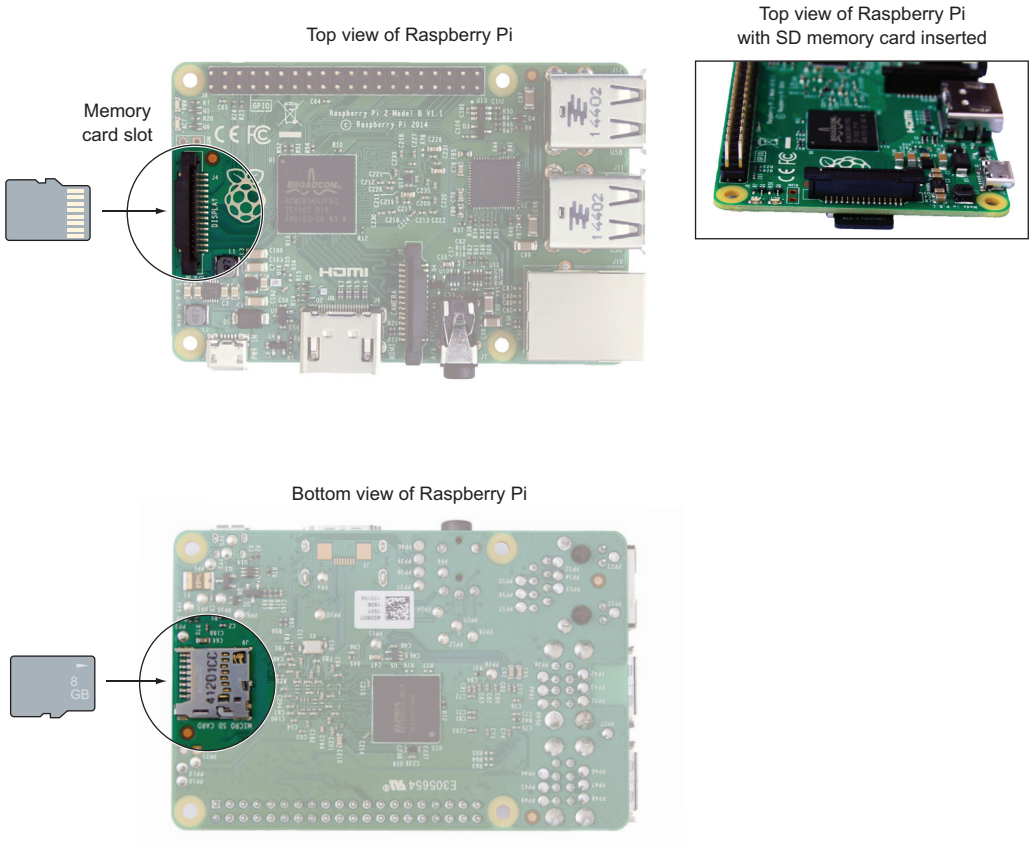


Figure 1.6 An SD memory card provides the storage memory used by the Raspberry Pi to hold all the software and files, including the operating system. Raspberry Pi kits come with an SD memory card preloaded with the software needed to start up your Pi. The two left images show the location of the SD memory card slot on the underside of the Pi board. The right image shows an SD memory card inserted into the SD card slot.

[3] See http://elinux.org/RPi_SD_cards for more information on compatible cards.

SD cards come in various sizes

SD cards come in three sizes: the full-size SD card (largest), the miniSD, and the microSD (smallest). The Raspberry Pi 2 Model B uses a microSD card.

You can add more storage to your Pi by attaching USB peripherals such as a USB flash drive or a USB hard drive.

NOOBS

Your Raspberry Pi kit comes with an SD card preloaded with NOOBS. Developed by the Raspberry Pi Foundation, *New Out of the Box Software (NOOBS)* is a set of files that helps you set up your Pi for the first time. If you lose yours or need a NOOBS SD memory card, you can buy new ones online. Alternatively, if you have an SD card and want to install NOOBS on it, go to the Raspberry Pi Foundation website (www.raspberrypi.org/downloads) to learn how.

SD MEMORY CARD SLOT

Figure 1.6 shows the location of the SD memory card slot. This thin, metal slot is on the underside of the Raspberry Pi. For your Pi to work when you plug it in, it must have some initial knowledge to start up and display something on the screen. In addition to this startup information, it must also have a place to store any new information.

INSERTING THE SD CARD IN THE SLOT

Hold the card so that the end with the metal contacts is facing up and toward the Pi. Insert the card along the underside of the board into the slot. You'll hear a small click as the card is pushed into the slot. The card is held in place by a small spring mechanism. The card will only fit in one way, so if it doesn't fit, flip it over. If you need to remove the card, push it in again (you'll hear a click); then you can pull it out.

REPLACING A LOST OR BROKEN SD CARD

If you lose your SD card, you lose the information, applications, and operating system that are stored on the card. It's as if you lost your hard drive on a

home computer. You can easily replace the card, but you'll be starting over fresh. Here are the two options for replacing the card:

- Purchase an SD card at the store, and set it up anew. It's recommended that you get an SD memory card with at least 8 GB of storage space. You can download and install the startup software from the Raspberry Pi Foundation at www.raspberrypi.org/downloads. See appendix A for instructions on how to make a new SD card for your Raspberry Pi.
- Buy an SD memory card preinstalled with the Raspberry Pi startup software. You can find cards for sale on the Raspberry Pi Foundation website and at online retailers.

SD CARDS MAKE YOUR PI'S MEMORY PORTABLE

If your Raspberry Pi ever breaks, you can remove the SD memory card and insert it into a new Pi. All your files and software will be there. It's like taking your photo album with you to a new house. The memories are safe in the photo album, ready for you to enjoy.

> TIP You can set up multiple SD cards for your Raspberry Pi and switch them whenever you want to give your Pi a whole different personality. Maybe set up an SD card for the Pi as a media center, complete with games, music, and videos. Set up another for your Pi robot project. Each memory card can be set up uniquely, with different operating systems, applications, and files. Swap out the SD card and reboot your Pi, and you instantly have a Pi with different traits to meet your needs.

Connecting a TV or monitor: HDMI port

The *HDMI* port, shown in figure 1.7, is for connecting your Raspberry Pi to a TV or monitor. HDMI stands for *high-definition multimedia interface*. The output provides a combined audio and video signal—meaning both sound and picture come out of this port and go to your TV or monitor. If you want a crisp, clear display and you already own a high-definition TV or monitor, then you'll want to connect your Raspberry Pi to it using the HDMI output port. Because the HDMI output contains audio and video signals, if your TV or monitor has built-in speakers, the sound from your Raspberry Pi can be

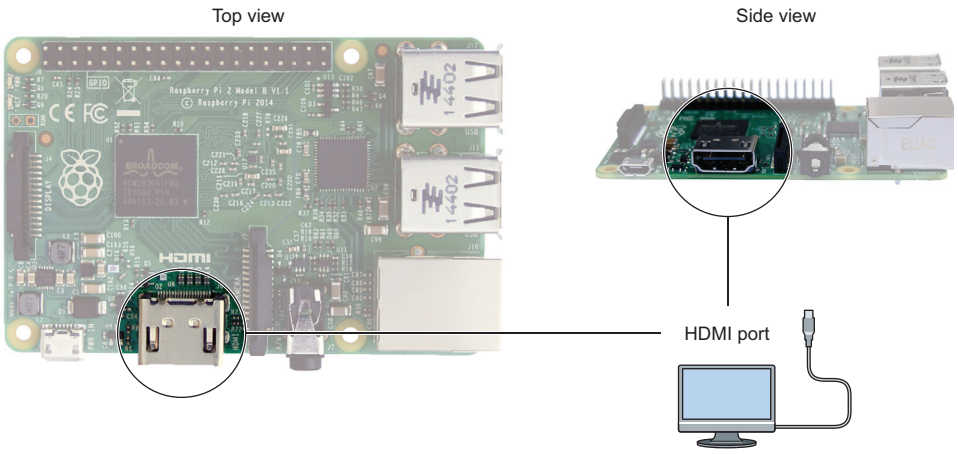


Figure 1.7 The HDMI port on the Raspberry Pi provides a high-definition audio and video signal that can be connected to a TV or monitor. Use an HDMI cable to connect your Pi to your TV or monitor. Depending on the connectors available on the TV or monitor, you may need an adapter.

set to come out of the speakers rather than through the 3.5 mm audio output.

Now that you know about the HDMI port, let's see how you can connect your Pi to a TV or monitor.

CONNECTING YOUR PI TO A TV OR MONITOR

Once you decide on the TV or monitor you plan to use, you'll need to look for the available video input ports on the TV or monitor (look on the back or sides to find them). What kinds of ports do you see? Unfortunately, manufacturers often provide a variety of different ports. Think of it like a matching game. Your goal is to match the connectors on your TV to the connectors on the Pi. If they don't match, you'll need to use one of the adapters discussed in a minute. Either way, you're sure to get it solved.

IDENTIFYING PORTS AND MAKING THE CONNECTION

Take time to study the connections on your TV or monitor. Try to identify the video ports, comparing them to the pictures of connectors in figure 1.8.

This section provides instructions on how you can connect your Pi to a TV or monitor with either an HDMI or a DVI port. If your TV or monitor has different video input ports, check appendix B for tips on connecting to them.

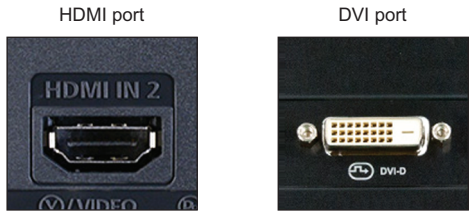


Figure 1.8 HDMI and DVI are common types of video input ports found on modern TVs and monitors. It's easiest to connect a Raspberry Pi to a TV or monitor with an HDMI port. HDMI provides a high-definition picture and doesn't require any adapters or converters—only an HDMI cable, which is included in many Pi kits. The DVI port requires a special adapter to connect with a Pi.

HDMI

The HDMI port is a metal, mostly rectangular port that is labeled *HDMI*. Connect an HDMI cable from the screen's HDMI port to your Raspberry Pi's HDMI port (see figure 1.9). If you've connected your HDMI cable, you can now skip ahead to the discussion of other ports on the Pi.

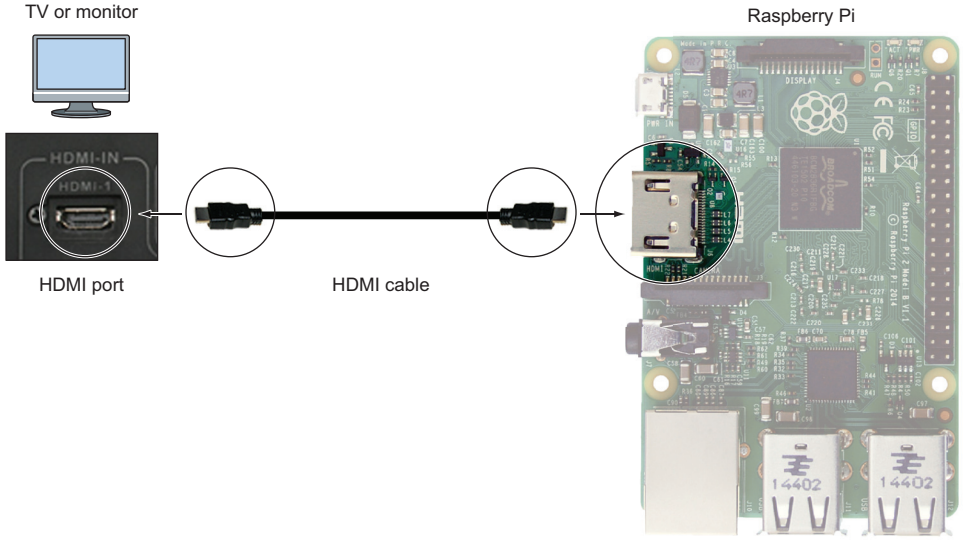


Figure 1.9 A Raspberry Pi can be connected to a TV or monitor using an HDMI cable. Connect the cable from the Pi's HDMI port to the TV's or monitor's HDMI input. In addition to video, the HDMI cable also contains the Pi's audio output, which can be played through the TV's or monitor's speakers.

DVI

DVI ports on TVs and monitors come in several different forms. They're all rectangular ports with three rows of eight square pinholes and a horizontal hole or set of holes next to them. If you already have an HDMI cable, the solution is to purchase an HDMI-to-DVI adapter. You can find these online or in a computer store. Plug the adapter into the computer screen's DVI port, and then plug your HDMI cable into the back of the adapter and the other end into the HDMI port on your Raspberry Pi (see figure 1.10).

Another solution, rather than to use an adapter, is to purchase a DVI-to-HDMI cable. These can be found online or at a computer store. Plug the DVI connector on the cable into your computer screen, and plug the HDMI connector into your Pi's HDMI port.

Great! You've completed an important step by connecting your Pi to a TV or monitor.

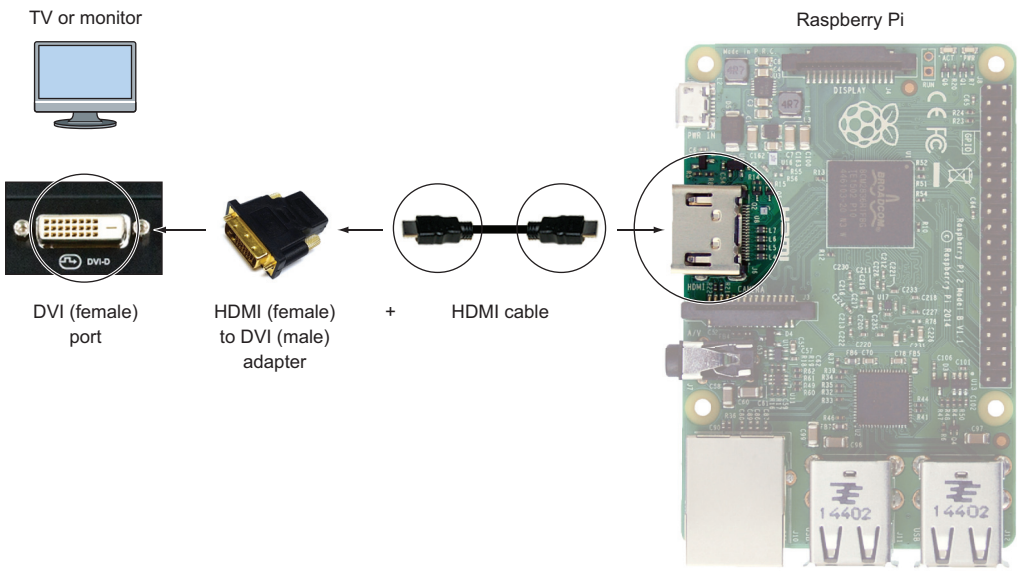


Figure 1.10 The Raspberry Pi can be connected to a TV or monitor with a DVI port using an HDMI-to-DVI adapter and an HDMI cable. One end of the HDMI cable plugs into the Pi's HDMI port. The other is connected to the adapter, and the adapter is connected to the TV or monitor. Adapters are available through online retailers or local computer stores.

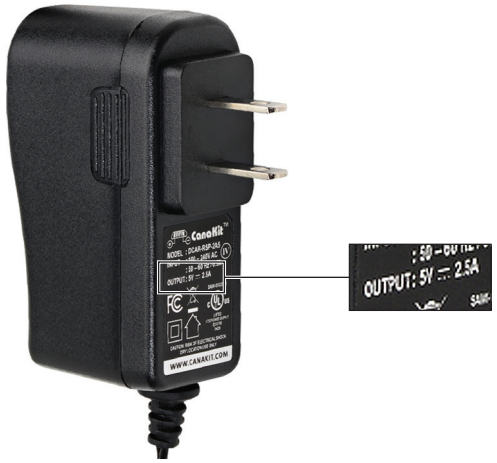
Other ports and connections

You'll find other ports on your Raspberry Pi. We'll cover those in later chapters, or you can reference appendix B for more information on specific ports and connections. Some of these include the following:

- *GPIO pins*—The two long rows of pins on the Raspberry Pi are used to send and receive electrical signals. Part 3 of this book will cover how to program those pins and build projects.
- *Internet*—You can connect your Raspberry Pi to the internet or your home network by plugging in an Ethernet cable. But you may find that the easiest way to get online is to use the USB Wi-Fi adapter that is provided in many Raspberry Pi kits. Appendix B has information on the Ethernet port and using USB Wi-Fi adapters.
- *3.5 mm audio/video out*—The small round connector is for plugging in headphones or powered speakers. Chapter 8 will show you how to play sounds as you turn your Raspberry Pi into a music player.

Let's see how you can get power to your Pi.

Powering your Pi: microUSB power port



Power for your Raspberry Pi is supplied through the microUSB power port located near a corner of the board (see figure 1.11). This port is where you connect a power supply to your Pi; it's the same as the port found on many mobile phones. Raspberry Pi kits come with a microUSB power supply.

Figure 1.11 The Raspberry Pi requires a microUSB power supply that provides at least 1.2 A of electric current. If you plan to use all the USB ports on your Pi, you may want one that provides 2 A or more of electric current. The recommended voltage is 5 volts (V), but the Pi can operate at voltages ranging from 4.8 to 5.2 V. If you have a power supply you want to use with your Pi, check its output voltage and current, which are listed on the charger in small print. In this example, the charger has an output of 5.1 V and 2.5 A of current, making it a suitable power supply for a Pi. Using the incorrect voltage or insufficient current can damage or destroy your Pi, so check carefully.

NOTE Only certain mobile phone chargers can be used to power your Raspberry Pi. The charger must produce sufficient electrical current to power it. If you want to go this route, then you should read the fine print on the charger. The charger must produce 1.2 amp (A) or more for the Pi.

It's alive! Plugging in the Pi

Before plugging your Raspberry Pi into the power supply, go through this quick checklist:

1. Are you sure your keyboard, mouse, and monitor are connected to the Pi?
2. Have you turned on your TV or monitor and set it to the correct input source? For example, if you plugged your Raspberry Pi into the TV's HDMI port, make sure the TV is set to HDMI input.
3. Have you inserted your SD card with NOOBS into your Pi?

An example setup is shown in figure 1.12.

Figure 1.12 Example setup of a Raspberry Pi with peripherals connected and SD card inserted. A keyboard and mouse are connected to the Pi's two available USB ports. A microUSB power supply is plugged into the Pi; the other end is lying on the desk, ready to be plugged into the wall. An HDMI cable is connected from the Pi's HDMI port to the back of the monitor. The Ethernet port has an Ethernet cable plugged into it from a router (not shown).

TIP TVs and monitors often allow you to connect multiple video sources. Maybe your TV has a Wii, a DVD player, and a digital video recorder. These TVs and monitors have the option to select which input is displayed to the screen. Use your TV's or monitor's input selector to set the correct input.

All right, if you have all three steps checked off, it's time to power up your Raspberry Pi. Plug your power supply into a wall outlet, and plug the

microUSB connector into your Pi. Your Pi's lights will begin to flash. Enjoy the beautiful glow from the lights—this is a sign that your Raspberry Pi is starting up. It's also referred to as *booting*; this is when the computer detects the devices you have connected to it and starts up the computer's operating system (OS). Some believe the term *boot* originated from kicking a horse to get it to start moving. You can imagine that you're giving your Pi a bit of a boot to get it started.

Getting your Pi running: software

You've got your Pi plugged in and ready to rock. It's time to get it running and doing something useful—and for that, you need some software.

An OS is a common set of instructions, or software, that helps manage the computer. Common OSs you've most likely encountered are Microsoft Windows, Apple's OS X, and Linux. All of these OSs control the connection of your keyboard, mouse, monitor, and other peripherals. Most important, the OS serves as a foundation for you to put applications on your computer and use them.

The SD memory card that comes with your Pi kit already contains the files for installing several different OSs on your Pi. We'll step through installing the Raspbian OS—the default for the Pi—and configuring it.

Installing the Raspbian operating system

The first time you boot a Raspberry Pi, you'll need to install an OS on it and then configure it to work nicely for you. Let's walk through the first task: installing an OS. You'll configure it in the next section. Once you plug in your Pi, you'll see the NOOBS menu for selecting an OS, as shown in figure 1.13.

The Raspberry Pi has a variety of OSs that can be installed on it. The Raspberry Pi Foundation recommends the Raspbian OS, and it's what we'll use for this book. Let's go over how to install it on your Pi.

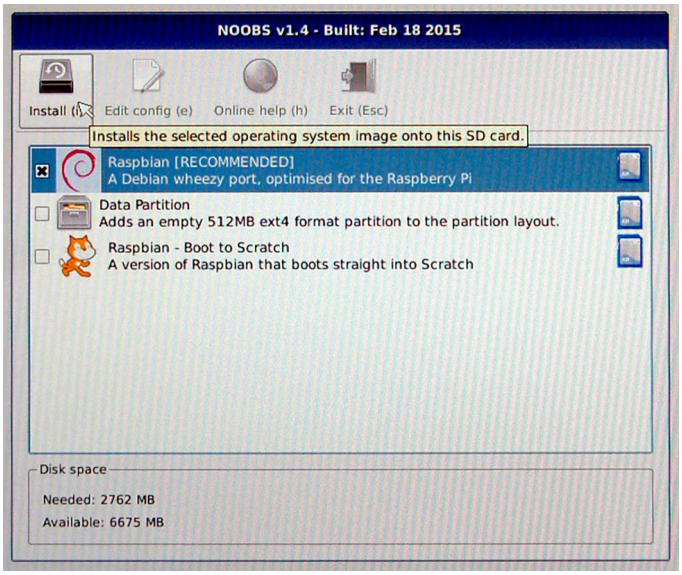


Figure 1.13 The NOOBS selection menu allows you to choose the OS you want to install on your SD card and use with your Raspberry Pi. This menu appears the first time you start up your Pi.

What if you don't see the NOOBS software screen?

If you don't see the NOOBS software screen after your Pi boots up for the first time, then there are a few things to check.

If you don't see lights flashing on your Pi when you plug it in, make sure the electrical outlet you're using has power. Many a Pi owner has accidentally plugged a Pi into a power strip and forgotten to switch on the power strip. Sounds silly, but even the best programmers make mistakes.

If your Pi's lights blink when you plug in the power supply but the screen of your monitor doesn't show anything, make sure the monitor is plugged into an electrical outlet, the HDMI cable is connected from the monitor to the Pi, and you've turned on the monitor.

Finally, if your Pi starts booting up and you see lots of messages displaying on a black screen, but you never see the NOOBS selection menu, it's likely that your SD card has an error. See appendix A for ways to fix an SD card.

Sometimes you'll run into issues with your Pi. If you do, use the troubleshooting steps in appendix A, and search the Raspberry Pi Foundation website[a] to find solutions.

[a] The Raspberry Pi Foundation website is www.raspberrypi.org.

On the NOOBS selection menu (see figure 1.13), follow these steps:

1. Select Raspbian (make sure there is an X in the box next to Raspbian; if not, click the box to select it).
2. Click the Install button at the top of the menu.
3. A message appears, warning you that the process will install the OS and that all existing data on your SD card will be overwritten.[4] Select Yes to continue with the installation.
4. Wait for the installation to complete. It will take 5 to 10 minutes, so get a drink or grab a snack while you're waiting.
5. When the installation is done, a box pops up, letting you know the OS was installed successfully. Click OK, and your Raspberry Pi will start loading Raspbian.
6. When it's finished loading Raspbian, your Raspberry Pi reboots itself. A black screen appears, followed by many, many, many messages. Don't worry; the messages are the Pi performing its startup tasks, such as detecting the keyboard, mouse, and TV or monitor.

Kudos to you! You've installed your Raspberry Pi's OS, Raspbian. Now you'll want to configure how it works to suit you.

Configuring the operating system: making it yours

You've finished installing the Raspbian OS on your SD memory card and gotten it running for the first time. The next thing you'll see is the Raspberry Pi configuration screen, shown in figure 1.14.

> TIP You can't use your mouse with this menu! Use the arrow keys (up, down, left, and right) and Tab key to move around the menu instead. Press Enter to select the highlighted menu item.
>
> Let's walk through some of the basic configuration settings you may want to change.

[4] When you're warned that all data will be overwritten, this doesn't include NOOBS, which is retained on the SD card so that you can reinstall the OS if you ever need to.

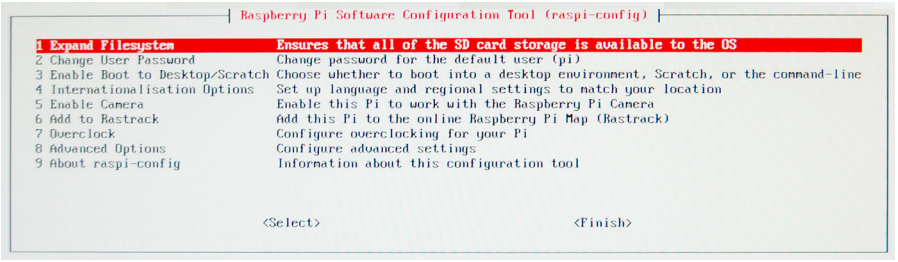


Figure 1.14 When your Pi boots up for the first time, you'll see the Raspberry Pi configuration menu. This menu makes it easier to set up your Pi by allowing you to change settings such as the time zone and keyboard layout. The menu also has the option to set your Pi to always boot to the Raspbian desktop environment.

CHANGING THE KEYBOARD SETTINGS

The Raspberry Pi is made in the U.K., so it's preset to a U.K. keyboard. If you live in other parts of the world, the keyboard may make unexpected characters appear on the screen. For example, you might type a # symbol (Shift-3), and your Pi displays the symbol for a British pound. Weird, right?

You can use the configuration tool to change your Pi's keyboard layout by following these steps:

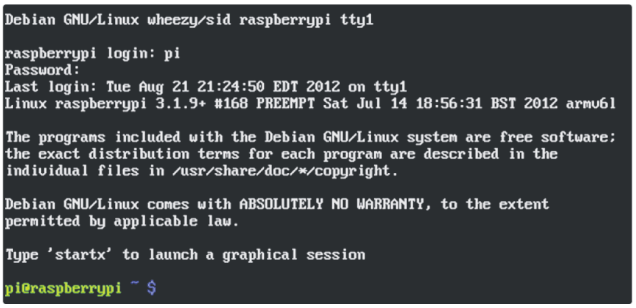
1. On the Raspberry Pi configuration menu, select option 4—Internationalisation Options—and press Enter.
2. Select Change Keyboard Layout, and press Enter.
3. Select your keyboard model—for example, Dell—and press Enter.
4. You see options for the keyboard layout's country of origin. Select the appropriate country, and press Enter.
5. A list of keyboard layouts appears. Select the one for your location, and press Enter.
6. On the next series of screens, you can set shortcut keys. Set them to match your personal preferences. If you aren't sure, accept the defaults (press Enter until you're back to the configuration menu).

You can always return to the configuration tool if needed. You'll learn how in a later section when you're introduced to the command-line mode for Raspbian.

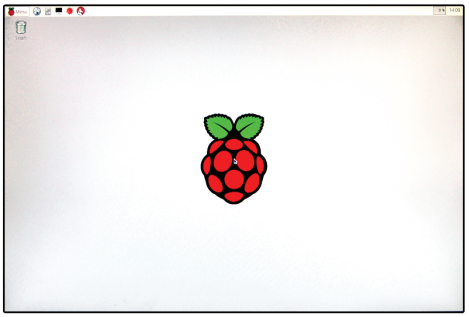
CHOOSING HOW YOUR RASPBERRY PI STARTS UP

Raspbian, like most OSs, allows you to use it in two different ways (see figure 1.15):

- *Command-line mode*—You type in commands to the OS. This can be tough for novices, because you need to know the commands and type them in exactly. Because this mode is more difficult to use, you'll only use it in this book when you need to run commands that require administrative or super-user permissions. For example, you'll need the command line when you make Python programs that use the GPIO pins or you want to alter your Pi's configuration.
- *Graphical-user-interface (GUI) mode*—Everything appears in windows, icons, and menus that are point and click. Just like on Windows and Mac computers, this will be your main way to interact with your Pi and program in Python. It represents the most natural way to access applications, files, and folders.



Raspbian command-line mode



Raspbian graphical-user-interface (GUI) mode

Figure 1.15 Example screen images of a command-line mode (top) and a GUI mode (bottom) for a Raspberry Pi running the Raspbian OS. The command-line mode is text-based: you enter instructions at the prompt. The GUI is pretty much the same as a Windows or Mac interface, with windows, icons, and menus that you interact with using a mouse pointer.

Question: Which option do you prefer?

- Your Raspberry Pi booting up to a screen with a blinking cursor, waiting for you to type in commands
- Your Raspberry Pi booting up and showing you a desktop with application icons arranged on the screen, waiting for you to point to and click them with your mouse

If you chose the second option, you can set Raspbian to always boot to the desktop with the following steps:

1. On the Raspberry Pi configuration menu, select option 3—Enable Boot to Desktop/Scratch—and press Enter.
2. Select the second option—"Desktop Log in as user 'pi' at the graphical desktop"—and press Enter.

Fantastic! Next time your Raspberry Pi boots up, you'll be taken to the Raspbian desktop.

> TIP If you decide you prefer to boot the Raspberry Pi to the command line, you can always launch the Raspbian desktop by entering `startx` at the command line.

> TIP Sometimes you may find yourself using the Raspbian GUI, but you want to use the command line. There is an easy way to change. You can open the command-line mode in a window by clicking the Menu Button, then selecting the Accessories category and clicking the Terminal[5] icon.

MAKING OTHER CHANGES

The Raspberry Pi configuration menu includes other options such as setting up a camera and over-clocking. These are available if you ever want to use them. Check the Raspberry Pi forums for more information on these options.

Saving your configuration and rebooting

If you're happy with the changes made to your Raspberry Pi, follow these steps to exit the Raspberry Pi configuration tool and reboot your Pi:

[5] Terminal is short for LXTerminal or Linux terminal. Raspbian is a Linux-based OS, and *terminal* refers to the command-line mode where you can enter commands.

1. On the Raspberry Pi configuration menu, use the arrow keys to select Finish, and press Enter.
2. You're prompted with this message: "Do you want to reboot now?" Select Yes, and press Enter.

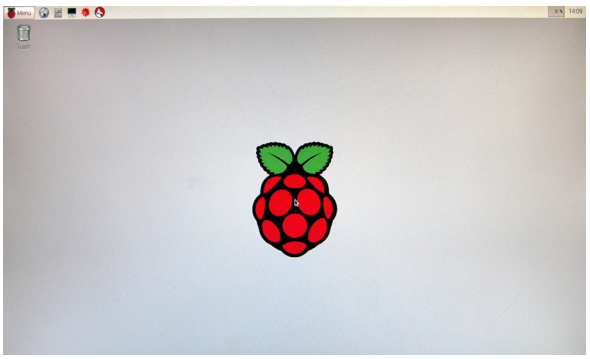


Figure 1.16 A view of the Raspbian desktop after your Raspberry Pi boots up. The desktop is similar to the desktop in Microsoft Windows or Apple Mac OS X. Don't worry if your desktop is different from this one. Depending on when you bought your Pi, you may have received an SD card with an older or newer version of Raspbian.

Your Raspberry Pi will display lots of lines of text as it boots up. (Yes, it does that again! Don't worry, it will seem normal to you soon.) This is your Pi's startup sequence when it connects peripherals and starts up the OS. Next, a white screen with a Raspberry Pi will appear, along with a set of icons—this is your Raspbian desktop (see figure 1.16). Congratulations! Your Raspberry Pi is ready to go.

A BIT OF PI IN YOUR FACE: TROUBLESHOOTING

If you don't see the view shown in figure 1.16, don't be discouraged. It's likely that you didn't select the option to boot to desktop. If your screen shows the command-line mode for Raspbian (figure 1.17), you can log in and launch the Raspbian GUI.

```
[ ok ] Setting up ALSA...done.
[info] Setting console screen modes.
[info] Skipping font and keymap setup (handled by console-setup).
[ ok ] Setting up console font and keymap...done.
[ ok ] Setting up X socket directories... /tmp/.X11-unix /tmp/.ICE-unix.
INIT: Entering runlevel: 2
[info] Using makefile-style concurrent boot in runlevel 2.
[ ok ] Network Interface Plugging Daemon...skip eth0...done.
[ ok ] Starting enhanced syslogd: rsyslogd.
[ ok ] Starting periodic command scheduler: cron.
[ ok ] Starting system message bus: dbus.
Starting dphys-swapfile swapfile setup ...
want /var/swap=100MByte, checking existing: keeping it
done.
[ ok ] Starting NTP server: ntpd.
[ ok ] Starting OpenBSD Secure Shell server: sshd.

Debian GNU/Linux 7 raspberrypi tty1

raspberrypi login:
```

Figure 1.17 If you didn't set up your Pi to boot to the Raspbian desktop, the command-line mode will be displayed when your Raspberry Pi boots up. It will ask you for your login name and password.

At the command line, you'll be prompted to enter your login and password. The default login is `pi`, and the password is `raspberry`. After entering that information, launch the Raspbian Desktop from the command line using the following steps:

1. Type `startx`.
2. Press Enter.

Once you execute the command, the Pi will start up the Raspbian GUI mode and display your Raspberry Pi's desktop. If you happen to have a different problem, head to appendix A for troubleshooting ideas.

Getting around: learning Raspbian

Take a cruise around your Raspberry Pi, and look at some of the applications that come already installed with the Raspbian OS.

Finding and opening applications on your Raspberry Pi

There are many applications on your Raspberry Pi. You can access them by clicking the Menu button in the top-left corner of the desktop (see figure 1.18). Enjoy exploring what comes installed on your Pi.

Your files and folders

Similar to Windows Explorer or Mac Finder, Raspbian has some built-in tools to make it easier to navigate the folders and files on your Raspberry Pi.

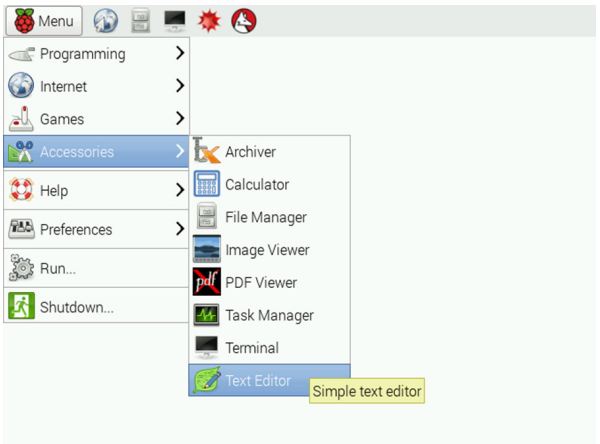


Figure 1.18 The Raspbian application menu opens when you click the Menu button in the top-left corner of the desktop. You can open an application by moving your mouse over the categories listed on the menu and then clicking the application.

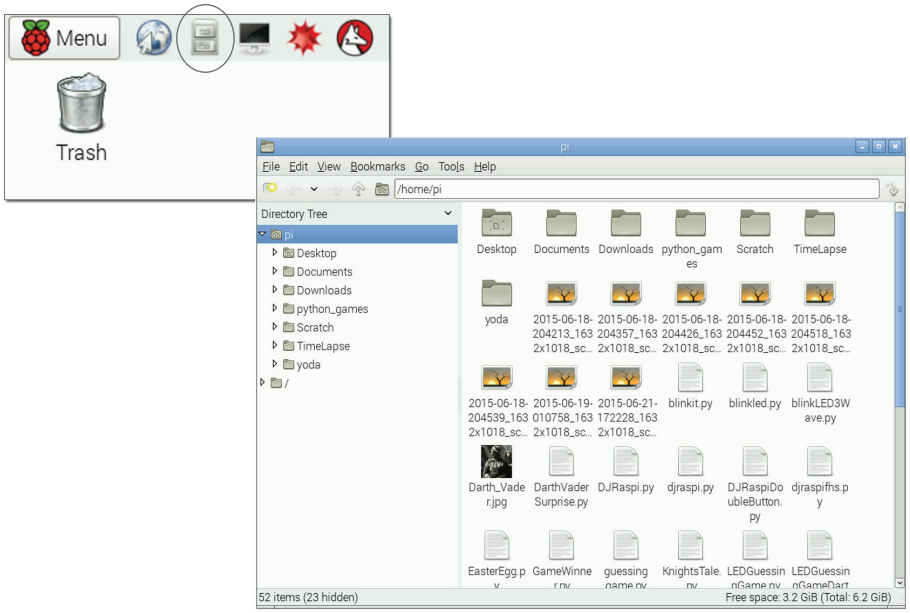


Figure 1.19 File Manager in Raspbian allows you to manage files as you do in Windows Explorer or Mac Finder. You access File Manager using the folder icon in the upper-left corner of the desktop. This is a view of a Pi with a lot of files stored in the /home/pi folder.

In Raspbian, the application for managing files is called *File Manager*, and it's accessed by clicking the folder icon located in the top-left corner of the Raspbian desktop. Figure 1.19 shows the icon and the File Manager application. Just as in Windows Explorer, you can

- Navigate into folders by double-clicking them.
- Drag files to move them to another folder.
- Copy and paste files using the right-click menu on files and folders.
- Rename files.
- Open files by double-clicking them.

The Pi was built for coding. Let's see how you can write code on your Pi.

Writing code

You're going to learn to write code in the Python programming language. Meet a new program, IDLE. IDLE is a tool that'll help you write programs in Python. IDLE stands for Integrated DeveLopment Environment. The Python language was named after Monty Python, and the IDLE acronym is a nod to Eric Idle, one of the founding Monty Python members.

Follow these steps:

Click the Menu button on your desktop.

Select Programming > Python 3.

After a second or two, IDLE opens the Python Shell, as shown in figure 1.20.

NOTE Previous Raspberry Pi models have desktop icons for Python: IDLE and IDLE 3. *You'll use Python 3 (or IDLE 3) for the exercises in this book.* On older Pi models, the IDLE 3 icon opens the Python Shell for Python 3. You may have guessed that the IDLE (without the 3) icon opens IDLE for Python 2.

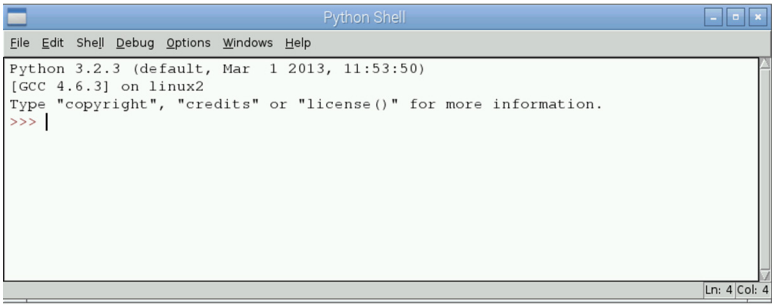


Figure 1.20 IDLE is a development environment that makes it easier to write Python programs. This is the IDLE Python Shell that you can use to enter Python commands or instructions one at a time.

NOTE To start the Python Shell from the Raspbian command line, type `python3` and press Enter. You'll see a `>>>` prompt and may interactively enter Python commands. When you're finished using the Python Shell, `type exit()` and press Enter to end your Python session.

The Python Shell shown in figure 1.20 allows you to enter Python commands and press Enter to execute them. The command prompt lets you type in commands after the triple greater-than symbols (>>>).

Do the following:

1 Enter `3 + 4`.

2 Press Enter.

The screen displays the answer: 7. Try some subtraction:

1 Enter `17 - 9`.

2 Press Enter.

The screen displays the answer: 8. Now let's make Python talk to you by printing a message to the screen:

1 Enter `print("I am alive!")`.

2 Press Enter.

Your screen should display "I am alive!"

Outstanding work! You wrote three lines of code. When you pressed Enter after each one, the Raspberry Pi's processor executed those commands and did what you asked. That is powerful!

Fruit Picker Extra: shopping at the Pi Store

Your Raspberry Pi can do many things. We've included special sections throughout the book called *Fruit Picker Extras* to teach you some different things your Pi can do. This Fruit Picker Extra is about shopping at the Pi Store.

The Pi Store is an online app store that provides access to games, apps, and resources for your Pi (see figure 1.21). You can browse the Pi Store from any device, such as a mobile phone or laptop. To access it from your Raspberry Pi, double-click the Pi Store icon on your desktop. If you want to download content to your Pi, you need to have your Pi connected to the internet, and you'll also need to create an IndieCity account with an email address and password.

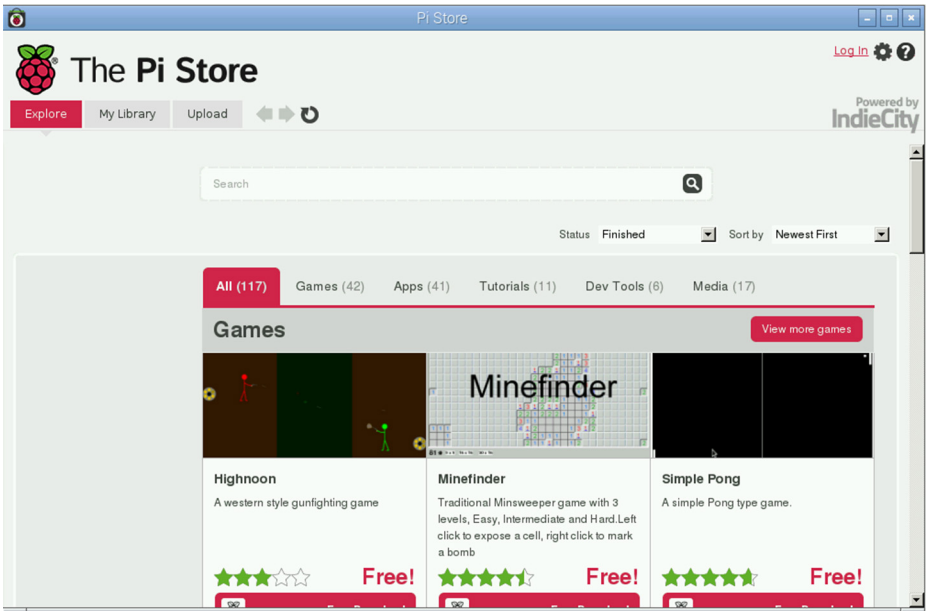


Figure 1.21 You can access the Pi Store from the icon on your Raspbian desktop. The store allows you to browse and download apps and content including games, tutorials, and digital magazines. You'll find free and fee-based content, organized into five categories: Games, Apps, Tutorials, Dev Tools, and Media.

Some apps are free; others require you to pay a fee. You'll find great resources, such as free issues of *MagPi*, the Raspberry Pi community magazine, a digital magazine full of tips, projects, and programming tutorials (look for these in the Pi Store's Media category). Have fun downloading free games and tutorials onto your Pi!

Challenge

Each chapter will have challenges at the end for you to try. If you can't figure them out, check the back of the book (see appendix C) for hints and answers.

Scavenger hunt

Time to explore your Raspberry Pi with a scavenger hunt. The goal is to learn more about the Pi by looking around, opening applications, and playing with them. Try to complete this list of scavenger-hunt items:

1. Find a game where squirrels eat other squirrels. Can you achieve the title of Omega Squirrel? Hint: Double-click the Python Games desktop icon to look for it.
2. Find a calculator application on your Raspberry Pi. Calculate the answer to a math problem: 87×34. Hint: The calculator is found under Menu > Accessories.
3. Without unplugging your Raspberry Pi, can you figure out how to shut down or restart it?
4. Turn your desktop's background black.
5. Bonus: Open Scratch, and try to make a cat dance.

Consider yourself an official Raspberry Pi explorer. If you want, take some more time to click some icons and see what they do. You've accomplished a lot!

Summary

The Raspberry Pi is like other computers in a lot of ways, but with several important differences. The similarities with other computers include these:

- A Pi requires a keyboard, mouse, and monitor, much like other desktop computers. The ports for plugging these in are part of the Pi.
- The Pi can be set up with a desktop OS, Raspbian. It's similar to Microsoft Windows or Apple OS X.
- Although its computing power is limited (similar to a smart phone), the Pi can still allow you to do many things you do on a desktop or laptop, such as browsing websites, playing games, and listening to music.

The Raspberry Pi has qualities and capabilities that make it special and unique. These key differences from other computers include the following:

- The Pi's cost and size are much smaller, making it a great candidate for projects.
- The Pi was designed for programming in Python and comes preloaded with the Python development environment so you can get coding right away.
- The Pi uses an SD memory card to store all files and software, including the OS.
- It has GPIO pins that can send and receive electrical signals. In part 3 of this book, you'll learn how you can use these to create projects that interact with the world around you.

6

Chapter 6 from *Hello Raspberry Pi*
by Ryan Heitz

Blinky Pi

In this chapter, you'll be learning about

- *Giving your Pi the ability to talk to the outside world through connectors to anything*
- *Programming the world outside your Pi with simple electric/electronic circuits*
- *Programming the connectors using your previous Python knowledge to make light patterns*

Setting robots in motion, creating smart homes with sensors, and designing an interactive electronic art exhibit sound like vastly different topics, but they're all things you can do with your Raspberry Pi. In each case, the Pi can act as the brain and interact with the world by doing things like

- Checking a robot's sensors and controlling its motors
- Sensing a room's occupants and adjusting the thermostat or lights
- Controlling sound, motion, and light as part of an art display

In this chapter, you'll set up your Pi to control small light bulbs called *light-emitting diodes (LEDs)*. You'll make the LEDs blink using Python. To do this, you'll need to learn a bit about how to build electrical circuits on breadboards. If you've never heard of a breadboard, don't worry! It's a small board with lots

of holes in it to make it easier to build electrical circuits. You'll also be using short wires (called *jumper wires*) to connect certain holes. You'll even learn how to add resistors that keep your LEDs from burning out. See figure 6.1 for a list of parts and what they look like; gather the parts, and let's get started!

Figure 6.1 The Blinky Pi project requires parts that are commonly found in Raspberry Pi starter kits or that can be purchased online.

Setting up your Pi for physical computing

Your Pi is unique compared to most computers because of its input and output pins, called *GPIO* pins. Let's learn how to work with those pins.

> DEFINITION GPIO stands for *general purpose input and output*. These are the pins on your Raspberry Pi that allow it to sense and control things around it.

GPIO pins

The Raspberry Pi 2 Model B and Raspberry Pi 1 Model B+ have 40 pins located on the edge of the board, arranged in 2 rows of 20 pins each (see figure 6.2). Most of the pins on a Pi are used for input and output, so they're often referred to as the Pi's GPIO pins.

> WARNING This project is written for Raspberry Pi 2 Model B. Earlier models of the Raspberry Pi have only 26 pins. See appendix B for information about the differences from the more modern Pi boards. To complete this project with a Raspberry Pi 1 Model B, you may select different pins to light up your LEDs.

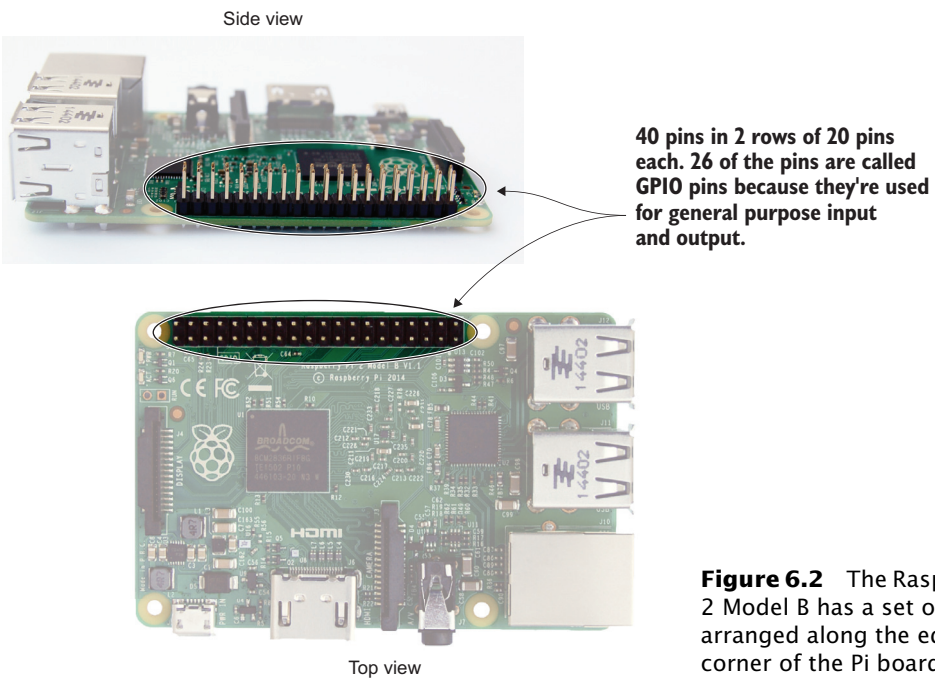


Figure 6.2 The Raspberry Pi 2 Model B has a set of pins arranged along the edge and corner of the Pi board.

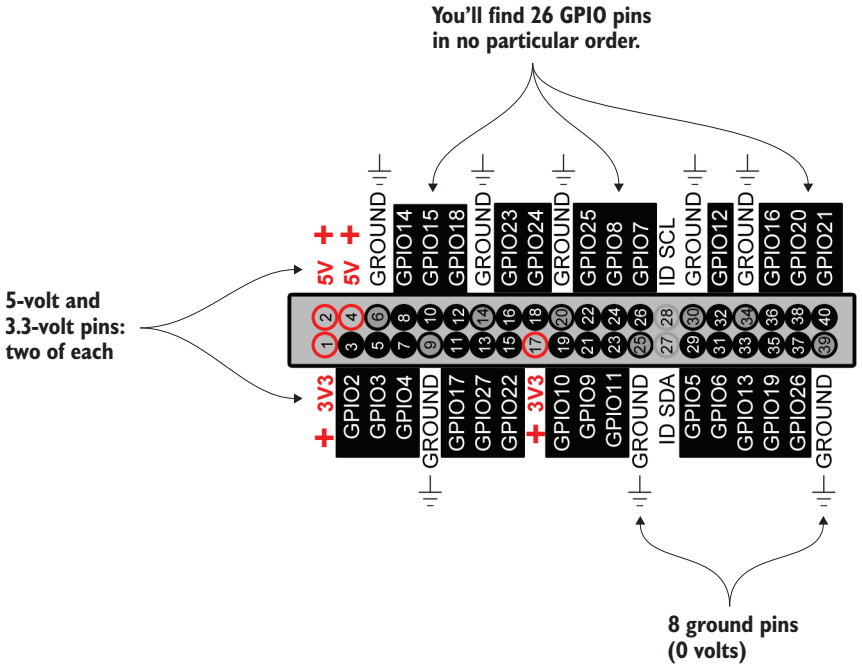


Figure 6.3 The Raspberry Pi B+ has 40 pins. They do different things: some provide 5 volts or 3.3 volts, some are ground pins (0 volts), and many of them are input and output pins that you can program.

Because all the pins look identical, you need a key or diagram to tell you what each one does. Figure 6.3 shows the pins labeled.

Physical pins vs. GPIO pin numbers

In this book, we'll always refer to the GPIO pin numbers, *not* the physical pin locations. The physical pins are numbered from 1 to 40 (shown in the circles in figure 6.3). The GPIO pin numbers go from 1 to 26, and those numbers don't match the physical pin numbers. For example, GPIO 24 corresponds to physical pin 18. By always using the GPIO numbering, it will be easier to wire your circuits and create programs.

Wow, that's a lot of pins! Some pins are for power and are labeled either 3V3 or 5V. These produce 3.3 volts or 5 volts, respectively. There are also 8

ground pins and 26 GPIO pins[1]—26 pins, just like there are 26 letters in the alphabet.

The GPIO pins support sending out electrical signals (output) or listening for electrical signals from sensors (input). In your body, your brain can send signals to your hand to smack yourself on the forehead (try it!)—this is just like the output from a Pi. Signals are sent out of your Pi to make something happen in the world.

The opposite of output is input. When someone pokes you, your body can detect that poke using nerves in your body. An electrical signal (input) is sent to your brain so you know you've been poked. This is like the way your Pi can be used to detect input or actions in the world.

You'll learn how to output signals in this chapter and chapter 7. Chapter 8 will cover detecting input from the world, such as detecting when a button has been pressed.

Let's get ready to connect some wires! But wait: connecting an LED directly to the GPIO pins on the board of your Raspberry Pi isn't feasible, because the pins are so close together. What can you do? You need more space to build circuits.

Breaking out the GPIO pins to a breadboard

To give you room, you'll move the GPIO pins over to a breadboard. This is called *breaking them out*. To do this, you need a ribbon cable, breakout board, and solderless breadboard (see figure 6.4).

Breadboards make it simple to prototype circuits. Like a park might provide large, open fields that make it easy to play sports, think of a breadboard as a nice, open electrical playing field where you can play with electrical parts. The breadboard allows you to plug wires and components into small holes. You can build and rebuild circuits on a breadboard with little effort.

[1] Oddly, you'll notice that the GPIO pins are numbered from 2 to 27. Pins 0 and 1 are used for communicating with other computer chips using a super-special protocol called I2C. These are labeled ID SDA and ID SCL in figure 6.3.

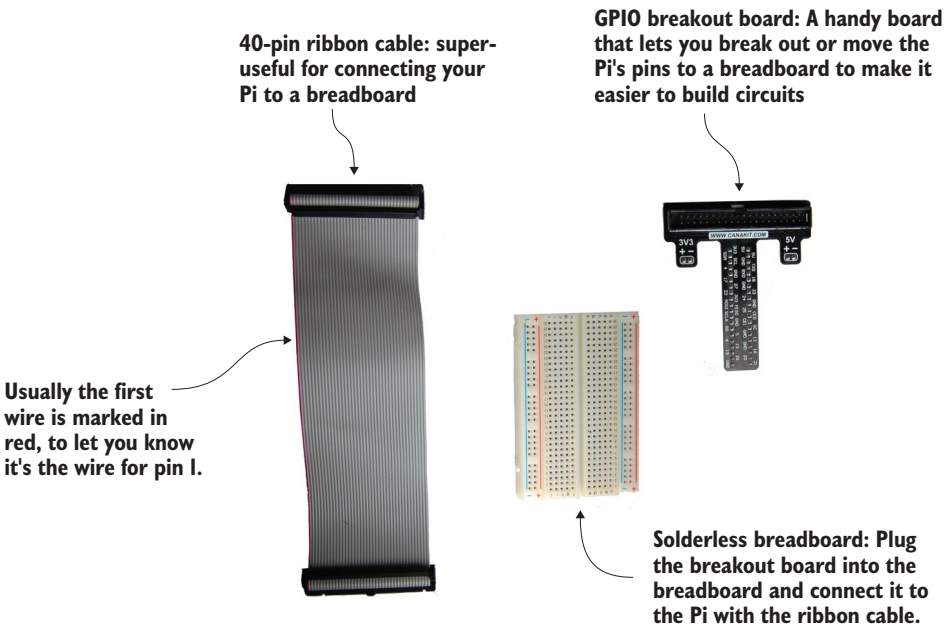


Figure 6.4 To easily create projects using your Pi's GPIO pins, you can connect the Pi to a breadboard using a ribbon cable and breakout board. The parts shown are examples of the ones commonly found in many Raspberry Pi kits.

Find your breakout board, and insert it into the top of the breadboard. Line up the pins before you push it down *hard* (see figure 6.5). Your particular breakout board may look a little different, but they all act the same. With the breakout board in place, it'll be easier to build circuits with your GPIO pins.

Connect one end of the ribbon cable to the Pi's GPIO pins; line it up carefully before you push it down. Then connect the other end of the cable to the breakout board on your breadboard (see figure 6.6). A breakout board has a notch in it so the ribbon cable will only fit one way.

> WARNING Ribbon cables usually have a stripe that marks the first wire. White or grey ribbon cables often use a red stripe. Black ribbon cables often have a white stripe. These mark the first wire on the cable. Make sure this first wire is connected toward the edge of your Pi's board and away from the USB ports.

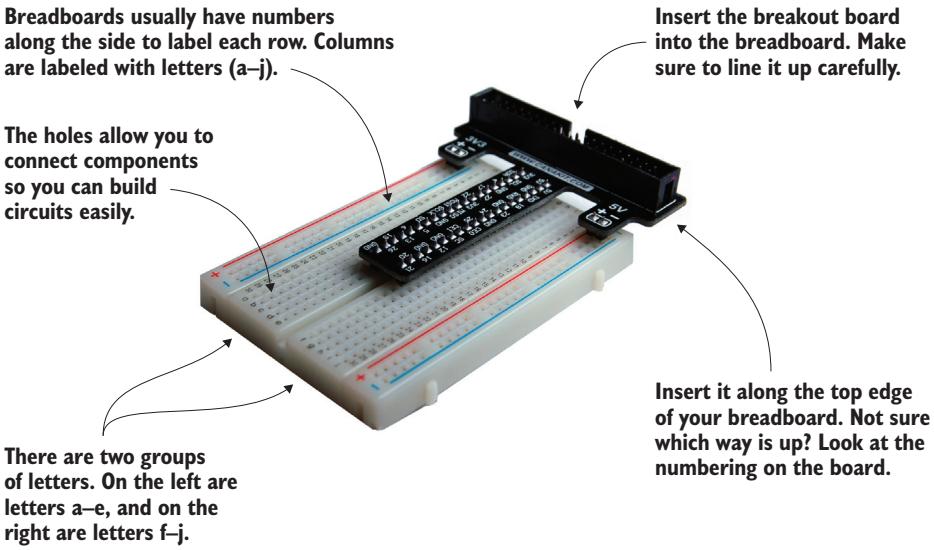


Figure 6.5 Carefully line up the breakout board, and then press it firmly into the breadboard. The two rows of pins on the breakout board should straddle the center gap.

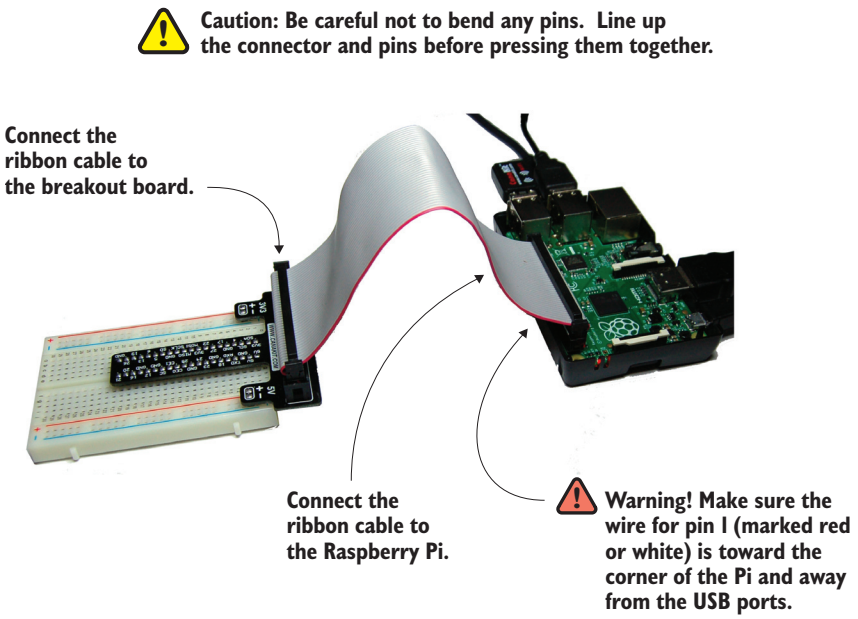


Figure 6.6 Connect one end of the ribbon cable to the breakout board. Connect the other end to your Raspberry Pi.

Breadboard basics

A breadboard[2] has a set of internal connections that you can't see. But if you had X-ray vision, you'd see that certain holes are connected. Let's look at the connections in your breadboard (see figure 6.7).

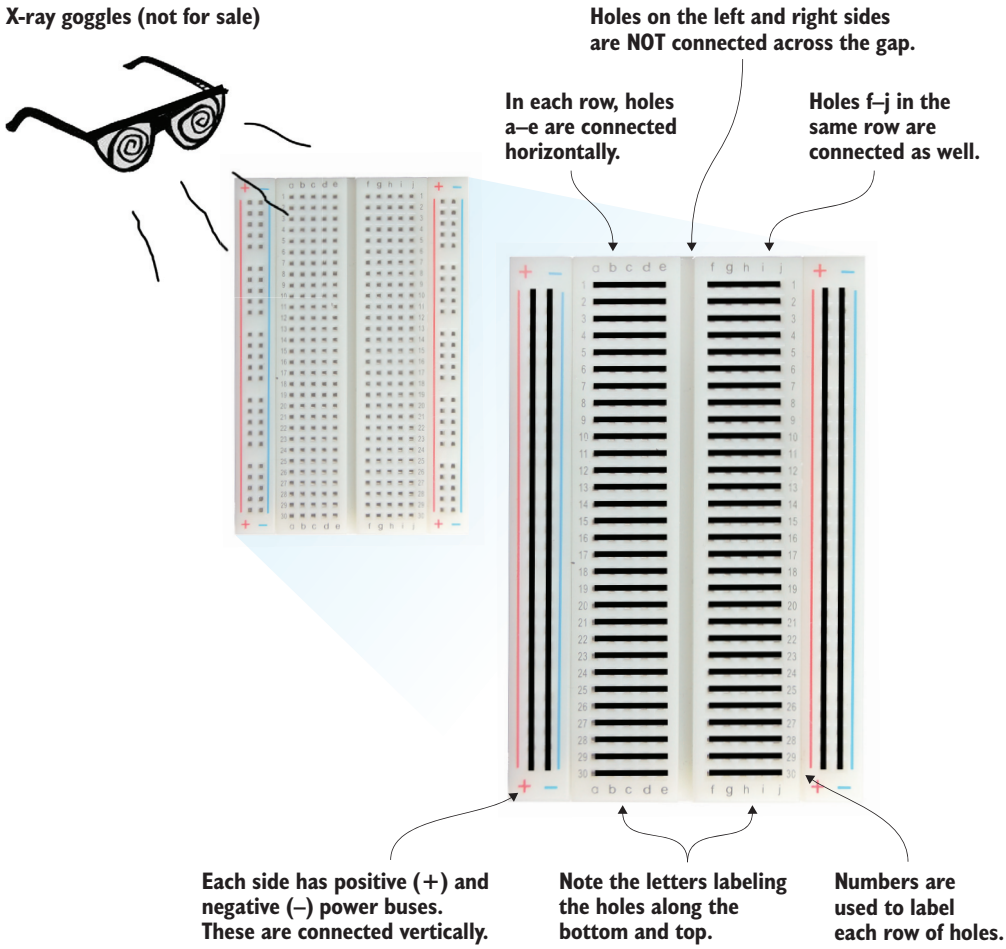


Figure 6.7 Breadboards have internal connections. You need to know about them in order to build circuits. Rows of pins are connected horizontally, but not across the gap in the middle. Long rails called *power buses* run vertically along the sides of the board.

[2] Prior to the development of the kind of breadboards we're using, people built circuits on pieces of wood that were used to cut bread on (hence the name). They needed a quick way to connect circuits, and by drilling holes and using nails and wires, they could use bread boards to try different circuits.

On this breadboard, rows are labeled with numbers (1–30), and the columns have letters (a–e on the left side and f–j on the right side). You can refer to a specific hole in the breadboard by saying its row number and letter. For example, if you wanted to refer to the hole located in row 25, column c, you could say 25c (see figure 6.8). Just as you might find your seat at a stadium by walking along the aisle to find the correct row, and then moving along the row to find the right seat, you'll use the letters and numbers to guide you in building your circuits.

BREADBOARD (BB) HOLES

We'll refer to the row and column, but we'll prepend the letters *BB* so you know it's the breadboard location we're talking about. Figure 6.8 shows the location of BB25c. If we're talking about a GPIO pin or connection, we'll add *GP* before the number (GPIO pin 21 is GP21).

Try to keep in mind what is connected in a breadboard and what isn't. If you forget, you can always look back at figure 6.7. For example, notice that

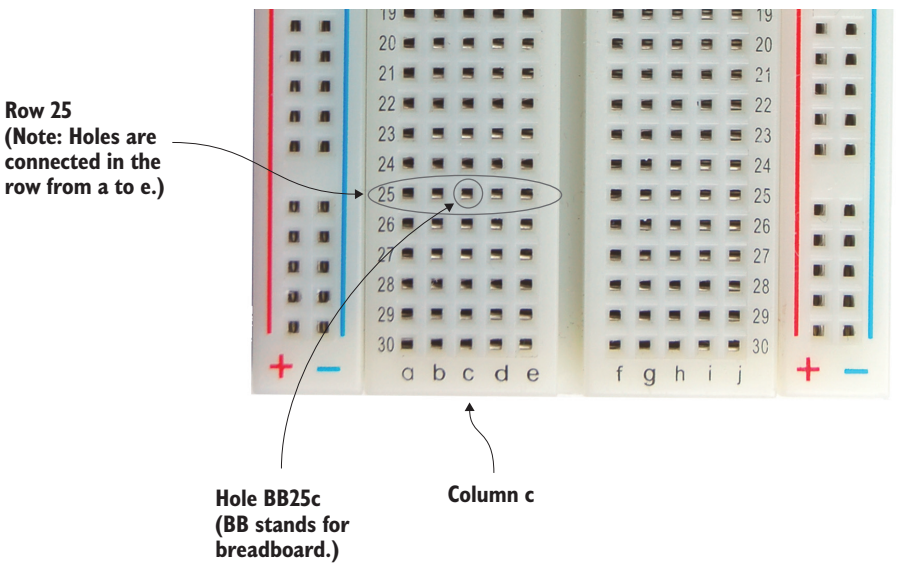


Figure 6.8 To find a specific hole on a breadboard, use the row and column labels. This is a close-up of a breadboard, showing how you can find the location of hole 25c (we'll refer to the hole as BB25c, where *BB* stands for *breadboard*).

BB25 a, b, c, d, and e are all connected. Similarly, BB30 f, g, h, i, and j are connected. But the left side of the board isn't connected to the right. For example, BB25e isn't connected to BB25f. To connect them, you'd put a jumper from BB25e to BB25f.

You can see vertical columns of holes along the sides of the breadboard. These are the *power buses* and provide easy ways to connect electrical components to power (positive) and ground (negative).

Circuits 101

Let's learn about electricity and circuits. At the simplest level, a *circuit* is a loop or path where the electrical power starts at a source (the positive side of a power source), goes through one or more electrical components (such as a light or motor), and then completes the loop (or path) by connecting back to the negative side of the source.

What is electricity?

Electricity is the flow of charge. Typically, it is the flow of electrons, which have a negative charge. To get electrons to flow, you need to have a difference in charges. Just as the north pole of a magnet is attracted to its opposite—the south pole of another magnet—positive and negative electric charges are attracted to one another. If the charge is free to move, it will move. We generally think of circuits as having electricity flowing from the positive (+) side of the source to the negative (-) side of the source. For your Pi, the power is coming from the power supply (Micro USB plug). The Pi as a power source can provide either +3.3 volts or +5 V (volts). It provides this power through the physical pins 1, 2, and 4, but can also send +3.3 V out any of the 26 GPIO pins (you'll program it to do that soon).

Voltage (volts)

Voltage is a measure of the difference in electrical charge between the positive and negative source. When you have two different charges, they're attracted to one another (positive and negative attract). The greater the difference in charge, the greater the force (or electrical pressure) wanting to move charges through the circuit from the positive side to the negative side.

Voltage is measured in volts (V), named after Alessandro Volta, who is credited with inventing the first battery. A 9 volt (or 9 V) battery has a greater electric force for moving charge than a AA battery, which only has a voltage of 1.5 V.

Current (amperes)

The *current* in a circuit is the amount of charge flowing. So whereas voltage is a measure of how badly charges *want* to flow, the current is a measure of how much charge is *actually* flowing.

Imagine that you could be inside a wire and see the charge flowing through it. A large current would mean a lot of charge (usually electrons) bumping along and

through the wire over some period of time. A small current in that same wire would mean a lot less charge flowing over that same time period. Current is measured in amperes (A), named after André-Marie Ampère. A current of 1 ampere (or 1 A) is equivalent to the amount of charge of 6.241×10^{18} electrons flowing through a wire per second! That is a lot of charge flowing. You can decrease the current in a circuit by increasing the resistance of the circuit to the flow of electric charge.

RESISTANCE (OHMS)

The *resistance* in a circuit is a measure of how much it opposes the flow of charge (current). A light bulb, a motor, and your body all have resistance. The opposite of resistance is *conductance*. Substances such as metal (copper, silver, and gold) are all good conductors, and this is why we build circuits with metal wires for the electricity to flow through.

Sometimes you need to control the current (the flow of charge). Resistors are used to do this; they're made of materials that slow down the flow of charge. The most common ones are made out of carbon (you'll be using these in your projects). The resistance of a circuit is measured in ohms, named after Georg Ohm, and is represented using the Greek symbol omega (Ω).

PI CIRCUITS

You can think of your Pi as providing 3.3 V from the positive side of the Pi or, later, coming out of one of the GPIO pins. This +3.3 V is a force that is trying to push electric charge to the negative (-) side of your source. The negative side is sometimes called the *ground*—think of it as a big sink or reservoir to which electricity wants to flow if there is a path to get there. During the next few chapters, you'll build circuits with LEDs and resistors. You use a resistor with an LED to decrease the flow of electric charge (the current) so it won't be too large and burn out your LED. Burning an LED smells bad!

On your breadboard, think of all the GPIO pins as potential sources of voltage (positive). Circuits from the GPIO pins should end back at any one of the many ground (negative) connections.

Building the LED circuit

Your first project is to light up a red LED. You'll control the LED using GPIO pin 21 (GPIO21). You need these parts:

- Raspberry Pi, ribbon cable, and breakout board connected to your breadboard
- 1 red LED (5 mm)
- 1 180 ohm resistor
- 1 jumper wire (male-to-male)

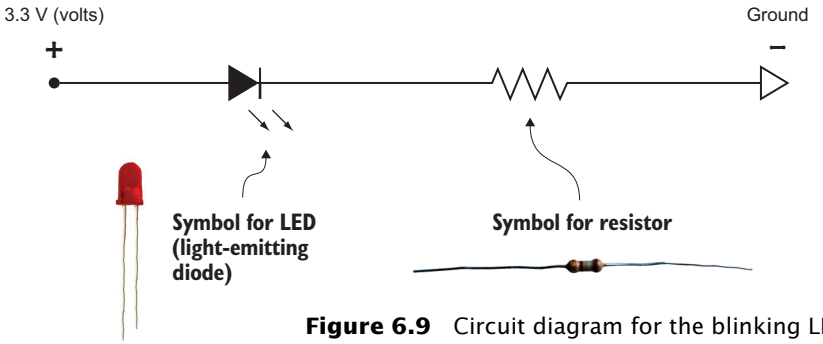


Figure 6.9 Circuit diagram for the blinking LED project

You'll build the LED circuit on your breadboard and then program it to light up. Figure 6.9 shows the circuit diagram. To light the LED, you'll have electricity (+3.3 V) flow from your Pi's GPIO pin 21 through the LED, through the resistor, and then to ground (0 V).

Figure 6.10 shows the LED circuit built on the breadboard. Note that there are many different ways to create this circuit—this is just one way. Let's walk through the steps to build the circuit.

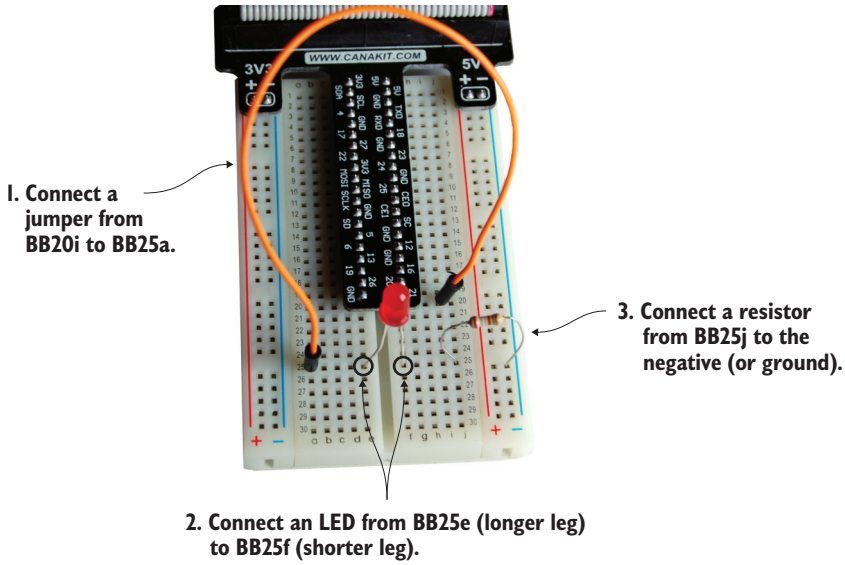


Figure 6.10 LED circuit built on the breadboard. You're using GPIO pin 21 as the power source. The light won't turn on until you program the voltage to come out of the pin.

NOTE You may have a different breadboard than the one used in this book. If so, the numbering on your breadboard may be different than what is shown here. In that case, you'll need to create the circuit following the same principles, but with different numbered holes.

Step 1. Connect the jumper from GPIO pin 21

Raspberry Pi GPIO pins can output 3.3 V. You could pick any pin, but this project uses GPIO pin 21.

Connect a short piece of wire from GPIO21 on your breadboard to an empty row on the breadboard. Use row 25. Firmly push the wire into the hole. The metal tip of the wire should go down into the hole, not sit on top.

The breakout board pins are connected to rows on the breadboard. We'll refer to the holes on the breadboard (see figure 6.11). Insert one end of the jumper into BB20i and the other end into BB25a.

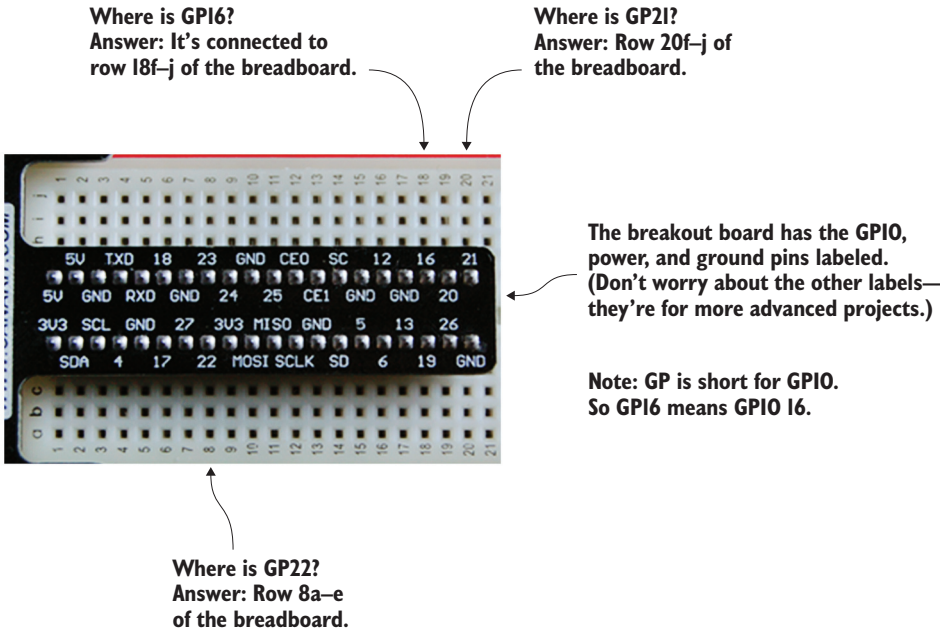


Figure 6.11 The breakout board has labels that correspond to the pins on your Pi. To connect a wire to GP16, you plug it into the breadboard in the hole labelled BB18f or BB18j.

Step 2. Add the red LED

It's time to connect the red LED. LEDs only let electricity flow through them one way, so it's important to put them in the right way. LEDs have two wires or *legs*. The longer leg is called the *anode* and connects to the positive side of the circuit (see figure 6.12). The shorter leg, called the *cathode*, connects to the negative or ground side of the circuit.

With the red LED, connect the longer leg to BB25e and the shorter leg to BB25f. You may need to bend the legs and push them a bit to get them into the holes.

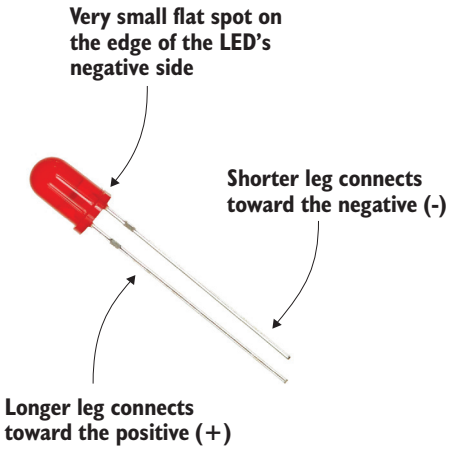


Figure 6.12 LEDs have two legs (wires) coming out of them. The longer leg is called the anode and connects to the positive side of the circuit. The shorter one is called the cathode and connects to the negative side of a circuit.

Step 3. Connect a resistor

Grab your 180 ohm resistor.[3] You can identify a resistor by its color-coded bands. A 180 ohm resistor has colored bands of brown, grey, and brown (see figure 6.13). They are followed by a fourth band that indicates the tolerance

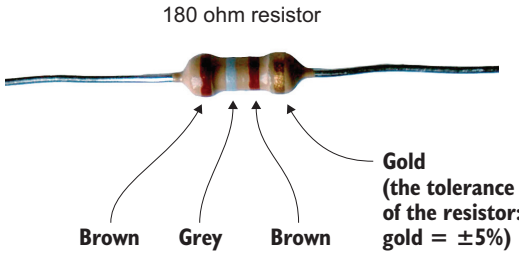


Figure 6.13 The value of a resistor is determined by its colored bands. See the sidebar "Resistor color codes" for a chart; there are also many online color-code charts.

[3] If you don't have a 180 ohm resistor, you can use a resistor with a value between 100 and 330 ohms. If you use a resistor with a value that is too large, the LED may not light up or will be dim. Try experimenting with different resistors to adjust the brightness.

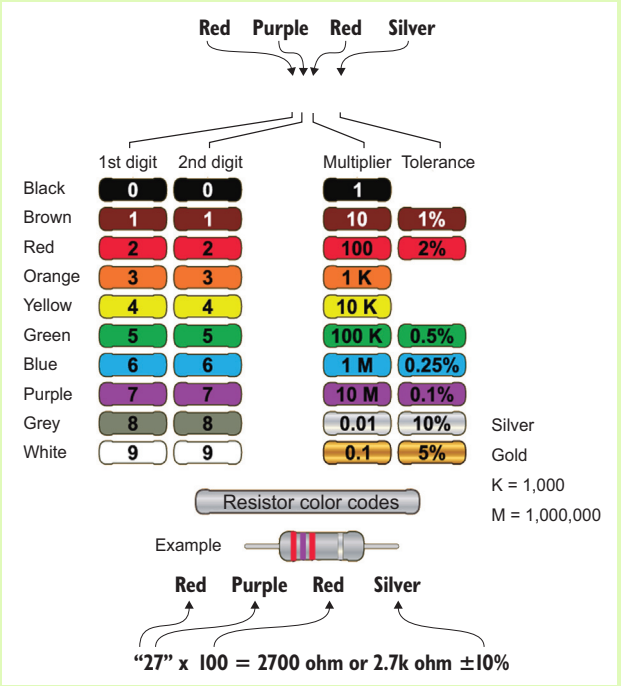
or quality of the resistor. Common colors for the fourth band are gold (±5% tolerance) and silver (±10% tolerance).

The resistor prevents too much electric current[4] from passing through your LED and burning it out. Insert one end of the 180 ohm resistor into BB25j and the other end into the negative (-) power bus (or ground).

Electricity will flow either way through a resistor, so which way you connect it doesn't matter. Remember that the negative power bus or ground rail is running vertically along the right side of the breadboard. Most boards have a blue stripe next to it.

Resistor color codes

Resistors have color codes that tell their value and tolerance. This chart shows you how to read the resistor color bands.



[4] Current is a measure of the flow of electric charges per second. If the current through an LED is too high, the LED will burn out.

(continued)

For example, consider a resistor with red, purple, red, and silver bands. Follow these steps to use the chart:

- Look up the digit for the first band and the digit for the second band, and put them together. In this case, the digits are 2 and 7: put them together, and you get 27. Note that you don't add the numbers; you treat them as the first and second digits of the resistor value.
- Find the multiplier by looking up the color for the third band. In this case, it's 100 ohms (red).
- Put it all together: 27 × 100 ohms is 2,700 ohms or 2.7K ohms (K = 1,000).
- The fourth band (silver) tells you the resistor has a tolerance of ±10%.

A red, purple, red, and silver resistor is a 2.7K ohm resistor with a ±10% tolerance. Use this handy chart any time you need to look up the value of a resistor.

That's it! You have a completed LED circuit built on your breadboard. Now it's time to program it!

Software: blinkLED program

Open IDLE by choosing Python 3 under Menu > Programming. This opens IDLE to the Python 3.x Shell. In the Python Shell, let's check to see if your Pi has the GPIO libraries you need already installed:

```
>>>  import RPi.GPIO as GPIO
```

If you don't see an error, you're ready to go. If you see an error saying there is no module named `RPi.GPIO`, please refer to the sidebar "Updating your Pi."

Updating your Pi

Before programming, you need to check that your Pi is up to date. Make sure your Pi is connected to the internet. Open the Terminal program by going to Menu --> Accessories --> Terminal, and run the following commands to update your Raspberry Pi and be certain you have the Raspberry Pi GPIO packages you need.

First, let's update the `apt-get` database. The `apt-get` program handles installing and removing software from your Pi. In Terminal, enter this command:

```
pi@raspberrypi ~ $ sudo apt-get update
```

You'll need to wait while a bunch of files are downloaded and installed. You'll see lots of messages displayed in Terminal. When the command completes, you'll see the Terminal $ prompt again. Next, to get the latest Pi software, enter

```
pi@raspberrypi ~ $ sudo apt-get upgrade
```

Once again, files will be downloaded and installed. After a series of messages, you'll see a warning about the upgrade using additional disk space, and this prompt: "Do you want to continue [Y/n]?" Enter Y and press Enter to continue the upgrade.

This is a great time to grab a sandwich and soda. It can take 15 minutes or more for the update to complete. When it's finished, you'll have the latest Raspberry Pi software and Python libraries, including the ones you need to communicate with and control the GPIO pins.

You're going to write a program that blinks an LED. It'll send a voltage (+3.3 V) out of a GPIO pin to light the LED, then turn it off, and repeat that over and over. Begin by creating the following new program in IDLE. In the Python Shell, start a new program by pressing Ctrl-N or selecting File > New Window.

Listing 6.1 Blinking LED program

```
import RPi.GPIO as GPIO
import time

# Variable for the GPIO pin number
LED_pin_red = 21

# Tell the Pi we are using the breakout board pin numbering
GPIO.setmode(GPIO.BCM)

# Set up the GPIO pin for output
GPIO.setup(LED_pin_red, GPIO.OUT)

# Loop to blink our led
while True:
    GPIO.output(LED_pin_red, GPIO.HIGH)
    print("On")
    time.sleep(1)
    GPIO.output(LED_pin_red, GPIO.LOW)
    print("Off")
    time.sleep(1)
```

Load the libraries you need to control the GPIO pins.

Create a variable for the GPIO pin number you're using to control the LED.

Set up GPIO pin 21 as an output.

Turn on GPIO pin 21 (GPIO.HIGH means turn on).

Turn off GPIO pin 21 (GPIO.LOW means turn off).

Save the program as blinkLED.py in your home folder. The program can't be run the same ways you've run programs before using IDLE.

Running the program

Select Run > Run Module (or press F5) from the IDLE text editor to run your program.With older versions of Raspbian, programs using GPIO pins must be run from the Raspbian command prompt as the superuser (or root)[5]. If you run the program at the Python Shell in IDLE, you'll get an error:

```
RuntimeError: No access to /dev/mem. Try running as root!
```

In this case, you use the `sudo` command to do this. To run the blinkLED.py program, open LXTerminal and enter the following command:

```
pi@raspberrypi ~ $ sudo python3 blinkLED.py
```

Behold the blinking LED! Try making the light blink faster by adjusting the value in the `sleep` function. Use a smaller number of seconds, such as 0.5 or 0.1.[6] To stop the program, press Ctrl-C.

> NOTE Stopping the program with Ctrl-C may result in the light being left on (depending on when you press it). Also, the next time you run the program, you may see a runtime error, but the program still works. We don't cover it here, but look online for the Python commands `try/except/finally` and the `GPIO.cleanup()` command. It's a fancy way to make sure all the GPIO pins are reset when you exit the program.

TROUBLESHOOTING

If the light isn't blinking, here are some things you can check:

- Are the on and off messages displaying on the screen? If so, it's probably not your code that has a problem. Check the circuit on the breadboard. Make sure the ribbon cable is connected properly, with the first wire

[5] In October 2015, the Raspberry Pi Foundation released Raspbian version "Jessie," which allows you to run programs using the GPIO pins directly from IDLE. With "Jessie" you don't need to open the command prompt. Simply press F5 or select Run > Run Module from the IDLE text editor menu to run your programs.

[6] Too small a number may cause the light to appear to stay on, but more dimly. This is because your eyes can only perceive blinking that is greater than about 1/25th of a second, or 0.04 of a second.

connected toward the edge of your Pi, away from the USB ports. Double-check that the jumper, LED, and resistor are connected to the correct holes.

- Could your LED be inserted the wrong way? Make sure the shorter leg is toward the negative or ground side. Try turning it around.
- Double-check the size of the resistor you used in the circuit. If the resistor is too large, the LED won't light up. A resistor that is between 100 and 300 ohms should work.
- Look through your Python program for errors. Check that you have set `LED_pin_red` equal to 21 and that you're setting it `HIGH` and then `LOW`.

blinkLED: how it works

Let's take a closer look at how the blinkLED.py code works.

LOADING LIBRARIES

The `import` commands load the libraries or toolboxes you want to use in your program:

```
import RPi.GPIO as GPIO
import time
```

These commands load the Python libraries for controlling the Pi's GPIO pins. They also load the `time` library so you can use the `sleep` function to control the rate of blinking.

Importing libraries with the as keyword

Notice the `as` keyword in `import RPi.GPIO as GPIO`. Why can't you just type `import RPi.GPIO`?

The `as` keyword tells Python to load the library to a certain name you specify. It's kind of like giving the whole library a nickname. In this case, it's so you can refer to `RPi.GPIO` as simply `GPIO`.

An example will make it clearer. Once you've imported the `RPi.GPIO` library as `GPIO`, you can type `GPIO.setmode(GPIO.BCM)`. Without it, you would have to type `RPi.GPIO.setmode(RPi.GPIO.BCM)`. You can see how using `as GPIO` saves you some typing!

Once the libraries are loaded, you can set up your GPIO pins.

SETTING UP A GPIO PIN FOR OUTPUT

To set up a GPIO pin, you first need to tell Python on your Pi that you'll be referring to pins by the standard breakout numbering scheme. These are the numbers printed on the breakout board. You use the `setmode` function:

```
GPIO.setmode(GPIO.BCM)
```

BCM stands for Broadcom—the maker of the computer chip that the Pi uses. Next you tell your Raspberry Pi that you'll be using `LED_pin_red` (GP21) for output, meaning you're planning to send some electricity out of it:

```
LED_pin_red = 21

GPIO.setup(LED_pin_red, GPIO.OUT)
```

`GPIO.OUT` prepares GP21 to send out +3.3 V of electricity.

LOOPING AND BLINKING

Finally, you create an infinite `while` loop and turn the LED on (set `GPIO.HIGH`) and off (set `GPIO.LOW`). You also add a delay using the `sleep` method found in Python's `time` library. Notice how the `sleep` function takes a parameter that is the number of seconds to sleep or pause. In this case, you use 1 second:

```
while True:
    GPIO.output(LED_pin_red, GPIO.HIGH)
    print("On")
    time.sleep(1)
    GPIO.output(LED_pin_red, GPIO.LOW)
    print("Off")
    time.sleep(1)
```

The `print` commands display messages to the screen. Although they aren't necessary to blink the LED, they can help debug your program. If you do use them, the screen could quickly fill with messages. Set a longer delay time to prevent this. If you see the messages on the screen but your LED isn't lighting up, then you probably have an error in your circuit and not in your program. Check your wiring, try turning around the LED, or try a different LED in case that one is defective.

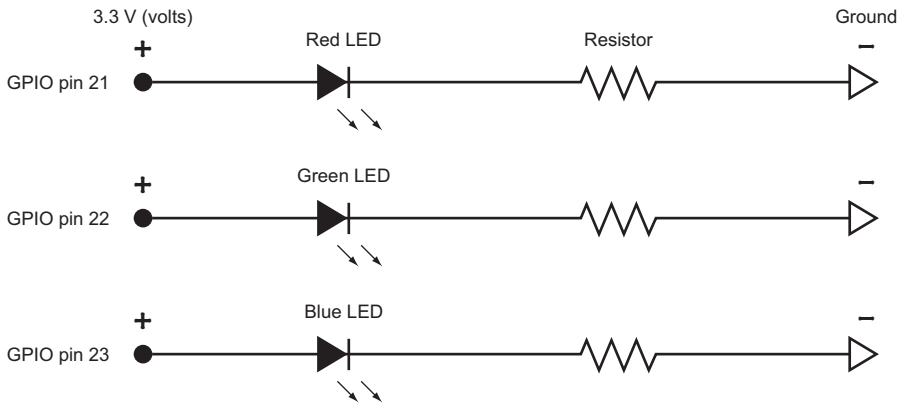
Adding more LEDs

One LED is fun, so three LEDs must be lots of fun. Let's try adding green and blue LEDs and modify the program to control them. Here are the parts you need:

- Raspberry Pi and circuit from before
- 1 green LED
- 1 blue LED
- 2 180 ohm resistors
- 2 jumper wires (male-to-male)

Building the circuit

You'll follow the same process as before to add the green and blue LEDs. Figure 6.14 shows what the circuit diagram looks like now, and figure 6.15 shows the circuit on a breadboard.



Note: Technically, the LEDs will be connected to a common ground on the Pi, so we could show these wires all connected together to one ground.

Figure 6.14 Circuit diagram for three LEDs: red, green, and blue. You'll use 180 ohm resistors like before. They will all be controlled by different GPIO pins. Red will use 21, green will use 22, and blue will be connected to pin 23. You could use any of the 26 different GPIO pins.

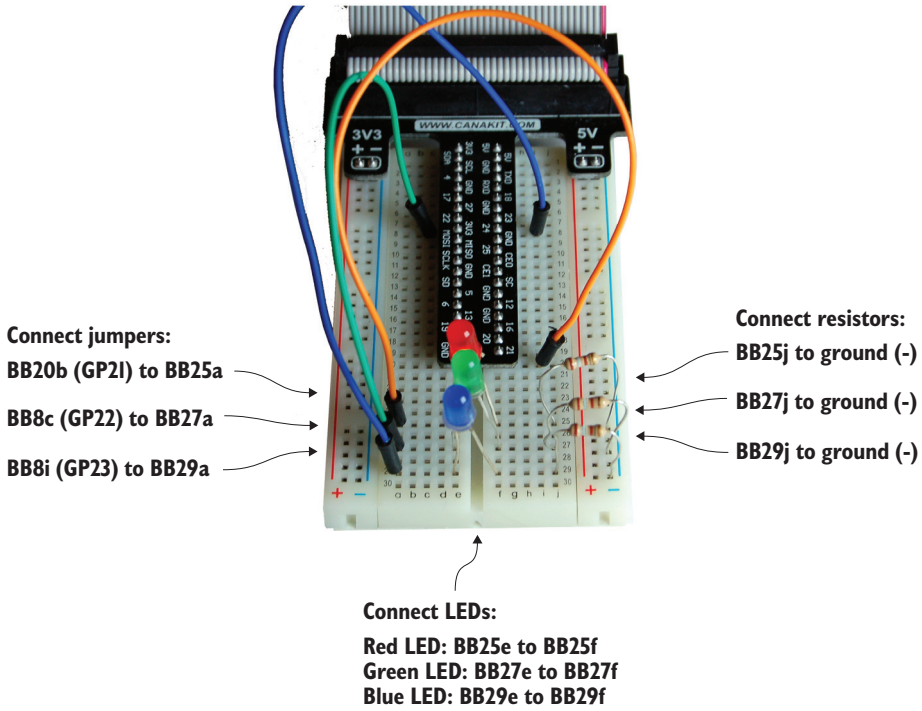


Figure 6.15 The three-LED circuit is built on the breadboard. Each LED and its corresponding resistor are placed in a row together. This example uses rows 25, 27, and 29.

To add the green LED, follow these steps:

1. GP22 is located on the *left side of the breakout board in row 8* on the breadboard. Connect it to *row 27*: insert one end of the jumper into BB8c and the other end into BB27a.
2. Connect the long leg of the green LED to BB27e and the shorter leg to BB27f. Bend the legs if needed.
3. Connect a 180 ohm resistor (brown, grey, and brown) from BB27j to the closest hole in the negative power bus.

Here are the steps to add the blue LED:

1. GPIO23 is located on the *right side of the breakout board in row 8* on the breadboard. Connect it to *row 29*: insert one end of the jumper into BB8i and the other end into BB29a.

2 Connect the long leg of the blue LED to BB29e and the shorter leg to BB29f. Bend the legs if needed.

3 Grab a 180 ohm resistor. You guessed it! It's color-coded brown, grey, and brown. Connect it from BB29j to the closest hole in the negative power bus.

Multiple LEDs: program it!

You need to make a few changes to the program to add more LEDs and get them all blinking at the same time. The following listing shows the updated code.

Listing 6.2 Three blinking LEDs

```
import RPi.GPIO as GPIO
import time

# Variable for the GPIO pin number
LED_pin_red = 21
LED_pin_green = 22
LED_pin_blue = 23

# Tell the Pi we are using the breakout board pin numbering
GPIO.setmode(GPIO.BCM)

# Set up the GPIO pins for output
GPIO.setup(LED_pin_red, GPIO.OUT)
GPIO.setup(LED_pin_green, GPIO.OUT)
GPIO.setup(LED_pin_blue, GPIO.OUT)

# Loop to blink our LEDs
while True:
    GPIO.output(LED_pin_red, GPIO.HIGH)
    GPIO.output(LED_pin_green, GPIO.HIGH)
    GPIO.output(LED_pin_blue, GPIO.HIGH)
    print("On")
    time.sleep(1)
    GPIO.output(LED_pin_red, GPIO.LOW)

    GPIO.output(LED_pin_green, GPIO.LOW)
    GPIO.output(LED_pin_blue, GPIO.LOW)
    print("Off")
    time.sleep(1)
```

Create variables for the GPIO pins you're using for the green and blue LEDs.

Set up GPIO pins 22 and 23 as outputs.

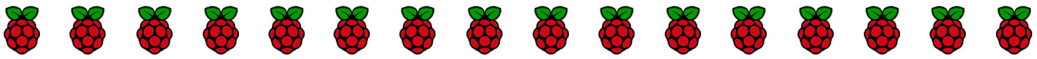
Turn on the GPIO pins.

Turn off the GPIO pins.

Save the code as blinkLED3.py, and try running it. Open LXTerminal, and enter the following command:

```
pi@raspberrypi ~ $ sudo python3 blinkLED3.py
```

Fantastic! You have your own light show going on!



Challenges

Try these challenges to practice controlling your Raspberry Pi's GPIO pins. Each one provides a unique problem to solve.

Wave pattern

Change the program to make each LED turn on, one at a time, until they're all on. Then, turn each LED off, one at a time. Hint: play with where you put the `time.sleep(1)` command. Can you make the LEDs light up and turn off in a wave pattern?

Simon Says

Write a function that blinks the LEDs and that can take five parameters representing a pattern of colorful blinks. Each parameter is a string representing a color: red, blue, or green. The function should blink the lights in the appropriate pattern. Here is a series of Simon Says patterns you should try to make your function produce:

Red, green, red, red, blue

Blue, green, blue, green, red

Green, blue, blue, red, green

Random blinking

Create a program that generates random durations for how long the lights stay on and off. The durations should be random floating-point numbers

between 0 and 3 seconds. Hint: you can use the `random` method to generate a random floating-point number between 0 and 1.0. Here is an example:

```
off_random_time = random.random() * 3
```

To scale this number so that it's between 0 and 3, you can multiply `off_random_time` by 3. If you get stuck on the challenge, check appendix C and the chapter source code for hints and solutions.

Summary

In this chapter, you learned the following things:

- A Pi is capable of interacting with the world around it. With a few extra parts, you can set it up for physical computing projects.
- A Pi can send out electrical signals! You can send output through the GPIO pins, and this can be used to light up LEDs or control many other electronic components (motors, buzzers, relays, and so on).
- Breadboards are like playgrounds for electronics. They make it easy to create circuits for your Pi because you can easily build and take apart circuits for use with the Pi.
- The `RPi.GPIO` library has built-in functions to set up and control output (voltage) to GPIO pins with Python.

Just imagine the possibilities of controlling pretty much any electrical device using your Raspberry Pi. Even better, imagine making the device work based on sensors (inputs) so you can create smart devices programmed by you!

Python—Absolute Basics

Maybe you have some familiarity with coding, but you have never done any Python. Many people in that situation find that they can pick up Python very quickly. Python seems to fit in people's brains with few surprises, and people who know some coding find themselves getting up to speed in Python quickly, and without the need for extremely detailed explanations.

If this sounds like you, *The Quick Python Book, 3rd edition* might be just what you're looking for. I've included twos chapters—one on "The absolute basics" of Python and one on lists, tuples and sets. These will give you a quick coder's eye view of some key features of the language.

If this approach resonates with you, you're likely to enjoy the rest of the book's coverage of Python, from dictionaries, strings, and functions to classes, libraries, data wrangling and beyond.

Chapter 4 from *The Quick Python Book, Third Edition* by Naomi Ceder

The absolute basics

This chapter covers

- Indenting and block structuring
- Differentiating comments
- Assigning variables
- Evaluating expressions
- Using common data types
- Getting user input
- Using correct Pythonic style

This chapter describes the absolute basics in Python: how to use assignments and expressions, how to type a number or a string, how to indicate comments in code, and so forth. It starts with a discussion of how Python block structures its code, which differs from every other major language.

4.1 Indentation and block structuring

Python differs from most other programming languages because it uses whitespace and indentation to determine block structure (that is, to determine what constitutes

the body of a loop, the `else` clause of a conditional, and so on). Most languages use braces of some sort to do this. Here is C code that calculates the factorial of 9, leaving the result in the variable `r`:

```
/* This is C code  */
int n, r;
n = 9;
r = 1;
while (n > 0) {
    r *= n;
    n--;
}
```

The braces delimit the body of the `while` loop, the code that is executed with each repetition of the loop. The code is usually indented more or less as shown, to make clear what's going on, but it could also be written like this:

```
/* And this is C code with arbitrary indentation */
    int n, r;
        n = 9;
        r = 1;
    while (n > 0) {
r *=  n;
n--;
}
```

The code still would execute correctly, even though it's rather difficult to read.

Here's the Python equivalent:

```
# This is Python code. (Yea!)
n = 9
r = 1
while n > 0:
    r = r * n
    n = n - 1
```

Python also supports C-style r *= n

Python also supports n -= 1

Python doesn't use braces to indicate code structure; instead, the indentation itself is used. The last two lines of the previous code are the body of the `while` loop because they come immediately after the `while` statement and are indented one level further than the `while` statement. If those lines weren't indented, they wouldn't be part of the body of the `while`.

Using indentation to structure code rather than braces may take some getting used to, but there are significant benefits:

- It's impossible to have missing or extra braces. You never need to hunt through your code for the brace near the bottom that matches the one a few lines from the top.
- The visual structure of the code reflects its real structure, which makes it easy to grasp the skeleton of code just by looking at it.

- Python coding styles are mostly uniform. In other words, you're unlikely to go crazy from dealing with someone's idea of aesthetically pleasing code. Everyone's code will look pretty much like yours.

You probably use consistent indentation in your code already, so this won't be a big step for you. If you're using IDLE, it automatically indents lines. You just need to backspace out of levels of indentation when desired. Most programming editors and IDEs—Emacs, VIM, and Eclipse, to name a few—provide this functionality as well. One thing that may trip you up once or twice until you get used to it is the fact that the Python interpreter returns an error message if you have a space (or spaces) preceding the commands you enter at a prompt.

4.2 Differentiating comments

For the most part, anything following a # symbol in a Python file is a comment and is disregarded by the language. The obvious exception is a # in a string, which is just a character of that string:

```
# Assign 5 to x
x = 5
x = 3                    # Now x is 3
x = "# This is not a comment"
```

You'll put comments in Python code frequently.

4.3 Variables and assignments

The most commonly used command in Python is assignment, which looks pretty close to what you might've used in other languages. Python code to create a variable called x and assign the value 5 to that variable is

```
x = 5
```

In Python, unlike in many other computer languages, neither a variable type declaration nor an end-of-line delimiter is necessary. The line is ended by the end of the line. Variables are created automatically when they're first assigned.

Variables in Python: buckets or labels?

The name *variable* is somewhat misleading in Python; *name* or *label* would be more accurate. However, it seems that pretty much everyone calls variables *variable*s at some time or another. Whatever you call them, you should know how they really work in Python.

A common, but inaccurate, explanation is that a variable is a container that stores a value, somewhat like a bucket. This would be reasonable for many programming languages (C, for example).

(continued)

However, in Python variables aren't buckets. Instead, they're labels or tags that refer to objects in the Python interpreter's namespace. Any number of labels (or variables) can refer to the same object, and when that object changes, the value referred to by *all* of those variables also changes.

To see what this means, look at the following simple code:

```
>>> a = [1, 2, 3]
>>> b = a
>>> c = b
>>> b[1] = 5
>>> print(a, b, c)
[1, 5, 3] [1, 5, 3] [1, 5, 3]
```

If you're thinking of variables as containers, this result makes no sense. How could changing the contents of one container simultaneously change the other two? However, if variables are just labels referring to objects, it makes sense that changing the object that all three labels refer to would be reflected everywhere.

If the variables are referring to constants or immutable values, this distinction isn't quite as clear:

```
>>> a = 1
>>> b = a
>>> c = b
>>> b = 5
>>> print(a, b, c)
1 5 1
```

Because the objects they refer to can't change, the behavior of the variables in this case is consistent with either explanation. In fact, in this case, after the third line a, b, and c all refer to the same unchangeable integer object with the value 1. The next line, b = 5, makes b refer to the integer object 5 but doesn't change the references of a or c.

Python variables can be set to any object, whereas in C and many other languages, variables can store only the type of value they're declared as. The following is perfectly legal Python code:

```
>>> x = "Hello"
>>> print(x)
Hello
>>> x = 5
>>> print(x)
5
```

x starts out referring to the string object `"Hello"` and then refers to the integer object `5`. Of course, this feature can be abused, because arbitrarily assigning the same variable name to refer successively to different data types can make code confusing to understand.

A new assignment overrides any previous assignments. The `del` statement deletes the variable. Trying to print the variable's contents after deleting it results in an error, as though the variable had never been created in the first place:

```
>>> x = 5
>>> print(x)
5
>>> del x
>>> print(x)
Traceback (most recent call last):
 File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
>>>
```

Here, you have your first look at a *traceback*, which is printed when an error, called an *exception*, has been detected. The last line tells you what exception was detected, which in this case is a `NameError` exception on `x`. After its deletion, `x` is no longer a valid variable name. In this example, the trace returns only `line 1, in <module>` because only the single line has been sent in the interactive mode. In general, the full dynamic call structure of the existing function at the time of the error's occurrence is returned. If you're using IDLE, you obtain the same information with some small differences. The code may look something like this:

```
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    print(x)
NameError: name 'x' is not defined
```

Chapter 14 describes this mechanism in more detail. A full list of the possible exceptions and what causes them is in the Python standard library documentation. Use the index to find any specific exception (such as `NameError`) you receive.

Variable names are case-sensitive and can include any alphanumeric character as well as underscores but must start with a letter or underscore. See section 4.10 for more guidance on the Pythonic style for creating variable names.

4.4 Expressions

Python supports arithmetic and similar expressions; these expressions will be familiar to most readers. The following code calculates the average of 3 and 5, leaving the result in the variable `z`:

```
x = 3
y = 5
z = (x + y) / 2
```

Note that arithmetic operators involving only integers do *not* always return an integer. Even though all the values are integers, division (starting with Python 3) returns a floating-point number, so the fractional part isn't truncated. If you want traditional integer division returning a truncated integer, you can use `//` instead.

Standard rules of arithmetic precedence apply. If you'd left out the parentheses in the last line, the code would've been calculated as `x + (y / 2)`.

Expressions don't have to involve just numerical values; strings, Boolean values, and many other types of objects can be used in expressions in various ways. I discuss these objects in more detail as they're used.

TRY THIS: VARIABLES AND EXPRESSIONS In the Python shell, create some variables. What happens when you try to put spaces, dashes, or other nonalphanumeric characters in the variable name? Play around with a few complex expressions, such as `x = 2 + 4 * 5 - 6 / 3`. Use parentheses to group the numbers in different ways and see how the result changes compared with the original ungrouped expression.

4.5 Strings

You've already seen that Python, like most other programming languages, indicates strings through the use of double quotes. This line leaves the string `"Hello, World"` in the variable `x`:

```
x = "Hello, World"
```

Backslashes can be used to escape characters, to give them special meanings. `\n` means the newline character, `\t` means the tab character, `\\` means a single normal backslash character, and `\"` is a plain double-quote character. It doesn't end the string:

```
x = "\tThis string starts with a \"tab\"."
x = "This string contains a single backslash(\\)."
```

You can use single quotes instead of double quotes. The following two lines do the same thing:

```
x = "Hello, World"
x = 'Hello, World'
```

The only difference is that you don't need to backslash `"` characters in single-quoted strings or `'` characters in double-quoted strings:

```
x = "Don't need a backslash"
x = 'Can\'t get by without a backslash'
x = "Backslash your \" character!"
x = 'You can leave the " alone'
```

You can't split a normal string across lines. This code won't work:

```
# This Python code will cause an ERROR -- you can't split the string
across two lines.
x = "This is a misguided attempt to
put a newline into a string without using backslash-n"
```

But Python offers triple-quoted strings, which let you do this and include single and double quotes without backslashes:

```
x = """Starting and ending a string with triple " characters
permits embedded newlines, and the use of " and ' without
backslashes"""
```

Now x is the entire sentence between the """ delimiters. (You can use triple single quotes—'''—instead of triple double quotes to do the same thing.)

Python offers enough string-related functionality that chapter 6 is devoted to the topic.

4.6 Numbers

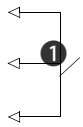
Because you're probably familiar with standard numeric operations from other languages, this book doesn't contain a separate chapter describing Python's numeric abilities. This section describes the unique features of Python numbers, and the Python documentation lists the available functions.

Python offers four kinds of numbers: *integers*, *floats*, *complex numbers*, and *Booleans*. An integer constant is written as an integer—0, –11, +33, 123456—and has unlimited range, restricted only by the resources of your machine. A float can be written with a decimal point or in scientific notation: 3.14, –2E-8, 2.718281828. The precision of these values is governed by the underlying machine but is typically equal to double (64-bit) types in C. Complex numbers are probably of limited interest and are discussed separately later in the section. Booleans are either `True` or `False` and behave identically to 1 and 0 except for their string representations.

Arithmetic is much like it is in C. Operations involving two integers produce an integer, except for division (/), which results in a float. If the // division symbol is used, the result is an integer, with truncation. Operations involving a float always produce a float. Here are a few examples:

```
>>> 5 + 2 - 3 * 2
1
>>> 5 / 2          # floating-point result with normal division
2.5
>>> 5 / 2.0        # also a floating-point result
2.5
>>> 5 // 2         # integer result with truncation when divided using '//'
2
>>> 30000000000    # This would be too large to be an int in many languages
30000000000
>>> 30000000000 * 3
90000000000
>>> 30000000000 * 3.0
90000000000.0
>>> 2.0e-8         # Scientific notation gives back a float
2e-08
>>> 3000000 * 3000000
```

```
9000000000000
>>> int(200.2)
200
>>> int(2e2)
200
>>> float(200)
200.0
```



These are explicit conversions between types ❶. `int` truncates float values.

Numbers in Python have two advantages over C or Java: Integers can be arbitrarily large, and the division of two integers results in a float.

4.6.1 Built-in numeric functions

Python provides the following number-related functions as part of its core:

```
abs, divmod, float, hex, int, max, min, oct,
pow, round
```

See the documentation for details.

4.6.2 Advanced numeric functions

More advanced numeric functions such as the trig and hyperbolic trig functions, as well as a few useful constants, aren't built into Python but are provided in a standard module called `math`. I explain modules in detail later. For now, it's sufficient to know that you must make the math functions in this section available by starting your Python program or interactive session with the statement

```
from math import *
```

The `math` module provides the following functions and constants:

```
acos, asin, atan, atan2, ceil, cos, cosh, e, exp, fabs, floor, fmod,
frexp, hypot, ldexp, log, log10, mod, pi, pow, sin, sinh, sqrt, tan,
tanh
```

See the documentation for details.

4.6.3 Numeric computation

The core Python installation isn't well suited to intensive numeric computation because of speed constraints. But the powerful Python extension `NumPy` provides highly efficient implementations of many advanced numeric operations. The emphasis is on array operations, including multidimensional matrices and more advanced functions such as the Fast Fourier Transform. You should be able to find `NumPy` (or links to it) at www.scipy.org.

4.6.4 Complex numbers

Complex numbers are created automatically whenever an expression of the form `nj` is encountered, with `n` having the same form as a Python integer or float. `j` is, of course,

standard notation for the imaginary number equal to the square root of –1, for example:

```
>>> (3+2j)
(3+2j)
```

Note that Python expresses the resulting complex number in parentheses as a way of indicating that what's printed to the screen represents the value of a single object:

```
>>> 3 + 2j - (4+4j)
(-1-2j)
>>> (1+2j) * (3+4j)
(-5+10j)
>>> 1j * 1j
(-1+0j)
```

Calculating `j * j` gives the expected answer of –1, but the result remains a Python complex-number object. Complex numbers are never converted automatically to equivalent real or integer objects. But you can easily access their real and imaginary parts with `real` and `imag`:

```
>>> z = (3+5j)
>>> z.real
3.0
>>> z.imag
5.0
```

Note that real and imaginary parts of a complex number are always returned as floating-point numbers.

4.6.5 Advanced complex-number functions

The functions in the `math` module don't apply to complex numbers; the rationale is that most users want the square root of –1 to generate an error, not an answer! Instead, similar functions, which can operate on complex numbers, are provided in the `cmath` module:

```
acos, acosh, asin, asinh, atan, atanh, cos, cosh, e, exp, log, log10,
pi, sin, sinh, sqrt, tan, tanh.
```

To make clear in the code that these functions are special-purpose complex-number functions and to avoid name conflicts with the more normal equivalents, it's best to import the `cmath` module by saying

```
import cmath
```

and then to explicitly refer to the `cmath` package when using the function:

```
>>> import cmath
>>> cmath.sqrt(-1)
1j
```

Minimizing from <module> import *

This is a good example of why it's best to minimize the use of the `from <module> import *` form of the `import` statement. If you used it to import first the `math` module and then the `cmath` module, the commonly named functions in `cmath` would override those of `math`. It's also more work for someone reading your code to figure out the source of the specific functions you use. Some modules are explicitly designed to use this form of import.

See chapter 10 for more details on how to use modules and module names.

The important thing to keep in mind is that by importing the `cmath` module, you can do almost anything you can do with other numbers.

TRY THIS: MANIPULATING STRINGS AND NUMBERS In the Python shell, create some string and number variables (integers, floats, *and* complex numbers). Experiment a bit with what happens when you do operations with them, including across types. Can you multiply a string by an integer, for example, or can you multiply it by a float or complex number? Also load the `math` module and try a few of the functions; then load the `cmath` module and do the same. What happens if you try to use one of those functions on an integer or float after loading the `cmath` module? How might you get the `math` module functions back?

4.7 The None value

In addition to standard types such as strings and numbers, Python has a special basic data type that defines a single special data object called `None`. As the name suggests, `None` is used to represent an empty value. It appears in various guises throughout Python. For example, a procedure in Python is just a function that doesn't explicitly return a value, which means that by default, it returns `None`.

`None` is often useful in day-to-day Python programming as a placeholder to indicate a point in a data structure where meaningful data will eventually be found, even though that data hasn't yet been calculated. You can easily test for the presence of `None` because there's only one instance of `None` in the entire Python system (all references to `None` point to the same object), and `None` is equivalent only to itself.

4.8 Getting input from the user

You can also use the `input()` function to get input from the user. Use the prompt string you want to display to the user as `input`'s parameter:

```
>>> name = input("Name? ")
Name? Jane
>>> print(name)
Jane
>>> age = int(input("Age? "))
Age? 28
```

Converts input from string to int

```
>>> print(age)
28
>>>
```

This is a fairly simple way to get user input. The one catch is that the input comes in as a string, so if you want to use it as a number, you have to use the `int()` or `float()` function to convert it.

> **TRY THIS: GETTING INPUT** Experiment with the `input()` function to get string and integer input. Using code similar to the previous code, what is the effect of not using `int()` around the call to `input()` for integer input? Can you modify that code to accept a float—say, 28.5? What happens if you deliberately enter the wrong type of value? Examples include a float in which an integer is expected and a string in which a number is expected—and vice versa.

4.9 Built-in operators

Python provides various built-in operators, from the standard (`+`, `*`, and so on) to the more esoteric, such as operators for performing bit shifting, bitwise logical functions, and so forth. Most of these operators are no more unique to Python than to any other language; hence, I won't explain them in the main text. You can find a complete list of the Python built-in operators in the documentation.

4.10 Basic Python style

Python has relatively few limitations on coding style with the obvious exception of the requirement to use indentation to organize code into blocks. Even in that case, the amount of indentation and type of indentation (tabs versus spaces) isn't mandated. However, there are preferred stylistic conventions for Python, which are contained in Python Enhancement Proposal (PEP) 8, which is summarized in appendix A and available online at www.python.org/dev/peps/pep-0008/. A selection of Pythonic conventions is provided in table 4.1, but to fully absorb Pythonic style, periodically reread PEP 8.

Table 4.1 Pythonic coding conventions

Situation	Suggestion	Example
Module/package names	Short, all lowercase, underscores only if needed	`imp, sys`
Function names	All lowercase, underscores_for_readablitiy	`foo(), my_func()`
Variable names	All lowercase, underscores_for_readablitiy	`my_var`
Class names	CapitalizeEachWord	`MyClass`
Constant names	ALL_CAPS_WITH_UNDERSCORES	`PI, TAX_RATE`

Table 4.1 Pythonic coding conventions *(continued)*

Situation	Suggestion	Example
Indentation	Four spaces per level, no tabs	
Comparisons	Don't compare explicitly to True or False	`if my_var:` `if not my_var:`

I strongly urge you to follow the conventions of PEP 8. They're wisely chosen and time-tested, and they'll make your code easier for you and other Python programmers to understand.

QUICK CHECK: PYTHONIC STYLE Which of the following variable and function names do you think are not good Pythonic style? Why?

```
bar(), varName, VERYLONGVARNAME, foobar, longvarname, foo_bar(),
really_very_long_var_name
```

Summary

- The basic syntax summarized above is enough to start writing Python code.
- Python syntax is predictable and consistent.
- Because the syntax offers few surprises, many programmers can get started writing code surprisingly quickly.

Chapter 5 from *The Quick Python Book, Third Edition* by Naomi Ceder

Lists, tuples, and sets

This chapter covers

- Manipulating lists and list indices
- Modifying lists
- Sorting
- Using common list operations
- Handling nested lists and deep copies
- Using tuples
- Creating and using sets

In this chapter, I discuss the two major Python sequence types: lists and tuples. At first, lists may remind you of arrays in many other languages, but don't be fooled: lists are a good deal more flexible and powerful than plain arrays.

Tuples are like lists that can't be modified; you can think of them as a restricted type of list or as a basic record type. I discuss the need for such a restricted data type later in the chapter. This chapter also discusses a newer Python collection type: sets. Sets are useful when an object's membership in the collection, as opposed to its position, is important

Most of the chapter is devoted to lists, because if you understand lists, you pretty much understand tuples. The last part of the chapter discusses the differences between lists and tuples in both functional and design terms.

5.1 Lists are like arrays

A list in Python is much the same thing as an array in Java or C or any other language; it's an ordered collection of objects. You create a list by enclosing a comma-separated list of elements in square brackets, like so:

```
# This assigns a three-element list to x
x = [1, 2, 3]
```

Note that you don't have to worry about declaring the list or fixing its size ahead of time. This line creates the list as well as assigns it, and a list automatically grows or shrinks as needed.

> **Arrays in Python**
>
> A typed `array` module available in Python provides arrays based on C data types. Information on its use can be found in the *Python Library Reference*. I suggest that you look into it only if you really need the performance improvement. If a situation calls for numerical computations, you should consider using `NumPy`, mentioned in chapter 4 and available at www.scipy.org.

Unlike lists in many other languages, Python lists can contain different types of elements; a list element can be any Python object. Here's a list that contains a variety of elements:

```
# First element is a number, second is a string, third is another list.
x = [2, "two", [1, 2, 3]]
```

Probably the most basic built-in list function is the `len` function, which returns the number of elements in a list:

```
>>> x = [2, "two", [1, 2, 3]]
>>> len(x)
3
```

Note that the `len` function doesn't count the items in the inner, nested list.

QUICK CHECK: LEN() What would `len()` return for each of the following: `[0]; []; [[1, 3, [4, 5], 6], 7]`?

5.2 List indices

Understanding how list indices work will make Python much more useful to you. Please read the whole section!

Elements can be extracted from a Python list by using a notation like C's array indexing. Like C and many other languages, Python starts counting from 0; asking for element 0 returns the first element of the list, asking for element 1 returns the second element, and so forth. Here are a few examples:

```
>>> x = ["first", "second", "third", "fourth"]
>>> x[0]
'first'
>>> x[2]
'third'
```

But Python indexing is more flexible than C indexing. If indices are negative numbers, they indicate positions counting from the end of the list, with –1 being the last position in the list, –2 being the second-to-last position, and so forth. Continuing with the same list x, you can do the following:

```
>>> a = x[-1]
>>> a
'fourth'
>>> x[-2]
'third'
```

For operations involving a single list index, it's generally satisfactory to think of the index as pointing at a particular element in the list. For more advanced operations, it's more correct to think of list indices as indicating positions *between* elements. In the list `["first", "second", "third", "fourth"]`, you can think of the indices as pointing like this:

x =[		"first",		"second",		"third",		"fourth"		]
Positive indices	0		1		2		3			
Negative indices	–4		–3		–2		–1			

This is irrelevant when you're extracting a single element, but Python can extract or assign to an entire sublist at once—an operation known as *slicing*. Instead of entering `list[index]` to extract the item just after `index`, enter `list[index1:index2]` to extract all items including `index1` and up to (but not including) `index2` into a new list. Here are some examples:

```
>>> x = ["first", "second", "third", "fourth"]
>>> x[1:-1]
['second', 'third']
>>> x[0:3]
['first', 'second', 'third']
>>> x[-2:-1]
['third']
```

It may seem reasonable that if the second index indicates a position in the list *before* the first index, this code would return the elements between those indices in reverse order, but this isn't what happens. Instead, this code returns an empty list:

```
>>> x[-1:2]
[]
```

When slicing a list, it's also possible to leave out `index1` or `index2`. Leaving out `index1` means "Go from the beginning of the list," and leaving out `index2` means "Go to the end of the list":

```
>>> x[:3]
['first', 'second', 'third']
>>> x[2:]
['third', 'fourth']
```

Omitting both indices makes a new list that goes from the beginning to the end of the original list—that is, copies the list. This technique is useful when you want to make a copy that you can modify without affecting the original list:

```
>>> y = x[:]
>>> y[0] = '1 st'
>>> y
['1 st', 'second', 'third', 'fourth']
>>> x
['first', 'second', 'third', 'fourth']
```

TRY THIS: LIST SLICES AND INDEXES Using what you know about the `len()` function and list slices, how would you combine the two to get the second half of a list when you don't know what size it is? Experiment in the Python shell to confirm that your solution works.

5.3 Modifying lists

You can use list index notation to modify a list as well as to extract an element from it. Put the index on the left side of the assignment operator:

```
>>> x = [1, 2, 3, 4]
>>> x[1] = "two"
>>> x
[1, 'two', 3, 4]
```

Slice notation can be used here too. Saying something like `lista[index1:index2] = listb` causes all elements of `lista` between `index1` and `index2` to be replaced by the elements in `listb`. `listb` can have more or fewer elements than are removed from `lista`, in which case the length of `lista` is altered. You can use slice assignment to do several things, as shown here:

```
>>> x = [1, 2, 3, 4]
>>> x[len(x):] = [5, 6, 7]
>>> x
[1, 2, 3, 4, 5, 6, 7]
```

Appends list to end of list

```
>>> x[:0] = [-1, 0]
>>> x
[-1, 0, 1, 2, 3, 4, 5, 6, 7]
>>> x[1:-1] = []
>>> x
[-1, 7]
```

Appends list to front of list

Removes elements from list

Appending a single element to a list is such a common operation that there's a special `append` method for it:

```
>>> x = [1, 2, 3]
>>> x.append("four")
>>> x
[1, 2, 3, 'four']
```

One problem can occur if you try to append one list to another. The list gets appended as a single element of the main list:

```
>>> x = [1, 2, 3, 4]
>>> y = [5, 6, 7]
>>> x.append(y)
>>> x
[1, 2, 3, 4, [5, 6, 7]]
```

The `extend` method is like the `append` method except that it allows you to add one list to another:

```
>>> x = [1, 2, 3, 4]
>>> y = [5, 6, 7]
>>> x.extend(y)
>>> x
[1, 2, 3, 4, 5, 6, 7]
```

There's also a special `insert` method to insert new list elements between two existing elements or at the front of the list. `insert` is used as a method of lists and takes two additional arguments. The first additional argument is the index position in the list where the new element should be inserted, and the second is the new element itself:

```
>>> x = [1, 2, 3]
>>> x.insert(2, "hello")
>>> print(x)
[1, 2, 'hello', 3]
>>> x.insert(0, "start")
>>> print(x)
['start', 1, 2, 'hello', 3]
```

`insert` understands list indices as discussed in section 5.2, but for most uses, it's easiest to think of `list.insert(n, elem)` as meaning `insert elem` just before the *n*th element of list. `insert` is just a convenience method. Anything that can be done with `insert` can also be done with slice assignment. That is, `list.insert(n, elem)` is the

same thing as `list[n:n] = [elem]` when `n` is nonnegative. Using `insert` makes for somewhat more readable code, and `insert` even handles negative indices:

```
>>> x = [1, 2, 3]
>>> x.insert(-1, "hello")
>>> print(x)
[1, 2, 'hello', 3]
```

The `del` statement is the preferred method of deleting list items or slices. It doesn't do anything that can't be done with slice assignment, but it's usually easier to remember and easier to read:

```
>>> x = ['a', 2, 'c', 7, 9, 11]
>>> del x[1]
>>> x
['a', 'c', 7, 9, 11]
>>> del x[:2]
>>> x
[7, 9, 11]
```

In general, `del list[n]` does the same thing as `list[n:n+1] = []`, whereas `del list[m:n]` does the same thing as `list[m:n] = []`.

The `remove` method isn't the converse of `insert`. Whereas `insert` inserts an element at a specified location, `remove` looks for the first instance of a given value in a list and removes that value from the list:

```
>>> x = [1, 2, 3, 4, 3, 5]
>>> x.remove(3)
>>> x
[1, 2, 4, 3, 5]
>>> x.remove(3)
>>> x
[1, 2, 4, 5]
>>> x.remove(3)
Traceback (innermost last):
 File "<stdin>", line 1, in ?
ValueError: list.remove(x): x not in list
```

If `remove` can't find anything to remove, it raises an error. You can catch this error by using the exception-handling abilities of Python, or you can avoid the problem by using `in` to check for the presence of something in a list before attempting to remove it.

The `reverse` method is a more specialized list modification method. It efficiently reverses a list in place:

```
>>> x = [1, 3, 5, 6, 7]
>>> x.reverse()
>>> x
[7, 6, 5, 3, 1]
```

TRY THIS: MODIFYING LISTS Suppose that you have a list 10 items long. How might you move the last three items from the end of the list to the beginning, keeping them in the same order?

5.4 Sorting lists

Lists can be sorted by using the built-in Python `sort` method:

```
>>> x = [3, 8, 4, 0, 2, 1]
>>> x.sort()
>>> x
[0, 1, 2, 3, 4, 8]
```

This method does an in-place sort—that is, changes the list being sorted. To sort a list without changing the original list, you have two options. You can use the `sorted()` built-in function, discussed in section 5.4.2, or you can make a copy of the list and sort the copy:

```
>>> x = [2, 4, 1, 3]
>>> y = x[:]
>>> y.sort()
>>> y
[1, 2, 3, 4]
>>> x
[2, 4, 1, 3]
```

Sorting works with strings, too:

```
>>> x = ["Life", "Is", "Enchanting"]
>>> x.sort()
>>> x
['Enchanting', 'Is', 'Life']
```

The `sort` method can sort just about anything because Python can compare just about anything. But there's one caveat in sorting: The default key method used by `sort` requires all items in the list to be of comparable types. That means that using the `sort` method on a list containing both numbers and strings raises an exception:

```
>>> x = [1, 2, 'hello', 3]
>>> x.sort()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: '<' not supported between instances of 'str' and 'int'
```

Conversely, you can sort a list of lists:

```
>>> x = [[3, 5], [2, 9], [2, 3], [4, 1], [3, 2]]
>>> x.sort()
>>> x
[[2, 3], [2, 9], [3, 2], [3, 5], [4, 1]]
```

According to the built-in Python rules for comparing complex objects, the sublists are sorted first by ascending first element and then by ascending second element.

`sort` is even more flexible; it has an optional `reverse` parameter that causes the sort to be in reverse order when `reverse=True`, and it's possible to use your own key function to determine how elements of a list are sorted.

5.4.1 Custom sorting

To use custom sorting, you need to be able to define functions—something I haven't talked about yet. In this section, I also discuss the fact that `len(string)` returns the number of characters in a string. String operations are discussed more fully in chapter 6.

By default, `sort` uses built-in Python comparison functions to determine ordering, which is satisfactory for most purposes. At times, though, you want to sort a list in a way that doesn't correspond to this default ordering. Suppose that you want to sort a list of words by the number of characters in each word, as opposed to the lexicographic sort that Python normally carries out.

To do this, write a function that returns the value, or key, that you want to sort on, and use it with the `sort` method. That function in the context of `sort` is a function that takes one argument and returns the key or value that the `sort` function is to use.

For number-of-characters ordering, a suitable key function could be

```
def compare_num_of_chars(string1):
    return len(string1)
```

This key function is trivial. It passes the length of each string back to the `sort` method, rather than the strings themselves.

After you define the key function, using it is a matter of passing it to the `sort` method by using the `key` keyword. Because functions are Python objects, they can be passed around like any other Python objects. Here's a small program that illustrates the difference between a default sort and your custom sort:

```
>>> def compare_num_of_chars(string1):
...     return len(string1)
>>> word_list = ['Python', 'is', 'better', 'than', 'C']
>>> word_list.sort()
>>> print(word_list)
['C', 'Python', 'better', 'is', 'than']
>>> word_list = ['Python', 'is', 'better', 'than', 'C']
>>> word_list.sort(key=compare_num_of_chars)
>>> print(word_list)
['C', 'is', 'than', 'Python', 'better']
```

The first list is in lexicographical order (with uppercase coming before lowercase), and the second list is ordered by ascending number of characters.

Custom sorting is very useful, but if performance is critical, it may be slower than the default. Usually, this effect is minimal, but if the key function is particularly complex, the effect may be more than desired, especially for sorts involving hundreds of thousands or millions of elements.

One particular place to avoid custom sorts is where you want to sort a list in descending, rather than ascending, order. In this case, use the `sort` method's `reverse` parameter set to `True`. If for some reason you don't want to do that, it's still better to sort the list normally and then use the `reverse` method to invert the order of the

resulting list. These two operations together—the standard sort and the reverse—will still be much faster than a custom sort.

5.4.2 The sorted() function

Lists have a built-in method to sort themselves, but other iterables in Python, such as the keys of a dictionary, don't have a `sort` method. Python also has the built-in function `sorted()`, which returns a sorted list from any iterable. `sorted()` uses the same `key` and `reverse` parameters as the sort method:

```
>>> x = (4, 3, 1, 2)
>>> y = sorted(x)
>>> y
[1, 2, 3, 4]
>>> z = sorted(x, reverse=True)
>>> z
[4, 3, 2, 1]
```

TRY THIS: SORTING LISTS Suppose that you have a list in which each element is in turn a list: `[[1, 2, 3], [2, 1, 3], [4, 0, 1]]`. If you wanted to sort this list by the second element in each list so that the result would be `[[4, 0, 1], [2, 1, 3], [1, 2, 3]]`, what function would you write to pass as the `key` value to the `sort()` method?

5.5 Other common list operations

Several other list methods are frequently useful, but they don't fall into any specific category.

5.5.1 List membership with the in operator

It's easy to test whether a value is in a list by using the `in` operator, which returns a Boolean value. You can also use the converse, the `not in` operator:

```
>>> 3 in [1, 3, 4, 5]
True
>>> 3 not in [1, 3, 4, 5]
False
>>> 3 in ["one", "two", "three"]
False
>>> 3 not in ["one", "two", "three"]
True
```

5.5.2 List concatenation with the + operator

To create a list by concatenating two existing lists, use the `+` (list concatenation) operator, which leaves the argument lists unchanged:

```
>>> z = [1, 2, 3] + [4, 5]
>>> z
[1, 2, 3, 4, 5]
```

5.5.3 List initialization with the * operator

Use the * operator to produce a list of a given size, which is initialized to a given value. This operation is a common one for working with large lists whose size is known ahead of time. Although you can use `append` to add elements and automatically expand the list as needed, you obtain greater efficiency by using * to correctly size the list at the start of the program. A list that doesn't change in size doesn't incur any memory reallocation overhead:

```
>>> z = [None] * 4
>>> z
[None, None, None, None]
```

When used with lists in this manner, * (which in this context is called the *list multiplication operator*) replicates the given list the indicated number of times and joins all the copies to form a new list. This is the standard Python method for defining a list of a given size ahead of time. A list containing a single instance of `None` is commonly used in list multiplication, but the list can be anything:

```
>>> z = [3, 1] * 2
>>> z
[3, 1, 3, 1]
```

5.5.4 List minimum or maximum with min and max

You can use `min` and `max` to find the smallest and largest elements in a list. You'll probably use `min` and `max` mostly with numerical lists, but you can use them with lists containing any type of element. Trying to find the maximum or minimum object in a set of objects of different types causes an error if comparing those types doesn't make sense:

```
>>> min([3, 7, 0, -2, 11])
-2
>>> max([4, "Hello", [1, 2]])
Traceback (most recent call last):
  File "<pyshell#58>", line 1, in <module>
    max([4, "Hello",[1, 2]])
TypeError: '>' not supported between instances of 'str' and 'int'
```

5.5.5 List search with index

If you want to find where in a list a value can be found (rather than wanting to know only whether the value is in the list), use the `index` method. This method searches through a list looking for a list element equivalent to a given value and returns the position of that list element:

```
>>> x = [1, 3, "five", 7, -2]
>>> x.index(7)
3
>>> x.index(5)
Traceback (innermost last):
 File "<stdin>", line 1, in ?
ValueError: 5 is not in list
```

Attempting to find the position of an element that doesn't exist in the list raises an error, as shown here. This error can be handled in the same manner as the analogous error that can occur with the `remove` method (that is, by testing the list with `in` before using `index`).

5.5.6 List matches with count

`count` also searches through a list, looking for a given value, but it returns the number of times that the value is found in the list rather than positional information:

```
>>> x = [1, 2, 2, 3, 5, 2, 5]
>>> x.count(2)
3
>>> x.count(5)
2
>>> x.count(4)
0
```

5.5.7 Summary of list operations

You can see that lists are very powerful data structures, with possibilities that go far beyond those of plain old arrays. List operations are so important in Python programming that it's worth laying them out for easy reference, as shown in table 5.1.

Table 5.1 List operations

List operation	Explanation	Example
`[]`	Creates an empty list	`x = []`
`len`	Returns the length of a list	`len(x)`
`append`	Adds a single element to the end of a list	`x.append('y')`
`extend`	Adds another list to the end of the list	`x.extend(['a', 'b'])`
`insert`	Inserts a new element at a given position in the list	`x.insert(0, 'y')`
`del`	Removes a list element or slice	`del(x[0])`
`remove`	Searches for and removes a given value from a list	`x.remove('y')`
`reverse`	Reverses a list in place	`x.reverse()`
`sort`	Sorts a list in place	`x.sort()`
`+`	Adds two lists together	`x1 + x2`
`*`	Replicates a list	`x = ['y'] * 3`
`min`	Returns the smallest element in a list	`min(x)`
`max`	Returns the largest element in a list	`max(x)`
`index`	Returns the position of a value in a list	`x.index['y']`
`count`	Counts the number of times a value occurs in a list	`x.count('y')`

Table 5.1 List operations *(continued)*

List operation	Explanation	Example
`sum`	Sums the items (if they can be summed)	`sum(x)`
`in`	Returns whether an item is in a list	`'y' in x`

Being familiar with these list operations will make your life as a Python coder much easier.

QUICK CHECK: LIST OPERATIONS What would be the result of `len([[1,2]] * 3)`?

What are two differences between using the `in` operator and a list's `index()` method?

Which of the following will raise an exception?: `min(["a", "b", "c"])`; `max([1, 2, "three"])`; `[1, 2, 3].count("one")`

TRY THIS: LIST OPERATIONS If you have a list `x`, write the code to safely remove an item if—and only if—that value is in the list.

Modify that code to remove the element only if the item occurs in the list more than once.

5.6 Nested lists and deep copies

This section covers another advanced topic that you may want to skip if you're just learning the language.

Lists can be nested. One application of nesting is to represent two-dimensional matrices. The members of these matrices can be referred to by using two-dimensional indices. Indices for these matrices work as follows:

```
>>> m = [[0, 1, 2], [10, 11, 12], [20, 21, 22]]
>>> m[0]
[0, 1, 2]
>>> m[0][1]
1
>>> m[2]
[20, 21, 22]
>>> m[2][2]
22
```

This mechanism scales to higher dimensions in the manner you'd expect.

Most of the time, this is all you need to concern yourself with. But you may run into an issue with nested lists; specifically the way that variables refer to objects and how some objects (such as lists) can be modified (are mutable). An example is the best way to illustrate:

```
>>> nested = [0]
>>> original = [nested, 1]
>>> original
[[0], 1]
```

Figure 5.1 shows what this example looks like.

Now the value in the nested list can be changed by using either the nested or the original variables:

```
>>> nested[0] = 'zero'
>>> original
[['zero'], 1]
>>> original[0][0] = 0
>>> nested
[0]
>>> original
[[0], 1]
```

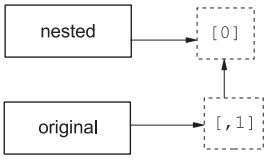


Figure 5.1 A list with its first item referring to a nested list

But if `nested` is set to another list, the connection between them is broken:

```
>>> nested = [2]
>>> original
[[0], 1]
```

Figure 5.2 illustrates this condition.

You've seen that you can obtain a copy of a list by taking a full slice (that is, `x[:]`). You can also obtain a copy of a list by using the + or * operator (for example, `x + []` or `x * 1`). These techniques are slightly less efficient than the slice method. All three create what is called a *shallow* copy of the list, which is probably what you want most of the time. But if your list has other lists nested in it, you may want to make a *deep* copy. You can do this with the `deepcopy` function of the `copy` module:

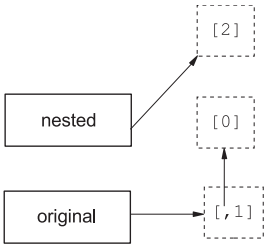


Figure 5.2 The first item of the original list is still a nested list, but the nested variable refers to a different list.

```
>>> original = [[0], 1]
>>> shallow = original[:]
>>> import copy
>>> deep = copy.deepcopy(original)
```

See figure 5.3 for an illustration.

The lists pointed at by the original or shallow variables are connected. Changing the value in the nested list through either one of them affects the other:

```
>>> shallow[1] = 2
>>> shallow
[[0], 2]
>>> original
[[0], 1]
>>> shallow[0][0] = 'zero'
>>> original
[['zero'], 1]
```

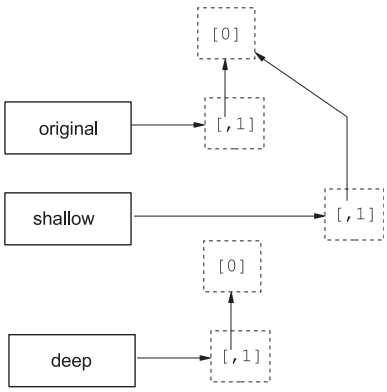


Figure 5.3 A shallow copy doesn't copy nested lists.

The deep copy is independent of the original, and no change to it has any effect on the original list:

```
>>> deep[0][0] = 5
>>> deep
[[5], 1]
>>> original
[['zero'], 1]
```

This behavior is the same for any other nested objects in a list that are modifiable (such as dictionaries).

Now that you've seen what lists can do, it's time to look at tuples.

TRY THIS: LIST COPIES Suppose that you have the following list: `x = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]` What code could you use to get a copy `y` of that list in which you could change the elements *without* the side effect of changing the contents of `x`?

5.7 Tuples

Tuples are data structures that are very similar to lists, but they can't be modified; they can only be created. Tuples are so much like lists that you may wonder why Python bothers to include them. The reason is that tuples have important roles that can't be efficiently filled by lists, such as keys for dictionaries.

5.7.1 Tuple basics

Creating a tuple is similar to creating a list: assign a sequence of values to a variable. A list is a sequence that's enclosed by `[` and `]`; a tuple is a sequence that's enclosed by `(` and `)`:

```
>>> x = ('a', 'b', 'c')
```

This line creates a three-element tuple.

After a tuple is created, using it is so much like using a list that it's easy to forget that tuples and lists are different data types:

```
>>> x[2]
'c'
>>> x[1:]
('b', 'c')
>>> len(x)
3
>>> max(x)
'c'
>>> min(x)
'a'
>>> 5 in x
False
>>> 5 not in x
True
```

The main difference between tuples and lists is that tuples are immutable. An attempt to modify a tuple results in a confusing error message, which is Python's way of saying that it doesn't know how to set an item in a tuple:

```
>>> x[2] = 'd'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

You can create tuples from existing ones by using the + and * operators:

```
>>> x + x
('a', 'b', 'c', 'a', 'b', 'c')
>>> 2 * x
('a', 'b', 'c', 'a', 'b', 'c')
```

A copy of a tuple can be made in any of the same ways as for lists:

```
>>> x[:]
('a', 'b', 'c')
>>> x * 1
('a', 'b', 'c')
>>> x + ()
('a', 'b', 'c')
```

If you didn't read section 5.6, you can skip the rest of this paragraph. Tuples themselves can't be modified. But if they contain any mutable objects (for example, lists or dictionaries), these objects may be changed if they're still assigned to their own variables. Tuples that contain mutable objects aren't allowed as keys for dictionaries.

5.7.2 One-element tuples need a comma

A small syntactical point is associated with using tuples. Because the square brackets used to enclose a list aren't used elsewhere in Python, it's clear that `[]` means an empty list and that `[1]` means a list with one element. The same thing isn't true of the parentheses used to enclose tuples. Parentheses can also be used to group items in expressions to force a certain evaluation order. If you say `(x + y)` in a Python program, do you mean that `x` and `y` should be added and then put into a one-element tuple, or do you mean that the parentheses should be used to force `x` and `y` to be added before any expressions to either side come into play?

This situation is a problem only for tuples with one element, because tuples with more than one element always include commas to separate the elements, and the commas tell Python that the parentheses indicate a tuple, not a grouping. In the case of one-element tuples, Python requires that the element in the tuple be followed by a comma, to disambiguate the situation. In the case of zero-element (empty) tuples, there's no problem. An empty set of parentheses must be a tuple because it's meaningless otherwise:

```
>>> x = 3
>>> y = 4
>>> (x + y)   # This line adds x and y.
```

```
7
>>> (x + y,)  # Including a comma indicates that the parentheses denote a
     tuple.
(7,)
>>> ()        # To create an empty tuple, use an empty pair of parentheses.
()
```

5.7.3 Packing and unpacking tuples

As a convenience, Python permits tuples to appear on the left side of an assignment operator, in which case variables in the tuple receive the corresponding values from the tuple on the right side of the assignment operator. Here's a simple example:

```
>>> (one, two, three, four) =  (1, 2, 3, 4)
>>> one
1
>>> two
2
```

This example can be written even more simply, because Python recognizes tuples in an assignment context even without the enclosing parentheses. The values on the right side are packed into a tuple and then unpacked into the variables on the left side:

```
one, two, three, four = 1, 2, 3, 4
```

One line of code has replaced the following four lines of code:

```
one = 1
two = 2
three = 3
four = 4
```

This technique is a convenient way to swap values between variables. Instead of saying

```
temp = var1
var1 = var2
var2 = temp
```

simply say

```
var1, var2 = var2, var1
```

To make things even more convenient, Python 3 has an extended unpacking feature, allowing an element marked with `*` to absorb any number of elements not matching the other elements. Again, some examples make this feature clearer:

```
>>> x = (1, 2, 3, 4)
>>> a, b, *c = x
>>> a, b, c
(1, 2, [3, 4])
>>> a, *b, c = x
>>> a, b, c
(1, [2, 3], 4)
>>> *a, b, c = x
>>> a, b, c
```

```
([1, 2], 3, 4)
>>> a, b, c, d, *e = x
>>> a, b, c, d, e
(1, 2, 3, 4, [])
```

Note that the starred element receives all the surplus items as a list and that if there are no surplus elements, the starred element receives an empty list.

Packing and unpacking can also be performed by using list delimiters:

```
>>> [a, b] = [1, 2]
>>> [c, d] = 3, 4
>>> [e, f] = (5, 6)
>>> (g, h) = 7, 8
>>> i, j = [9, 10]
>>> k, l = (11, 12)
>>> a
1
>>> [b, c, d]
[2, 3, 4]
>>> (e, f, g)
(5, 6, 7)
>>> h, i, j, k, l
(8, 9, 10, 11, 12)
```

5.7.4 Converting between lists and tuples

Tuples can be easily converted to lists with the `list` function, which takes any sequence as an argument and produces a new list with the same elements as the original sequence. Similarly, lists can be converted to tuples with the `tuple` function, which does the same thing but produces a new tuple instead of a new list:

```
>>> list((1, 2, 3, 4))
[1, 2, 3, 4]
>>> tuple([1, 2, 3, 4])
(1, 2, 3, 4)
```

As an interesting side note, `list` is a convenient way to break a string into characters:

```
>>> list("Hello")
['H', 'e', 'l', 'l', 'o']
```

This technique works because `list` (and `tuple`) apply to any Python sequence, and a string is just a sequence of characters. (Strings are discussed fully in chapter 6.)

QUICK CHECK: TUPLES Explain why the following operations aren't legal for the tuple `x = (1, 2, 3, 4)`:

```
x.append(1)
   x[1] = "hello"
   del x[2]
```

If you had a tuple `x = (3, 1, 4, 2)`, how might you end up with `x` sorted?

5.8 Sets

A *set* in Python is an unordered collection of objects used when membership and uniqueness in the set are main things you need to know about that object. Like dictionary keys (discussed in chapter 7), the items in a set must be immutable and hashable. This means that ints, floats, strings, and tuples can be members of a set, but lists, dictionaries, and sets themselves can't.

5.8.1 Set operations

In addition to the operations that apply to collections in general, such as `in`, `len`, and iteration in `for` loops, sets have several set-specific operations:

```
>>> x = set([1, 2, 3, 1, 3, 5])          ❶
>>> x
{1, 2, 3, 5}                             ❷
>>> x.add(6)                             ❸
>>> x
{1, 2, 3, 5, 6}
>>> x.remove(5)                          ❹
>>> x
{1, 2, 3, 6}
>>> 1 in x                               ❺
True
>>> 4 in x                               ❺
False
>>> y = set([1, 7, 8, 9])
>>> x | y                                ❻
{1, 2, 3, 6, 7, 8, 9}
>>> x & y                                ❼
{1}
>>> x ^ y                                ❽
{2, 3, 6, 7, 8, 9}
>>>
```

You can create a set by using `set` on a sequence, such as a list ❶. When a sequence is made into a set, duplicates are removed ❷. After creating a set by using the `set` function, you can use `add` ❸ and `remove` ❹ to change the elements in the set. The `in` keyword is used to check for membership of an object in a set ❺. You can also use `|` ❻ to get the union, or combination, of two sets, `&` to get their intersection ❼, and `^` ❽ to find their symmetric difference—that is, elements that are in one set or the other but not both.

These examples aren't a complete listing of set operations but are enough to give you a good idea of how sets work. For more information, refer to the official Python documentation.

5.8.2 Frozensets

Because sets aren't immutable and hashable, they can't belong to other sets. To remedy that situation, Python has another set type, `frozenset`, which is just like a set but

can't be changed after creation. Because frozensets are immutable and hashable, they can be members of other sets:

```
>>> x = set([1, 2, 3, 1, 3, 5])
>>> z = frozenset(x)
>>> z
frozenset({1, 2, 3, 5})
>>> z.add(6)
Traceback (most recent call last):
  File "<pyshell#79>", line 1, in <module>
    z.add(6)
AttributeError: 'frozenset' object has no attribute 'add'
>>> x.add(z)
>>> x
{1, 2, 3, 5, frozenset({1, 2, 3, 5})}
```

QUICK CHECK: SETS If you were to construct a set from the following list, how many elements would the set have?: `[1, 2, 5, 1, 0, 2, 3, 1, 1, (1, 2, 3)]`

LAB 5: EXAMINING A LIST In this lab, the task is to read a set of temperature data (the monthly high temperatures at Heathrow Airport for 1948 through 2016) from a file and then find some basic information: the highest and lowest temperatures, the mean (average) temperature, and the median temperature (the temperature in the middle if all the temperatures are sorted).

The temperature data is in the file lab_05.txt in the source code directory for this chapter. Because I haven't yet discussed reading files, here's the code to read the files into a list:

```
temperatures = []
with open('lab_05.txt') as infile:
     for row in infile:
        temperatures.append(int(row.strip())
```

You should find the highest and lowest temperature, the average, and the median. You'll probably want to use the `min()`, `max()`, `sum()`, `len()`, and `sort()` functions/methods.

BONUS Determine how many unique temperatures are in the list.

Summary

- Lists and tuples are structures that embody the idea of a sequence of elements, as are strings.
- Lists are like arrays in other languages, but with automatic resizing, slice notation, and many convenience functions.
- Tuples are like lists but can't be modified, so they use less memory and can be dictionary keys (see chapter 7).
- Sets are iterable collections, but they're unordered and can't have duplicate elements.

Modeling and prediction

The "science" part of "data science" is the modeling, the part that creates the predictions, correlations, or classifications that form the information we hope to get out a data science project.

For the final chapter of this ebook, I've chosen a chapter from *Real-world Machine Learning* by Henrik Brink, Joseph W. Richards, and Mark Fetherolf on "Modeling and prediction." This chapter lays out the basics of machine learning modeling and explains the difference between supervised and unsupervised learning. From there it uses the Python package scikit-learn to demonstrate several common machine learning approaches, including logistic regression, support vector machines, k-nearest neighbors, linear regression, and random forest algorithms to model problems like the survival chances of passengers on the Titanic, number recognition, and mileage prediction.

Chapter 3 from *Real World Machine Learning* by Henrik Brink, Joseph Richards, and Mark Fetherolf

Modeling and prediction

This chapter covers

- Discovering relationships in data through ML modeling
- Using models for prediction and inference
- Building classification models
- Building regression models

The previous chapter covered guidelines and principles of data collection, preprocessing, and visualization. The next step in the machine-learning workflow is to use that data to begin exploring and uncovering the relationships that exist between the input features and the target. In machine learning, this process is done by building statistical models based on the data. This chapter covers the basics required to understand ML modeling and to start building your own models. In contrast to most machine-learning textbooks, we spend little time discussing the various approaches to ML modeling, instead focusing attention on the big-picture concepts. This will help you gain a broad understanding of machine-learning model building and quickly get up to speed on building your own models to solve real-world problems. For those seeking more information about specific ML modeling techniques, please see the appendix.

We begin the chapter with a high-level overview of statistical modeling. This discussion focuses on the big-picture concepts of ML modeling, such as the purpose of models, the ways in which models are used in practice, and a succinct look at types of modeling techniques in existence and their relative strengths and weaknesses. From there, we dive into the two most common machine-learning models: classification and regression. In these sections, we give more details about how to build models on your data. We also call attention to a few of the most common algorithms used in practice in the "Algorithm highlight" boxes scattered throughout the chapter.

3.1 Basic machine-learning modeling

The objective of machine learning is to discover patterns and relationships in data and to put those discoveries to use. This process of discovery is achieved through the use of modeling techniques that have been developed over the past 30 years in statistics, computer science, and applied mathematics. These various approaches can range from simple to tremendously complex, but all share a common goal: to estimate the functional relationship between the input features and the target variable.

These approaches also share a common workflow, as illustrated in figure 3.1: use of historical data to build and optimize a model that is, in turn, used to make predictions based on new data. This section prepares you for the practical sections later in the chapter. You'll look at the general goal of machine learning modeling in the next section, and move on to seeing how the end product can be used and a few important aspects for differentiating between ML algorithms.

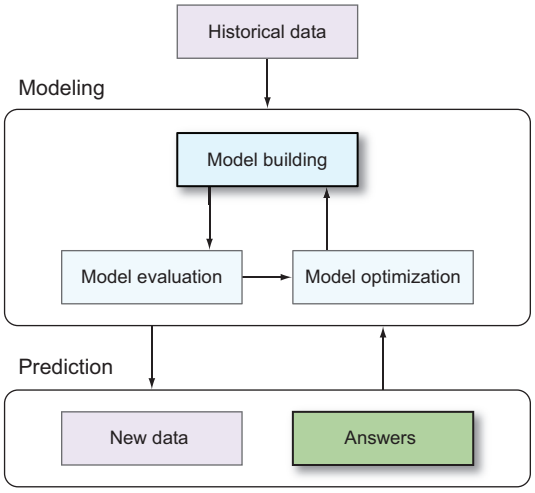


Figure 3.1 The basic ML workflow

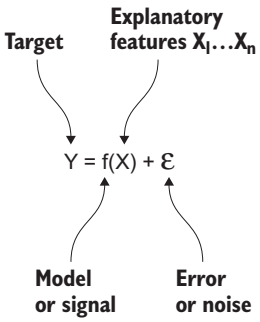
3.1.1 Finding the relationship between input and target

Let's frame the discussion of ML modeling around an example. Recall the Auto MPG dataset from chapter 2. The dataset contains metrics about automobiles, such as

manufacturer region, model year, vehicle weight, horsepower, and number of cylinders. The purpose of the dataset is to understand the relationship between the input features and a vehicle's miles per gallon (MPG) rating.

Input features are typically referred to using the symbol X, with subscripts differentiating inputs when multiple input features exist. For instance, we'll say that X_1 refers to manufacturer region, X_2 to model year, X_3 to vehicle weight, and so forth. The collection of all the input features is referred to as the bold **X**. Likewise, the target variable is typically referred to as Y.

The relationship between the inputs, **X**, and output, Y, can be succinctly represented by this simple formula:



In this equation, f represents the unknown function that relates the input variables to the target, Y. The goal of ML modeling is to accurately estimate f by *using data*. The symbol ε represents random noise in the data that's unrelated to the function f. The function f is commonly referred to as the *signal*, whereas the random variable ε is called the *noise*. The challenge of machine learning is to use data to determine what the true signal is, while ignoring the noise.

In the Auto MPG example, the function f describes the true MPG rating for an automobile as a function of that car's many input features. If you knew that function perfectly, you could know the MPG rating for any car, real or fictional. But you could have numerous sources of noise, ε, including (and certainly not limited to) the following:

- Imperfect measurement of each vehicle's MPG rating caused by small inaccuracies in the measuring devices—measurement noise
- Variations in the manufacturing process, causing each car in the fleet to have slightly different MPG measurements—manufacturing process noise
- Noise in the measurement of the input features, such as weight and horsepower
- Lack of access to the broader set of features that would exactly determine MPG

Using the noisy data that you have from hundreds of vehicles, the ML approach is to use modeling techniques to find a good estimate for f. This resultant estimate is referred to as an *ML model*.

In sections 3.2 and 3.3, we describe in further detail how these ML modeling techniques work. Indeed, the bulk of the academic literature on machine learning deals with how to best estimate f.

3.1.2 The purpose of finding a good model

Assuming that you have a good estimate of f, what next? Machine learning has two main goals: *prediction* and *inference*.

PREDICTION

After you have a model, you can use that model to generate predictions of the target, Y, for new data, $\mathbf{X}_{new}$, by plugging those new features into the model. In mathematical notation, if f_{est} denotes your machine-learning estimate of f (recall that f denotes the true relationship between the features and the target), then predictions for new data can be obtained by plugging the new data into this formula:

$$Y_{pred} = f_{est}(\mathbf{X}_{new})$$

These predictions can then be used to make decisions about the new data or may be fed into an automated workflow.

Going back to the Auto MPG example, suppose that you have an ML model, f_{est}, that describes the relationship between MPG and the input metrics of an automobile. Prediction allows you to ask the question, "What would the MPG of a certain automobile with known input metrics be?" Such a predictive ability would be useful for designing automobiles, because it would allow engineers to assess the MPG rating of different design concepts and to ensure that the individual concepts meet MPG requirements.

Prediction is the most common use of machine-learning systems. Prediction is central to many ML use cases, including these:

- Deciphering handwritten digits or voice recordings
- Predicting the stock market
- Forecasting
- Predicting which users are most likely to click, convert, or buy
- Predicting which users will need product support and which are likely to unsubscribe
- Determining which transactions are fraudulent
- Making recommendations

Because of the high levels of predictive accuracy attained by machine-learning approaches and the rapid speed by which ML predictions can be generated, ML is used every day by thousands of companies for predictive purposes.

INFERENCE

In addition to making predictions on new data, you can use machine-learning models to better understand the relationships between the input features and the output target.

A good estimate of f can enable you to answer deep questions about the associations between the variables at hand. For example:

- Which input features are most strongly related to the target variable?
- Are those relationships positive or negative?
- Is f a simple relationship, or is it a function that's more nuanced and nonlinear?

These inferences can tell you a lot about the data-generating process and give clues to the factors driving relationships in the data. Returning to the Auto MPG example, you can use inference to answer questions such as these: Does manufacturer region have an effect on MPG? Which of the inputs are most strongly related to MPG? And are they negatively or positively related? Answers to these questions can give you an idea of the driving factors in automobile MPG and give clues about how to engineer vehicles with higher MPG.

3.1.3 Types of modeling methods

Now the time has come to dust off your statistics knowledge and dive into some of the mathematical details of ML modeling. Don't worry—we'll keep the discussion relatively broad and understandable for those without much of a statistics background!

Statistical modeling has a general trade-off between predictive accuracy and model interpretability. Simple models are easy to interpret, yet won't produce accurate predictions (particularly for complicated relationships). Complex models may produce accurate predictions, but may be black-box and hard to interpret.

In addition, the machine-learning model has two main types: parametric and nonparametric. The essential difference is that parametric models assume that f takes a specific functional form, whereas nonparametric models don't make such strict assumptions. Therefore, parametric approaches tend to be simple and interpretable, but less accurate. Likewise, nonparametric approaches are usually less interpretable but more accurate across a broad range of problems. Let's take a closer look at both parametric and nonparametric approaches to ML modeling.

Parametric methods

The simplest example of a parametric approach is linear regression. In linear regression, f is assumed to be a linear combination of the numerical values of the inputs. The standard linear regression model is as follows:

$$f(\mathbf{X}) = \beta_0 + X_1 \times \beta_1 + X_2 \times \beta_2 + \ldots$$

In this equation, the unknown parameters, β_0, β_1,… can be interpreted as the intercept and slope parameters (with respect to each of the inputs). When you fit a parametric model to some data, you estimate the best values of each of the unknown parameters. Then you can turn around and plug those estimates into the formula for $f(\mathbf{X})$ along with new data to generate predictions.

Other examples of commonly used parametric models include logistic regression, polynomial regression, linear discriminant analysis, quadratic discriminant analysis,

(parametric) mixture models, and naïve Bayes (when parametric density estimation is used). Approaches often used in conjunction with parametric models for model selection purposes include ridge regression, lasso, and principal components regression. Further details about some of these methods are given later in this chapter, and a description of each approach is given in the appendix.

The drawback of parametric approaches is that they make strong assumptions about the true form of the function f. In most real-world problems, f doesn't assume such a simple form, especially when there are many input variables (X). In these situations, parametric approaches will fit the data poorly, leading to inaccurate predictions. Therefore, most real-world approaches to machine learning depend on nonparametric machine-learning methods.

NONPARAMETRIC METHODS

In *nonparametric* models, f doesn't take a simple, fixed function. Instead, the form and complexity of f adapts to the complexity of the data. For example, if the relationship between X and Y is wiggly, a nonparametric approach will choose a function f that matches the curvy patterns. Likewise, if the relationship between the input and output variable is smooth, a simple function f will be chosen.

A simple example of a nonparametric model is a classification tree. A *classification tree* is a series of recursive binary decisions on the input features. The classification tree learning algorithm uses the target variable to learn the optimal series of splits such that the terminal leaf nodes of the tree contain instances with similar values of the target.

Take, for example, the Titanic Passengers dataset. The classification tree algorithm first seeks the best input feature to split on, such that the resulting leaf nodes contain passengers who either mostly lived or mostly died. In this case, the best split is on the sex (male/female) of the passenger. The algorithm continues splitting on other input features in each of the subnodes until the algorithm can no longer detect any good subsequent splits.

Classification trees are nonparametric because the depth and complexity of the tree isn't fixed in advance, but rather is learned from the data. If the relationship between the target variable and the input features is complex and there's a sufficient amount of data, then the tree will grow deeper, uncovering more-nuanced patterns. Figure 3.2 shows two classification trees learned from different subsets of the Titanic Passengers dataset. In the left panel is a tree learned from only 400 passengers: the resultant model is simple, consisting of only a single split. In the right panel is a tree learned from 891 passengers: the larger amount of data enables the model to grow in complexity and find more-detailed patterns in the data.

Other examples of nonparametric approaches to machine learning include k-nearest neighbors, splines, basis expansion methods, kernel smoothing, generalized additive models, neural nets, bagging, boosting, random forests, and support vector machines. Again, more details about some of these methods are given later in this chapter, and a description of each approach is given in the appendix.

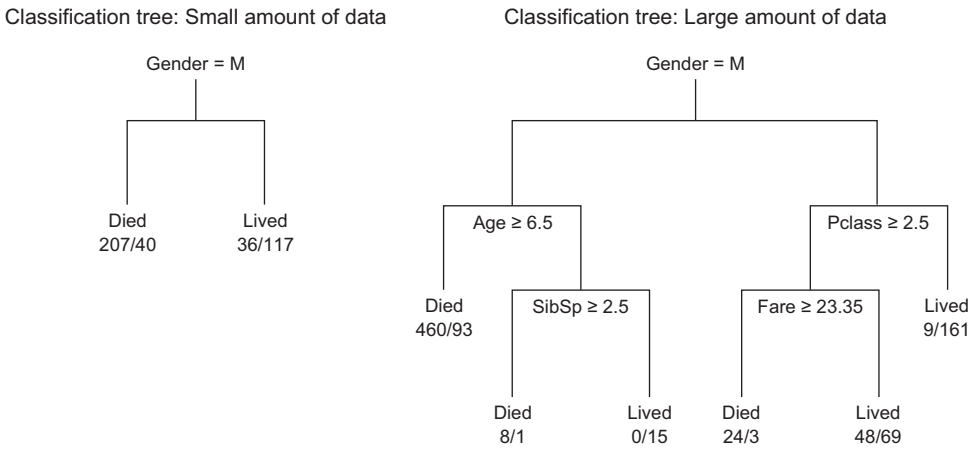


Figure 3.2 A decision tree is an example of a nonparametric ML algorithm, because its functional form isn't fixed. The tree model can grow in complexity with larger amounts of data to capture more-complicated patterns. In each terminal node of the tree, the ratio represents the number of training instances in that node that died versus lived.

3.1.4 Supervised versus unsupervised learning

Machine-learning problems fall into two camps: supervised and unsupervised. *Supervised problems* are ones in which you have access to the target variable for a set of training data, and *unsupervised problems* are ones in which there's no identified target variable.

All the examples so far in this book fall in the supervised camp. These problems each contain a target of interest (Did the Titanic passenger survive? Did the customer churn? What's the MPG?) and a set of training data with known values of the target. Indeed, most problems in machine learning are supervised in nature, and most ML techniques are designed to solve supervised problems. We spend the vast majority of this book describing how to solve supervised problems.

In unsupervised learning, you have access to only input features, and don't have an associated target variable. So what kinds of analyses can you perform if there's no target available? The unsupervised learning approach has two main classes:

- *Clustering*—Use the input features to discover natural groupings in the data and to divide the data into those groups. Methods: k-means, Gaussian mixture models, and hierarchical clustering.
- *Dimensionality reduction*—Transform the input features into a small number of coordinates that capture most of the variability of the data. Methods: principal component analysis (PCA), multidimensional scaling, manifold learning.

Both clustering and dimensionality reduction have wide popularity (particularly, k-means and PCA), yet are often abused and used inappropriately when a supervised approach is warranted.

But unsupervised problems do play a significant role in machine learning, often in support of supervised problems, either to help compile training data for learning or to derive new input features on which to learn. You'll return to the topic of unsupervised learning in chapter 8.

Now, let's transition to the more practical aspects of ML modeling. Next we describe the steps needed to start building models on your own data and the practical considerations of choosing which algorithm to use. We break up the rest of the chapter into two sections corresponding to the two most common problems in machine learning: classification and regression. We begin with the topic of classification.

3.2 Classification: predicting into buckets

In machine learning, *classification* describes the prediction of new data into buckets (classes) by using a *classifier* built by the machine-learning algorithm. Spam detectors put email into Spam and No Spam buckets, and handwritten digit recognizers put images into buckets from 0 through 9, for example. In this section, you'll learn how to build classifiers based on the data at hand. Figure 3.3 illustrates the process of classification.

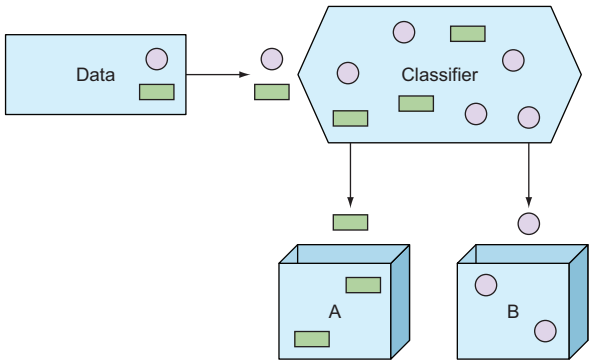


Figure 3.3 A classification process. Rectangles and circles are divided by a classifier into classes A and B. This is a case of binary classification with only two classes.

Let's again use an example. In chapter 2, you looked at the Titanic Passengers dataset for predicting survival of passengers onboard the ill-fated ship. Figure 3.4 shows a subset of this data.

PassengerId	Survived	Pclass	Gender	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Male	22	1	0	A/5 21171	7.25		S
2	1	1	Female	38	1	0	PC 17599	71.2833	C85	C
3	1	3	Female	26	0	0	STON/02. 3101282	7.925		S
4	1	1	Female	35	1	0	113803	53.1	C123	S
5	0	3	Male	35	0	0	373450	8.05		S
6	0	3	Male		0	0	330877	8.4583		Q

Figure 3.4 A subset of the Titanic Passengers dataset

As we've previously discussed, typically the best way to start an ML project is to get a feel for the data by visualizing it. For example, it's considered common knowledge that more women than men survived the Titanic, and you can see that this is the case from the mosaic plot in figure 3.5 (if you've forgotten about mosaic plots, look back at section 2.3.1).

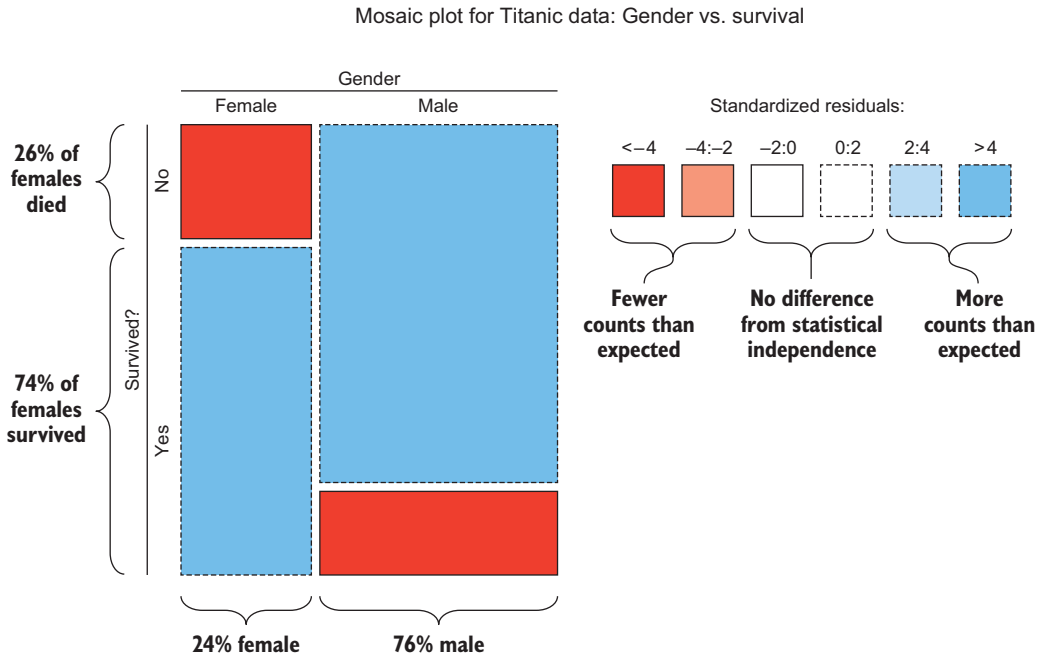


Figure 3.5 **Mosaic plot showing overwhelming support for the idea that more women than men survived the disaster.**

By using the visualization techniques in section 2.3, you can get a feeling for the performance of each feature in the Titanic Passengers dataset. But it's important to realize that just because a single feature looks good or bad, it doesn't necessarily show the performance of the feature in combination with one or more other features. Maybe the age together with the sex and social status divides the passengers much better than any single feature. In fact, this is one of the main reasons to use machine-learning algorithms in the first place: to find signals in many dimensions that humans can't discover easily.

The following subsections introduce the methodology for building classification models and making predictions. You'll look at a few specific algorithms and the difference between linear and nonlinear algorithms.

3.2.1 Building a classifier and making predictions

The first order of business is to choose the classification algorithm to use for building the classifier. Many algorithms are available, and each has pros and cons for different data and deployment requirements. The appendix provides a table of algorithms and a comparison of their properties. You'll use this table throughout the book for selecting algorithms to try for different problems. In this section, the choice of algorithm isn't essential; in the next chapter, you'll learn how to properly measure the performance of the algorithm and choose the best for the job.

The next step is to ready the data for modeling. After exploring some of the features in the dataset, you may want to preprocess the data to deal with categorical features, missing values, and so on (as discussed in chapter 2). The preprocessing requirements are also dependent on the specific algorithm, and the appendix lists these requirements for each algorithm.

For the Titanic survival model, you'll start by choosing a simple classification algorithm: logistic regression.[1] For logistic regression, you need to do the following:

1. Impute missing values.
2. Expand categorical features.
3. From chapter 2, you know that the Fare feature is heavily skewed. In this situation, it's advantageous (for some ML models) to transform the variable to make the feature distribution more symmetric and to reduce the potentially harmful impact of outliers. Here, you'll choose to transform Fare by taking the square root.

The final dataset that you'll use for modeling is shown in figure 3.6.

Pclass	Age	SibSp	Parch	sqrt_Fare	Gender = female	Gender = male	Embarked = C	Embarked = Q	Embarked = S
3	22	1	0	2.692582	0	1	0	0	1
1	38	1	0	8.442944	1	0	1	0	0
3	26	0	0	2.815138	1	0	0	0	1
1	35	1	0	7.286975	1	0	0	0	1
3	35	0	0	2.837252	0	1	0	0	1

Figure 3.6 The first five rows of the Titanic Passengers dataset after processing categorical features and missing values, and transforming the Fare variable by taking the square root (see the `prepare_data` function in the source code repository). All features are now numerical, which is the preferred format for most ML algorithms.

You can now go ahead and build the model by running the data through the logistic regression algorithm. This algorithm is implemented in the scikit-learn Python package, and the model-building and prediction code look like the following listing.

[1] The *regression* in logistic regression doesn't mean it's a regression algorithm. Logistic regression expands linear regression with a logistic function to make it suitable for classification.

Listing 3.1 Building a logistic regression classifier with scikit-learn

```
from sklearn.linear_model import LogisticRegression as Model

def train(features, target):
  model = Model()
  model.fit(features, target)
  return model

def predict(model, new_features):
  preds = model.predict(new_features)
  return preds

# Assume Titanic data is loaded into titanic_feats,
# titanic_target and titanic_test
model = train(titanic_feats, titanic_target)
predictions = predict(model, titanic_test)
```

Imports the logistic regression algorithm

Fits the logistic regression algorithm using features and target data

Makes predictions on a new set of features using the model

Returns the model built by the algorithm

Returns predictions (0 or 1)

After building the model, you predict the survival of previously unseen passengers based on their features. The model expects features in the format given in figure 3.6, so any new passengers will have to be run through exactly the same processes as the training data. The output of the `predict` function will be 1 if the passenger is predicted to survive, and 0 otherwise.

It's useful to visualize the classifier by plotting the decision boundary. Given two of the features in the dataset, you can plot the boundary that separates surviving passengers from the dead, according to the model. Figure 3.7 shows this for the Age and square-root Fare features.

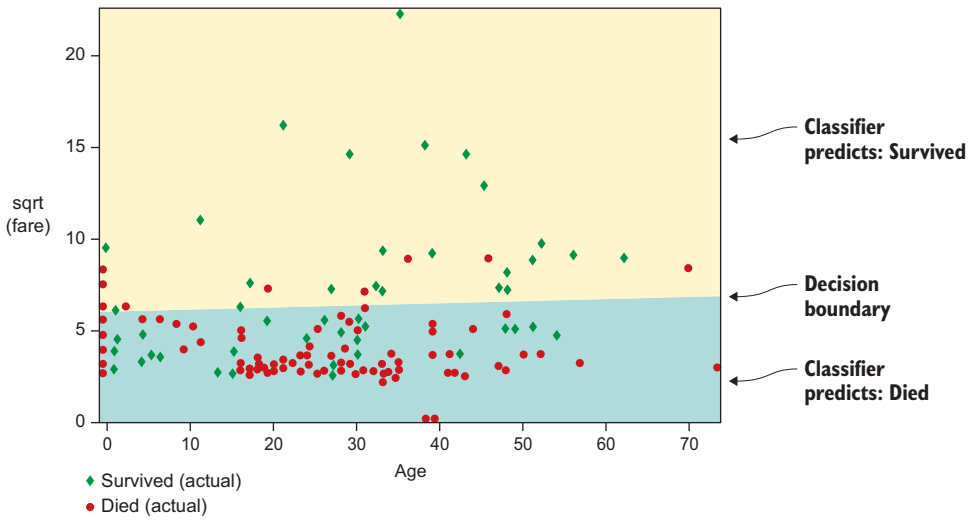


Figure 3.7 The decision boundary for the Age and sqrt(Fare) features. The diamonds show passengers who survived, whereas circles denote passengers who died. The light background denotes the combinations of Age and Fare that are predicted to yield survival. Notice that a few instances overlap the boundary. The classifier isn't perfect, but you're looking in only two dimensions. The algorithm on the full dataset finds this decision boundary in 10 dimensions, but that becomes harder to visualize.

Algorithm highlight: logistic regression

In these "Algorithm highlight" boxes, you'll take a closer look at the basic ideas behind the algorithms used throughout the book. This allows curious readers to try to code up, with some extra research, basic working versions of the algorithms. Even though we focus mostly on the use of existing packages in this book, understanding the basics of a particular algorithm can sometimes be important to fully realize the predictive potential.

The first algorithm you'll look at is the *logistic regression algorithm*, arguably the simplest ML algorithm for classification tasks. It's helpful to think about the problem as having only two features and a dataset divided into two classes. Figure 3.7 shows an example, with the features Age and sqrt(Fare); the target is Survived or Died. To build the classifier, you want to find the line that best splits the data into the target classes. A line in two dimensions can be described by two parameters. These two numbers are the parameters of the model that you need to determine.

The algorithm then consists of the following steps:

1. You can start the search by picking the parameter values at random, hence placing a random line in the two-dimensional figure.
2. Measure how well this line separates the two classes. In logistic regression, you use the statistical *deviance* for the goodness-of-fit measurement.
3. Guess new values of the parameters and measure the separation power.
4. Repeat until there are no better guesses. This is an *optimization* procedure that can be done with a range of optimization algorithms. Gradient descent is a popular choice for a simple optimization algorithm.

This approach can be extended to more dimensions, so you're not limited to two features in this model. If you're interested in the details, we strongly encourage you to research further and try to implement this algorithm in your programming language of choice. Then look at an implementation in a widely used ML package. We've left out plenty of details, but the preceding steps remain the basis of the algorithm.

Some properties of logistic regression include the following:

- The algorithm is relatively simple to understand, compared to more-complex algorithms. It's also computationally simple, making it scalable to large datasets.
- The performance will degrade if the decision boundary that separates the classes needs to be highly nonlinear. See section 3.2.2.
- Logistic regression algorithms can sometimes overfit the data, and you often need to use a technique called *regularization* that limits this danger. See section 3.2.2 for an example of overfitting.

Further reading

If you want to learn more about logistic regression and its use in the real world, check out *Applied Logistic Regression* by David Hosmer et al. (Wiley, 2013).

3.2.2 Classifying complex, nonlinear data

Looking at figure 3.7, you can understand why logistic regression is a linear algorithm: the decision boundary is a straight line. Of course, your data might not be well separated by a straight line, so for such datasets you should use a nonlinear algorithm. But nonlinear algorithms are typically more demanding computationally and don't scale well to large datasets. You'll look further at the scalability of various types of algorithms in chapter 8.

Looking again at the appendix, you can pick a nonlinear algorithm for modeling the Titanic Passengers dataset. A popular method for nonlinear problems is a support vector machine with a nonlinear kernel. Support vector machines are linear by nature, but by using a kernel, this model becomes a powerful nonlinear method. You can change a single line of code in listing 3.1 to use this new algorithm, and the decision boundary is plotted in figure 3.8:

```
from sklearn.svm import SVC as Model
```

You can see that the decision boundary in figure 3.8 is different from the linear one in figure 3.7. What you see here is a good example of an important concept in machine learning: overfitting. The algorithm is capable of fitting well to the data, almost at the single-record level, and you risk losing the ability to make good predictions on new data that wasn't included in the training set; the more complex you allow the model to become, the higher the risk of overfitting.

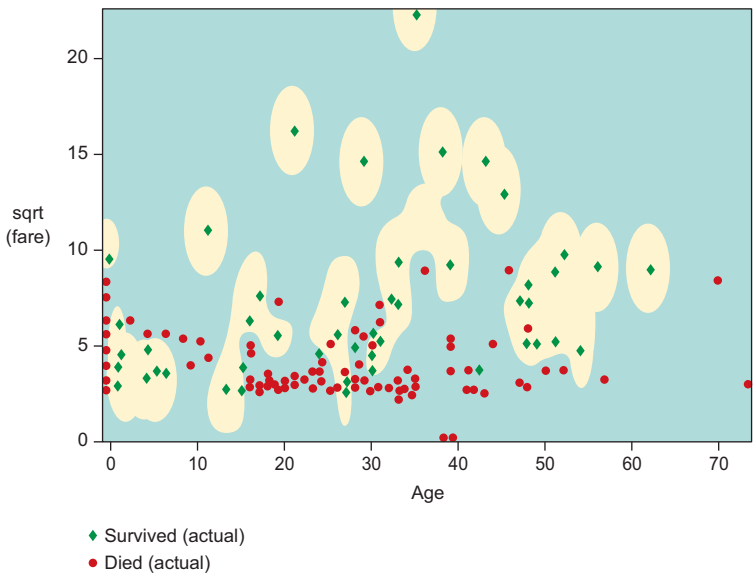


Figure 3.8 Nonlinear decision boundary of the Titanic survival support vector machine classifier with a nonlinear kernel. The light background denotes the combinations of Age and Fare that are predicted to yield survival.

Usually, you can avoid overfitting a nonlinear model by using model parameters built into the algorithm. By tweaking the parameters of the model, keeping the data unchanged, you can obtain a better decision boundary. Note that you're currently using intuition to determine when something is overfitting; in chapter 4, you'll learn how to use data and statistics to quantify this intuition. For now, you'll use our (the authors') experience and tweak a certain parameter called *gamma*. You don't need to know what gamma is at this point, only that it helps control the risk of overfitting. In chapter 5, you'll see how to optimize the model parameters without only guessing at better values. Setting gamma = 0.1 in the SVM classifier, you obtain the much improved decision boundary shown in figure 3.9.

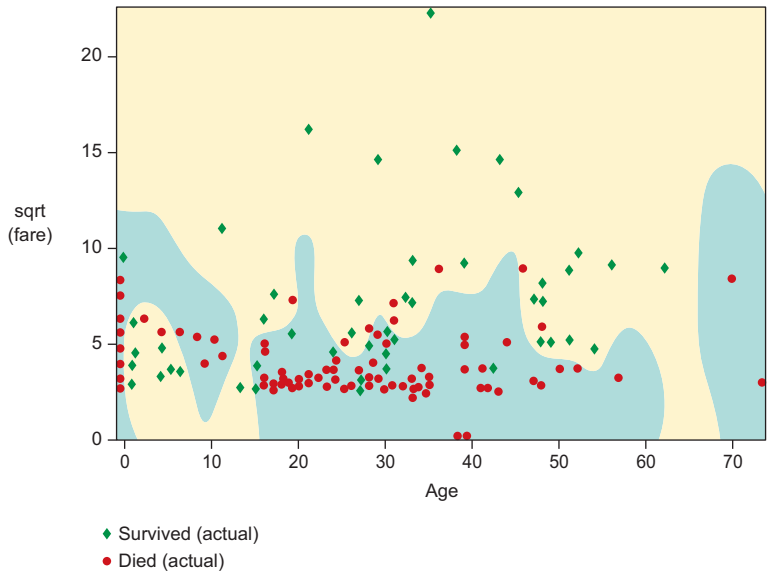


Figure 3.9 Decision boundary of nonlinear RBF-kernel SVM with gamma = 0.1

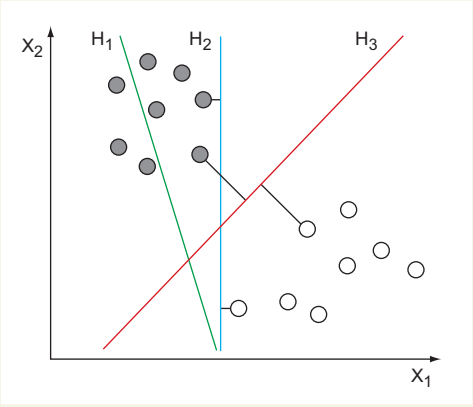
Algorithm highlight: support vector machines

The support vector machine (SVM) algorithm is a popular choice for both linear and nonlinear problems. It has some interesting theoretical and practical properties that make it useful in many scenarios.

The main idea behind the algorithm is, as with logistic regression discussed previously, to find the line (or equivalent in higher dimensions) that separates the classes optimally. Instead of measuring the distance to all points, SVMs try to find the largest *margin* between only the points on either side of the decision line. The idea is that there's no reason to worry about points that are well within the boundary, only ones that are close. In the following image, you can see that lines H_1 and H_2 are bad separation

(continued)

boundaries, because the distance to the closest point on both sides of the line isn't the largest it can be. H_3 is the optimal line.



An SVM decision boundary (H_3) is often superior to decision boundaries found by other ML algorithms.

Although this algorithm is also linear in the sense that the separation boundary is linear, SVMs are capable of fitting to nonlinear data, as you saw earlier in this section. SVMs use a clever technique in order to fit to nonlinear data: the kernel trick. A *kernel* is a mathematical construct that can "warp" the space where the data lives. The algorithm can then find a linear boundary in this warped space, making the boundary nonlinear in the original space.

Further reading

Hundreds of books have been written about machine-learning algorithms, covering everything from their theoretical foundation and efficient implementation to their practical use. If you're looking for a more rigorous treatment of these topics, we recommend two classic texts on ML algorithms:

3.2.3 Classifying with multiple classes

Up to this point, you've looked at classification into only two classes. In some cases, you'll have more than two classes. A good real-world example of multiclass classification is the handwritten digit recognition problem. Whenever you send old-school mail to your family, a robot reads the handwritten ZIP code and determines where to send the letter, and good digit recognition is essential in this process. A public dataset, the

MNIST database,[1] is available for research into these types of problems. This dataset consists of 60,000 images of handwritten digits. Figure 3.10 shows a few of the handwritten digit images.

Figure 3.10 Four randomly chosen handwritten digits from the MNIST database

The images are 28 × 28 pixels each, but we convert each image into $28^2 = 784$ features, one feature for each pixel. In addition to being a multiclass problem, this is also a high-dimensional problem. The pattern that the algorithm needs to find is a complex combination of many of these features, and the problem is nonlinear in nature.

To build the classifier, you first choose the algorithm to use from the appendix. The first nonlinear algorithm on the list that natively supports multiclass problems is the k-nearest neighbors classifier, which is another simple but powerful algorithm for nonlinear ML modeling. You need to change only one line in listing 3.1 to use the new algorithm, but you'll also include a function for getting the full prediction probabilities instead of just the final prediction:

```
from sklearn.neighbors import KNeighborsClassifier as Model

def predict_probabilities(model, new_features):
  preds = model.predict_proba(new_features)
  return preds
```

Building the k-nearest neighbors classifier and making predictions on the four digits shown in figure 3.10, you obtain the table of probabilities shown in figure 3.11.

You can see that the predictions for digits 1 and 3 are spot on, and there's only a small (10%) uncertainty for digit 4. Looking at the second digit (3), it's not surprising that this is hard to classify perfectly. This is the main reason to get the full probabilities in the first place: to be able to take action on things that aren't perfectly certain. This is easy to understand in the case of a post office robot routing letters; if the robot is sufficiently uncertain about some digits, maybe we should have a good old human look at it before we send it out wrong.

[1] You can find the MNIST Database of Handwritten Digits at http://yann.lecun.com/exdb/mnist/.

	Actual value	0	1	2	3	4	5	6	7	8	9
Digit 1	7	0	0	0	0.0	0	0.0	0	1	0	0.0
Digit 2	3	0	0	0	0.7	0	0.2	0	0	0	0.1
Digit 3	9	0	0	0	0.0	0	0.0	0	0	0	1.0
Digit 4	5	0	0	0	0.0	0	0.9	0	0	0	0.1

Predicted digit

Digit 2 has a probability of 0.2 of being 5.

Figure 3.11 Table of predicted probabilities from a k-nearest neighbors classifier, as applied to the MNIST dataset

Algorithm highlight: k-nearest neighbors

The *k-nearest neighbors algorithm* is a simple yet powerful nonlinear ML method. It's often used when model training should be quick, but predictions are typically slower. You'll soon see why this is the case.

The basic idea is that you can classify a new data record by comparing it with similar records from the training set. If a dataset record consists of a set of numbers, n_i, you can find the *distance* between records via the usual distance formula:

$$d = \sqrt{n_1^2 + n_2^2 \cdots + n_n^2}.$$

When making predictions on new records, you find the *closest* known record and assign that class to the new record. This would be a 1-nearest neighbor classifier, as you're using only the closest neighbor. Usually you'd use 3, 5, or 9 neighbors and pick the class that's most common among neighbors (you use odd numbers to avoid ties).

The training phase is relatively quick, because you index the known records for fast distance calculations to new data. The prediction phase is where most of the work is done, finding the closest neighbors from the entire dataset.

The previous simple example uses the usual Euclidean distance metric. You can also use more-advanced distance metrics, depending on the dataset at hand.

K-nearest neighbors is useful not only for classification, but for regression as well. Instead of taking the most common class of neighbors, you take the average or median values of the target values of the neighbors. Section 3.3 further details regression.

3.3 Regression: predicting numerical values

Not every machine-learning problem is about putting records into classes. Sometimes the target variable takes on numerical values—for example, when predicting dollar

values in a financial model. We call the act of predicting numerical values *regression*, and the model itself a *regressor*. Figure 3.12 illustrates the concept of regression.

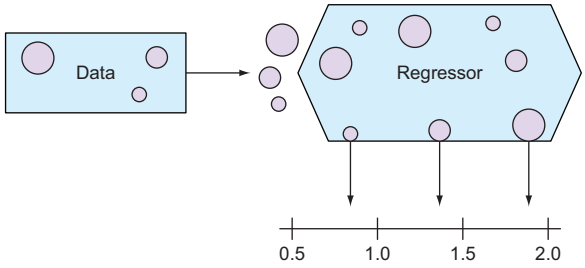


Figure 3.12 **In this regression process, the regressor is predicting the numerical value of a record.**

As an example of a regression analysis, you'll use the Auto MPG dataset introduced in chapter 2. The goal is to build a model that can predict the average miles per gallon of a car, given various properties of the car such as horsepower, weight, location of origin, and model year. Figure 3.13 shows a small subset of this data.

	MPG	Cylinders	Displacement	Horsepower	Weight	Acceleration	Model/year	Origin
0	18	8	307	130	3504	12.0	70	1
1	15	8	350	165	3693	11.5	70	1
2	18	8	318	150	3436	11.0	70	1
3	16	8	304	150	3433	12.0	70	1
4	17	8	302	140	3449	10.5	70	1

Figure 3.13 **Small subset of the Auto MPG data**

In chapter 2, you discovered useful relationships between the MPG rating, the car weight, and the model year. These relationships are shown in figure 3.14.

In the next section, you'll look at how to build a basic linear regression model to predict the miles per gallon values of this dataset of vehicles. After successfully building a basic model, you'll look at more-advanced algorithms for modeling nonlinear data.

3.3.1 Building a regressor and making predictions

Again, you'll start by choosing an algorithm to use and getting the data into a suitable format. Arguably, the linear regression algorithm is the simplest regression algorithm. As the name indicates, this is a linear algorithm, and the appendix shows the data preprocessing needed in order to use this algorithm. You need to (1) impute missing values and (2) expand categorical features. Our Auto MPG dataset has no missing values,

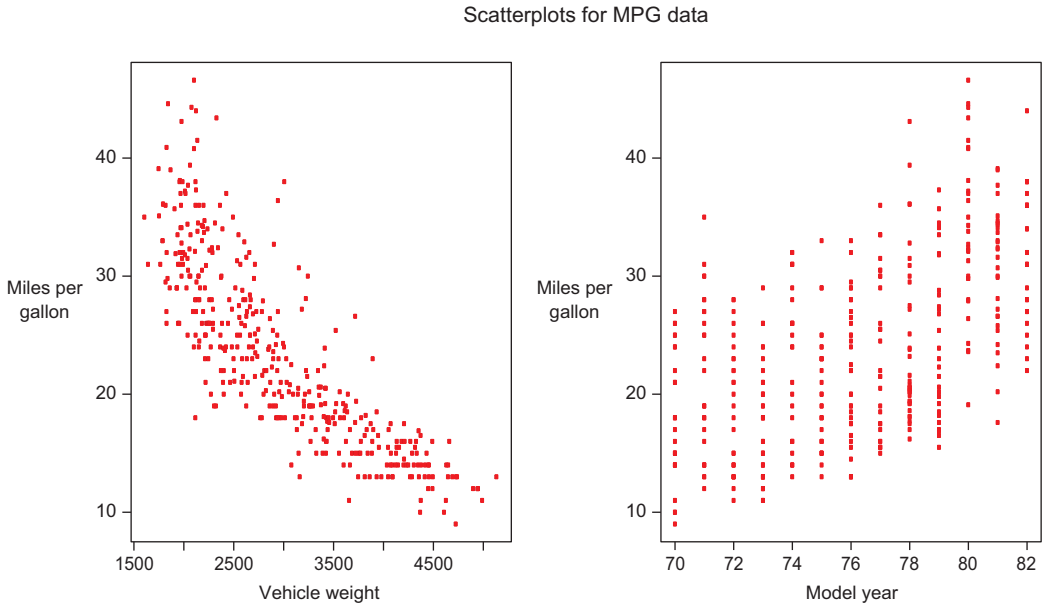


Figure 3.14 **Using scatter plots, you can see that Vehicle Weight and Model Year are useful for predicting MPG. See chapter 2 for more details.**

but there's one categorical column: Origin. After expanding the Origin column (as described in section 2.2.1 in chapter 2), you obtain the data format shown in figure 3.15.

You can now use the algorithm to build the model. Again, you can use the code structure defined in listing 3.1 and change this line:

```
from sklearn.linear_model import LinearRegression as Model
```

With the model in hand, you can make predictions. In this example, however, you'll split the dataset into a training set and a testing set before building the model. In chapter 4, you'll learn much more about how to evaluate models, but you'll use some

	MPG	Cylinders	Displacement	Horsepower	Weight	Acceleration	Model/year	Origin = 1	Origin = 2	Origin = 3
387	27	4	140	86	2790	15.6	82	1	0	0
388	44	4	97	52	2130	24.6	82	0	1	0
389	32	4	135	84	2295	11.6	82	1	0	0
390	28	4	120	79	2625	18.6	82	1	0	0
391	31	4	119	82	2720	19.4	82	1	0	0

Figure 3.15 **The Auto MPG data after expanding the categorical Origin column**

simple techniques in this section. By training a model on only some of the data while holding out a testing set, you can subsequently make predictions on the testing set and see how close your predictions come to the actual values. If you were training on all the data and making predictions on some of that training data, you'd be cheating, as the model is more likely to make good predictions if it's seen the data while training.

Figure 3.16 shows the results of making predictions on a held-out testing set, and how they compare to the known values. In this example, you train the model on 80% of the data and use the remaining 20% for testing.

Origin = 1	Origin = 3	Origin = 2	MPG	Predicted MPG
0	0	1	26.0	27.172795
1	0	0	23.8	24.985776
1	0	0	13.0	13.601050
1	0	0	17.0	15.181120
1	0	0	16.9	16.809079

Figure 3.16 Comparing MPG predictions on a held-out testing set to actual values

A useful way to compare more than a few rows of predictions is to use our good friend, the scatter plot, once again. For regression problems, both the actual target values and the predicted values are numeric. Plotting the predictions against each other in a scatter plot, introduced in chapter 2, you can visualize how well the predictions follow the actual values. This is shown for the held-out Auto MPG test set in figure 3.17. This figure

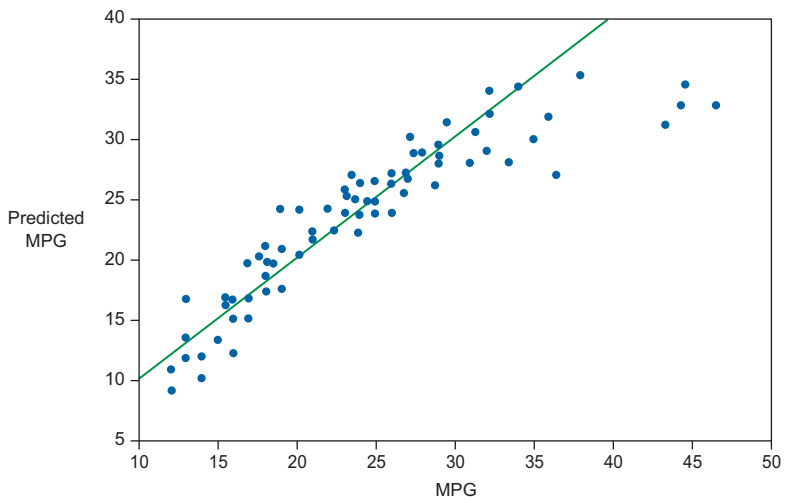


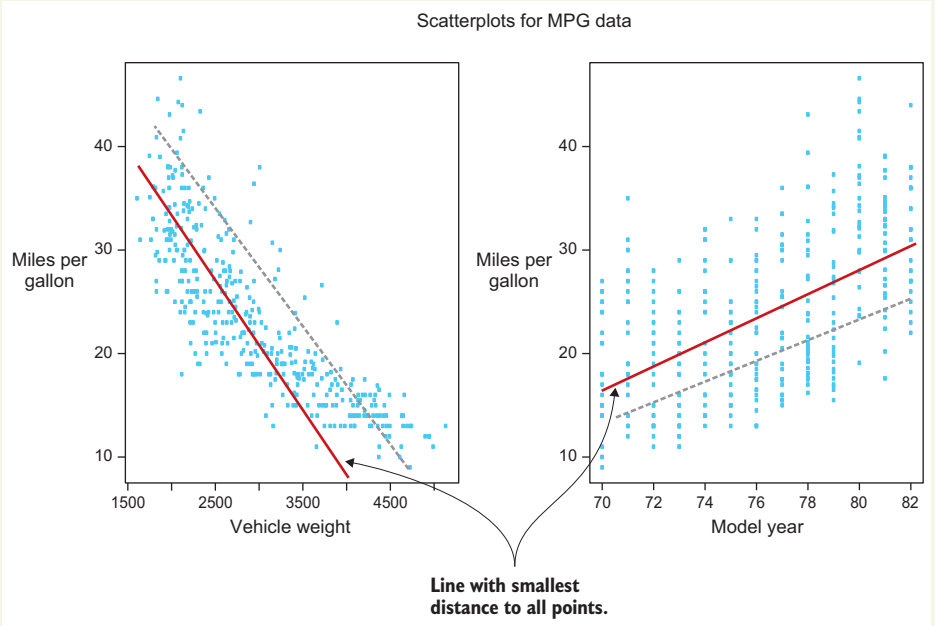
Figure 3.17 A scatter plot of the actual versus predicted values on the held-out test set. The diagonal line shows the perfect regressor. The closer all of the predictions are to this line, the better the model.

shows great prediction performance, as the predictions all fall close to the optimal diagonal line. By looking at this figure, you can get a sense of how your ML model might perform on new data. In this case, a few of the predictions for higher MPG values seem to be underestimated, and this may be useful information for you. For example, if you want to get better at estimating high MPG values, you might need to find more examples of high MPG vehicles, or you might need to obtain higher-quality data in this regime.

Algorithm highlight: linear regression

Like logistic regression for classification, *linear regression* is arguably the simplest and most widely used algorithm for building regression models. The main strengths are linear scalability and a high level of interpretability.

This algorithm plots the dataset records as points, with the target variable on the y-axis, and fits a straight line (or plane, in the case of two or more features) to these points. The following figure illustrates the process of optimizing the distance from the points to the straight line of the model.



Demonstration of how linear regression determines the best-fit line. Here, the dark line is the optimal linear regression fitted line on this dataset, yielding a smaller mean-squared deviation from the data to any other possible line (such as the dashed line shown).

A straight line can be described by two parameters for lines in two dimensions, and so on. You know this from the a and b in $y = a \times x + b$ from the basic math. These parameters are fitted to the data, and when optimized, they completely describe the model and can be used to make predictions on new data.

3.3.2 Performing regression on complex, nonlinear data

In some datasets, the relationship between features can't be fitted by a linear model, and algorithms such as linear regression may not be appropriate if accurate predictions are required. Other properties, such as scalability, may make lower accuracy a necessary trade-off. Also, there's no guarantee that a nonlinear algorithm will be more accurate, as you risk overfitting to the data. As an example of a nonlinear regression model, we introduce the random forest algorithm. Random forest is a popular method for highly nonlinear problems for which accuracy is important. As evident in the appendix, it's also easy to use, as it requires minimal preprocessing of data. In figures 3.18 and 3.19, you can see the results of making predictions on the Auto MPG test set via the random forest model.

Origin = 1	Origin = 3	Origin = 2	MPG	Predicted MPG
0	0	1	26.0	27.1684
1	0	0	23.8	23.4603
1	0	0	13.0	13.6590
1	0	0	17.0	16.8940
1	0	0	16.9	15.5060

Figure 3.18 Table of actual versus predicted MPG values for the nonlinear random forest regression model

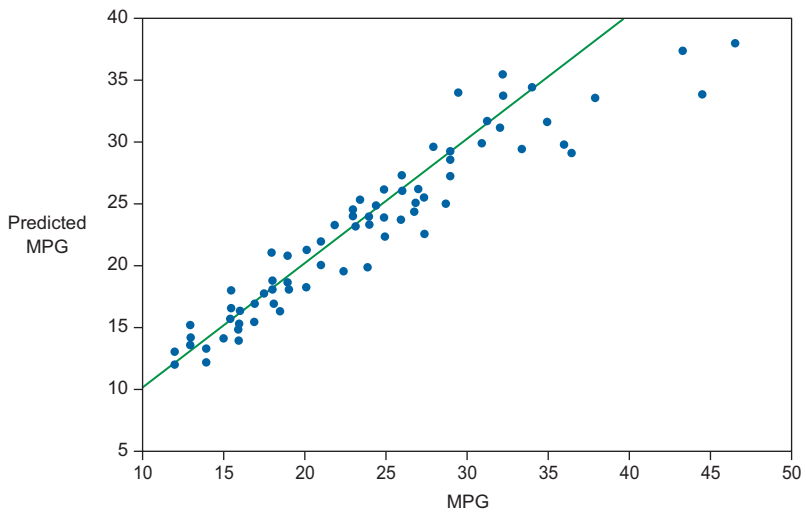


Figure 3.19 Comparison of MPG data versus predicted values for the nonlinear random forest regression model

This model isn't much different from the linear algorithm, at least visually. It's not clear which of the algorithms performs the best in terms of accuracy. In the next chapter, you'll learn how to quantify the performance (often called the *accuracy score* of the model) so you can make meaningful measurements of how good the prediction accuracy is.

Algorithm highlight: random forest

For the last algorithm highlight of this chapter, we introduce the *random forest* (RF) algorithm. This highly accurate nonlinear algorithm is widely used in real-world classification and regression problems.

The basis of the RF algorithm is the decision tree. Imagine that you need to make a decision about something, such as what to work on next. Some variables can help you decide the best course of action, and some variables weigh higher than others. In this case, you might ask first, "How much money will this make me?" If the answer is less than $10, you can choose to not go ahead with the task. If the answer is more than $10, you might ask the next question in the decision tree, "Will working on this make me happy?" and answer with a yes/no. You can continue to build out this tree until you've reached a conclusion and chosen a task to work on.

The decision tree algorithm lets the computer figure out, based on the training set, which variables are the most important, and put them in the top of the tree, and then gradually use less-important variables. This allows it to combine variables and say, "If the amount is greater than $10 and makes me happy, and amount of work less than 1 hour, then yes."

A problem with decision trees is that the top levels of the tree have a huge impact on the answer, and if the new data doesn't follow exactly the same distribution as the training set, the ability to generalize might suffer. This is where the random forest method comes in. By building a collection of decision trees, you mitigate this risk. When making the answer, you pick the majority vote in the case of classification, or take the mean in case of regression. Because you use votes or means, you can also give back full probabilities in a natural way that not many algorithms share.

Random forests are also known for other kinds of advantages, such as their immunity to unimportant features, noisy datasets in terms of missing values, and mislabeled records.

3.4 Summary

In this chapter, we introduced machine-learning modeling. Here we list the main takeaways from the chapter:

- The purpose of modeling is to describe the relationship between the input features and the target variable.
- You can use models either to generate predictions for new data (whose target is unknown) or to infer the true associations (or lack thereof) present in the data.

- There are hundreds of methods for ML modeling. Some are parametric, meaning that the form of the mathematical function relating the features to the target is fixed in advance. Parametric models tend to be more highly interpretable yet less accurate than nonparametric approaches, which are more flexible and can adapt to the true complexity of the relationship between the features and the target. Because of their high levels of predictive accuracy and their flexibility, nonparametric approaches are favored by most practitioners of machine learning.
- Machine-learning methods are further broken into supervised and unsupervised methods. Supervised methods require a training set with a known target, and unsupervised methods don't require a target variable. Most of this book is dedicated to supervised learning.
- The two most common problems in supervised learning are classification, in which the target is categorical, and regression, in which the target is numerical. In this chapter, you learned how to build both classification and regression models and how to employ them to make predictions on new data.
- You also dove more deeply into the problem of classification. Linear algorithms can define linear decision boundaries between classes, whereas nonlinear methods are required if the data can't be separated linearly. Using nonlinear models usually has a higher computational cost.
- In contrast to classification (in which a categorical target is predicted), you predict a numerical target variable in regression models. You saw examples of linear and nonlinear methods and how to visualize the predictions of these models.

3.5 Terms from this chapter

Word	Definition
model	The base product from using an ML algorithm on training data.
prediction	Predictions are performed by pulling new data through the model.
inference	The act of gaining insight into the data by building the model and not making predictions.
(non)parametric	Parametric models make assumptions about the structure of the data. Nonparametric models don't.
(un)supervised	Supervised models, such as classification and regression, find the mapping between the input features and the target variable. Unsupervised models are used to find patterns in the data without a specified target variable.
clustering	A form of unsupervised learning that puts data into self-defined clusters.
dimensionality reduction	Another form of unsupervised learning that can map high-dimensional datasets to a lower-dimensional representation, usually for plotting in two or three dimensions.
classification	A supervised learning method that predicts data into buckets.
regression	The supervised method that predicts numerical target values.

In the next chapter, you'll look at creating and testing models, the exciting part of machine learning. You'll see whether your choice of algorithms and features is going to work to solve the problem at hand. You'll also see how to rigorously validate a model to see how good its predictions are likely to be on new data. And you'll learn about validation methods, metrics, and some useful visualizations for assessing your models' performance.

index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

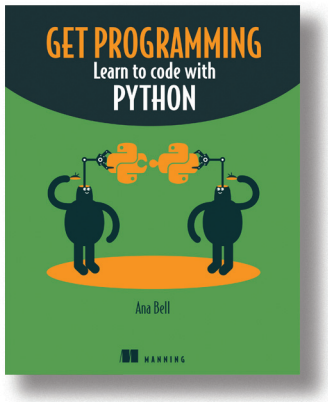
T

U

V

W

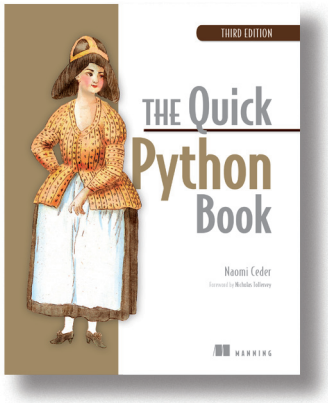
Z



Get Programming

By Ana Bell

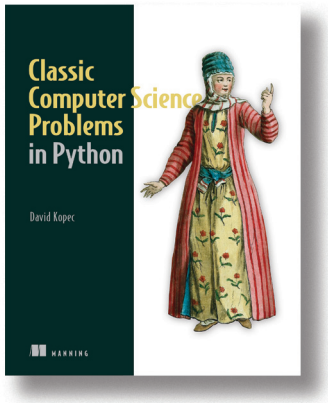
ISBN 9781617293788
456 pages
$34.99
March 2018



The Quick Python Book, Third Edition

By Naomi Ceder

ISBN 9781617294037
472 pages
$39.99
May 2018



Classic Computer Science Problems in Python

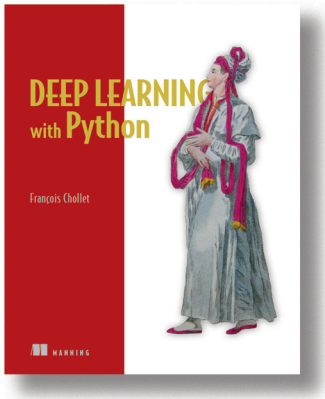
By David Kopec

ISBN 9781617295980
275 pages
$39.99
January 2019

Get Programming with Python in Motion

By Ana Bell

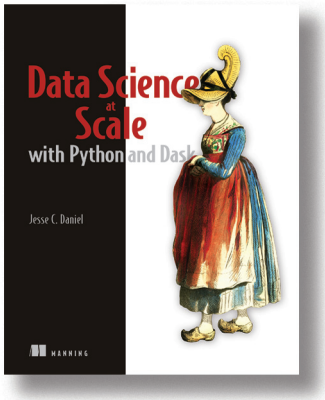
Course duration: 4h
$27.99
October 2018



Deep Learning with Python

By François Chollet

ISBN 9781617294433
384 pages
$49.99
November 2017



Data Science at Scale with Python and Dask

By Jesse C. Daniel

ISBN 9781617295607
400 pages
$49.99
Spring 2019