CHAPTER 5

Python Basics

5.0 Introduction

Although there are many languages that can be used to program the Raspberry Pi, Python is the most popular. In fact, the *Pi* in *Raspberry Pi* is inspired by the word Python.

In this chapter, you will find a host of recipes to help you get programming with Raspberry Pi.

5.1 Deciding Between Python 2 and Python 3

Problem

You need to use Python but are unsure about which version to use.

Solution

Use both. Use Python 3 until you face a problem that is best solved by reverting to version 2.

Discussion

Although the latest version of Python is Python 3 and has been for years, you will find that a lot of people stick to Python 2. In fact, in the Raspbian distribution, both versions are supplied and version 2 is just called Python, whereas version 3 is called Python 3. You even run Python 3 by using the command `python3`. The examples in this book are written for Python 3 unless otherwise stated. Most will run on both Python 2 and Python 3 without modification.

This reluctance on the part of the Python community to ditch the old Python 2 is largely because Python 3 introduced some changes that broke compatibility with version 2. This means that some of the huge body of third-party libraries that have been developed for Python 2 won't work under Python 3.

My strategy is to write in Python 3 wherever possible, and revert to Python 2 when I have to because of compatibility problems.

See Also

For a good summary of the Python 2 versus Python 3 debate, see the Python wiki.

5.2 Editing Python Programs with IDLE

Problem

You are not sure what to use to write your Python programs.

Solution

The common Raspberry Pi distributions come with the IDLE Python development tool in both the Python 2 and Python 3 versions. If you are using Raspbian, you will find shortcuts to both versions of IDLE labeled Python 2 and Python 3 in the Programming section of your Start menu (Figure 5-1).

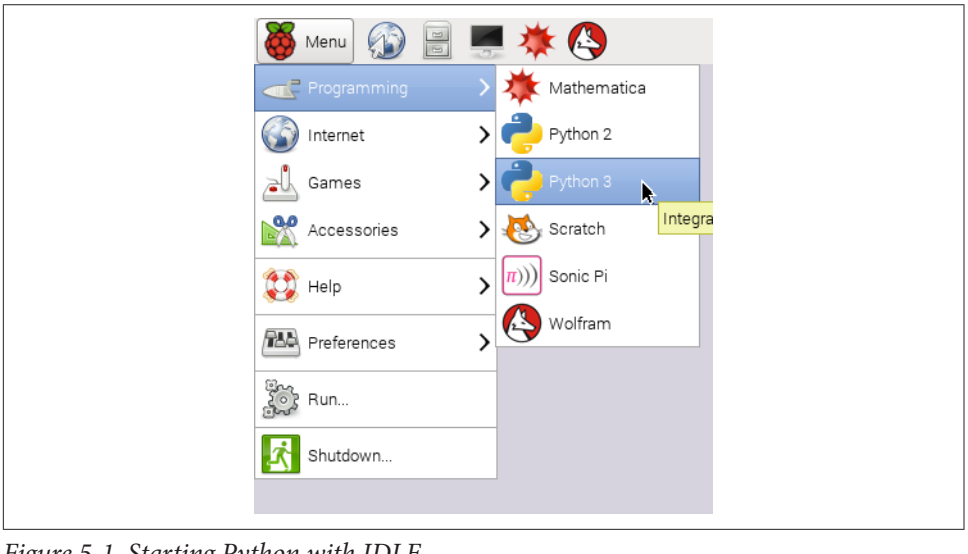

Figure 5-1. Starting Python with IDLE

Discussion

IDLE and IDLE 3 appear to be identical; the only difference is the version of Python that they use, so open IDLE 3 (Figure 5-2).

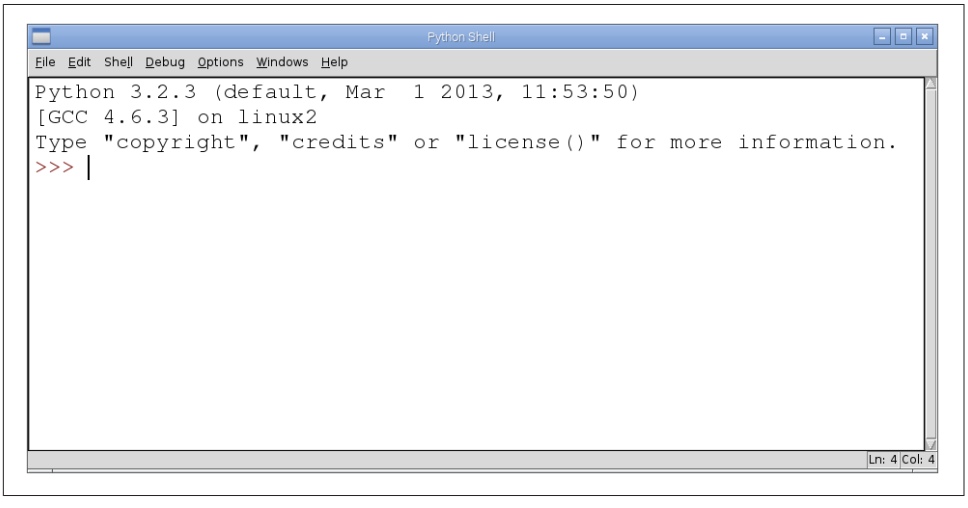

Figure 5-2. IDLE Python console

The window that opens is labeled Python Shell. This is an interactive area where you can type **`Python`** and see the results immediately. So try typing the following into the shell next to the >>> prompt:

```
>>> 2 + 2
4
>>>
```

Python has evaluated the expression `2 + 2` and given us the answer of `4`.

The Python Shell is fine for trying things out, but if you want to write programs, then you need to use an editor. To open a new file to edit using IDLE, select New Window from the File menu (Figure 5-3).

You can now enter your program into the editor window. Just to try things, paste the following text into the editor, save the file as *count.py*, and then select Run Module from the Run menu. You can see the results of the program running in the Python console:

```
for i in range(1, 10):
    print(i)
```

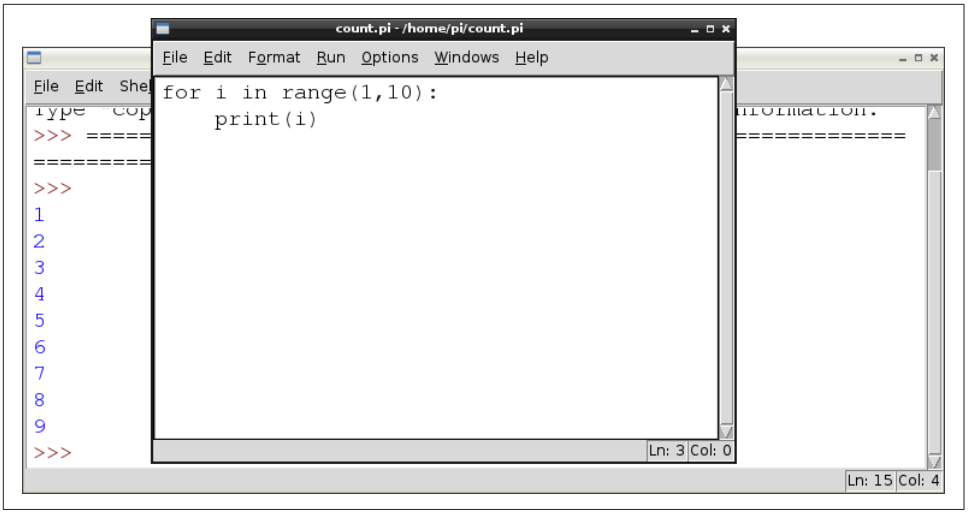

Figure 5-3. IDLE editor

Python is unusual for a programming language in that indentation is a fundamental part of the language. Whereas many C-based languages use { and } to delimit a block of code, Python uses the indentation level. So in the preceding example, Python knows that `print` is to be invoked repeatedly as part of the `for` loop because it is indented out four spaces.

The convention used in this book is to use four spaces for a level of indentation.

When you're starting out in Python, it's not uncommon to see an error such as `IndentationError: unexpected indent`, which means that somewhere things are not indented correctly. If everything appears to line up, double-check that none of the indents contain Tab characters. Python treats tabs differently.

Notice how IDLE uses color-coding to highlight the structure of the program.

See Also

Many of the recipes in this chapter use IDLE to edit Python examples.

As well as using IDLE to edit and run Python files, you can also edit files in nano (Recipe 3.6) and then run them from a Terminal session (Recipe 5.4).

5.3 Using the Python Console

Problem

You want to enter Python commands without writing a whole program.

Solution

Use the Python console, either within IDLE (Recipe 5.2) or in a Terminal session.

Discussion

To start a Python 2 console in a Terminal window, just type the command **`python`**; for a Python 3 console, enter the command **`python3`**.

The `>>>` prompt indicates that you can type Python commands. If you need to type multiline commands, then the console will automatically provide a continuation line indicated by three dots. You still need to indent any such lines, by four spaces, as shown in the following session:

```
>>> for i in range(1, 10):
...   print(i)
...
1
2
3
4
5
6
7
8
9
>>>
```

You will need to press Enter twice after your last command for the console to recognize the end of the indented block and run the code.

The Python console also provides a command history so that you can move back and forth through your previous commands using the up and down keys.

See Also

If you have more than a couple of lines that you want to type in, then chances are you would be better off using IDLE (Recipe 5.2) to edit and run a file.

5.4 Running Python Programs from the Terminal

Problem

Running programs from within IDLE is fine, but sometimes you want to run a Python program from a Terminal window.

Solution

Use the `python` or `python3` command in a Terminal, followed by the filename containing the program you want to run.

Discussion

To run a Python 2 program from the command line, use a command like this:

```
$ python myprogram.py
```

If you want to run the program using Python 3, then change the command `python` to `python3`. The Python program that you wish to run should be in a file with the extension *.py*.

You can run most Python programs as a normal user; however, there are some, especially those that use the GPIO port, that you need to run as superuser. If this is the case for your program, prefix the command with `sudo`:

```
$ sudo python myprogram.py
```

In the earlier examples, you have to include `python` in the command to run the program, but you can also add a line to the start of a Python program so that Linux knows it is a Python program. This special line is called a *shebang* (a contraction of *hash-bang*) and the following single-line example program has it as its first line.

```
#!/usr/bin/python
print("I'm a program, I can run all by myself")
```

Before you can run this directly from the command line, you have to give the file write permissions by using the command below (see Recipe 3.13).

```
$ chmod +x test.py
```

The parameter `+x` means add execute permission.

Now you will be able to run the Python program *test.py* by using the single command:

```
$ ./test.py
I'm a program, I can run all by myself
$
```

The ./ at the start of the line is needed for the command line to find the file.

See Also

Recipe 3.23 allows you to run a Python program as a timed event.

To automatically run a program at startup, see Recipe 3.22.

5.5 Variables

Problem

You want to give a value a name.

Solution

Assign a value to a name using =.

Discussion

In Python, you don't have to declare the type of a variable, you can just assign it a value, as shown in the following examples:

```
a = 123
b = 12.34
c = "Hello"
d = 'Hello'
e = True
```

You can define character-string constants using either single or double quotes. The logical constants in Python are `True` and `False`, and are case-sensitive.

By convention, variable names begin with a lowercase letter and if the variable name consists of more than one word, the words are joined together with an underscore character. It is always a good idea to give your variables descriptive names.

Some examples of valid variable names are `x`, `total`, and `number_of_chars`.

See Also

Variables can also be assigned a value that is a list (Recipe 6.1) or dictionary (Recipe 6.12).

For more information on arithmetic with variables, see Recipe 5.8.

5.6 Displaying Output

Problem

You want to see the value of a variable.

Solution

Use the `print` command. You can try the following example in the Python console (Recipe 5.3):

```
>>> x = 10
>>> print(x)
10
>>>
```

Discussion

In Python 2, you can use the `print` command without brackets. However, this is not true in Python 3, so for compatibility with both versions of Python, use brackets around the value you are printing.

See Also

To read user input, see Recipe 5.7.

5.7 Reading User Input

Problem

You want to prompt the user to enter a value.

Solution

Use the `input` (Python 3) or `raw_input` (Python 2) command. You can try the following example in the Python 3 console (Recipe 5.3):

```
>>> x = input("Enter Value:")
Enter Value:23
>>> print(x)
23
>>>
```

Discussion

In Python 2, `raw_input` must be substituted for `input` in the preceding example.

Python 2 also has an `input` function, but this validates the input and attempts to convert it into a Python value of the appropriate type, whereas `raw_input` does the same thing as `input` in Python 3 and just returns a string.

See Also

Find more information on Python 2 `input`.

5.8 Arithmetic

Problem

You want to do arithmetic in Python.

Solution

Use the +, -, *, and / operators.

Discussion

The most common operators for arithmetic are +, -, *, and /, which are, respectively, add, subtract, multiply, and divide.

You can also group parts of the expression together with parentheses, as shown in the following example, which, given a temperature in degrees Celsius, converts it to degrees Fahrenheit:

```
>>> tempC = input("Enter temp in C: ")
Enter temp in C: 20
>>> tempF = (int(tempC) * 9) / 5 + 32
>>> print(tempF)
68.0
>>>
```

Other arithmetic operators include % (modulo remainder) and ** (raise to the power of). For example, to raise 2 to the power of 8, you would write the following:

```
>>> 2 ** 8
256
```

See Also

See Recipe 5.7 on using the `input` command, and Recipe 5.12 on converting the string value from `input` to a number.

The `Math` library has many useful math functions that you can use.

5.9 Creating Strings

Problem

You want to create a string variable.

Solution

Use the assignment operator and a string constant to create a new string. You can use either double or single quotation marks around the string, but they must match.

For example:

```
>>> s = "abc def"
>>> print(s)
abc def
>>>
```

Discussion

If you need to include double or single quotes inside a string, then pick the type of quotes that you don't want to use inside the string as the beginning and end markers of the string. For example:

```
>>> s = "Isn't it warm?"
>>> print(s)
Isn't it warm?
>>>
```

Sometimes you'll need to include special characters such as tab or newline inside your string. This requires the use of what are called *escape characters*. To include a tab, use `\t`, and for a newline, use `\n`. For example:

```
>>> s = "name\tage\nMatt\t14"
>>> print(s)
name    age
Matt    14
>>>
```

See Also

For a full list of escape characters, see the Python Reference Manual.

5.10 Concatenating (Joining) Strings

Problem

You want to join a number of strings together.

Solution

Use the + (concatenate) operator.

For example:

```
>>> s1 = "abc"
>>> s2 = "def"
>>> s = s1 + s2
>>> print(s)
abcdef
>>>
```

Discussion

In many languages, you can have a chain of values to concatenate, some of which are strings and some of which are other types such as numbers; numbers will automatically be converted into strings during the concatenation. This is not the case in Python, and if you try the following command, you will get an error:

```
>>> "abc" + 23
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
```

You need to convert each component that you want to concatenate to a string before concatenating, as shown in this example:

```
>>> "abc" + str(23)
'abc23'
>>>
```

See Also

See Recipe 5.11 for more information about converting numbers to strings using the `str` function.

5.11 Converting Numbers to Strings

Problem

You want to convert a number into a string.

Solution

Use the `str` Python function. For example:

```
>>> str(123)
'123'
>>>
```

Discussion

A common reason for wanting to convert a number into a string is so you can then concatenate it with another string (Recipe 5.10).

See Also

For the reverse operation of turning a string into a number, see Recipe 5.12.

5.12 Converting Strings to Numbers

Problem

You want to convert a string into a number.

Solution

Use the `int` or `float` Python function.

For example, to convert `-123` into a number, you could use:

```
>>> int("-123")
-123
>>>
```

This will work on both positive and negative whole numbers.

To convert a floating-point number, use `float` instead of `int`:

```
>>> float("00123.45")
123.45
>>>
```

Discussion

Both `int` and `float` will handle leading zeros correctly and are tolerant of any spaces or other whitespace characters around the number.

You can also use `int` to convert a string representing a number in a number base other than the default of 10 by supplying the number base as the second argument. The following example converts the string representation of binary 1001 into a number:

```
>>> int("1001", 2)
9
>>>
```

This second example converts the hexadecimal number AFF0 to an integer:

```
>>> int("AFF0", 16)
45040
>>>
```

See Also

For the reverse operation of turning a number into a string, see Recipe 5.11.

5.13 Finding the Length of a String

Problem

You need to know how many characters there are in a string.

Solution

Use the `len` Python function.

Discussion

For example, to find the length of the string `abcdef`, you would use:

```
>>> len("abcdef")
6
>>>
```

See Also

The `len` command also works on lists (Recipe 6.3).

5.14 Finding the Position of One String Inside Another

Problem

You need to find the position of one string within another.

Solution

Use the `find` Python function.

For example, to find the starting position of the string `def` within the string `abcdef ghi`, you would use:

```
>>> s = "abcdefghi"
>>> s.find("def")
3
>>>
```

Note that the character positions start at 0, so a position of 3 means the fourth character in the string.

Discussion

If the string you're looking for doesn't exist in the string being searched, then `find` returns the value -1.

See Also

The `replace` function is used to find and then replace all occurrences of a string (Recipe 5.16).

5.15 Extracting Part of a String

Problem

You want to cut out a section of a string between certain character positions.

Solution

Use the Python `[:]` notation.

For example, to cut out a section from the second character to the fifth character of the string `abcdefghi`, you would use:

```
>>> s = "abcdefghi"
>>> s[1:5]
'bcde'
>>>
```

Note that the character positions start at 0, so a position of 1 means the second character in the string, and 5 means the sixth, but the character range is exclusive at the high end so the letter *f* is not included in this example.

Discussion

The `[:]` notation is actually quite powerful. You can omit either argument, in which case, the start or end of the string is assumed as appropriate. For example:

```
>>> s[:5]
'abcde'
>>>
```

and

```
>>> s = "abcdefghi"
>>> s[3:]
'defghi'
>>>
```

You can also use negative indices to count back from the end of the string. This can be useful in situations such as when you want to find the three-letter extension of a file, as in the following example:

```
>>> "myfile.txt"[-3:]
'txt'
```

See Also

Recipe 5.10 describes joining strings together rather than splitting them.

Recipe 6.10 uses the same syntax but with lists rather than strings.

5.16 Replacing One String of Characters with Another Inside a String

Problem

You want to replace all occurrences of a string within another string.

Solution

Use the `replace` function.

For example, to replace all occurrences of `X` with `times`, you would use:

```
>>> s = "It was the best of X. It was the worst of X"
>>> s.replace("X", "times")
'It was the best of times. It was the worst of times'
>>>
```

Discussion

The string being searched for must match exactly; that is, the search is case-sensitive and will include spaces.

See Also

See Recipe 5.14 for searching a string without performing a replacement.

5.17 Converting a String to Upper- or Lowercase

Problem

You want to convert all the characters in a string to upper- or lowercase letters.

Solution

Use the `upper` or `lower` function as appropriate.

For example, to convert `aBcDe` to uppercase, you would use:

```
>>> "aBcDe".upper()
'ABCDE'
>>>
```

To convert it to lowercase, you would use:

```
>>> "aBcDe".lower()
'abcde'
>>>
```

Discussion

Like most functions that manipulate a string in some way, `upper` and `lower` do not actually modify the string but rather return a modified copy of the string.

For example, the following code returns a copy of the string `s`, but note how the original string is unchanged.

```
>>> s = "aBcDe"
>>> s.upper()
'ABCDE'
>>> s
'aBcDe'
>>>
```

If you want to change the value of `s` to be all uppercase, then you need to do the following:

```
>>> s = "aBcDe"
>>> s = s.upper()
>>> s
'ABCDE'
>>>
```

See Also

See Recipe 5.16 for replacing text within strings.

5.18 Running Commands Conditionally

Problem

You want to run some Python commands only when some condition is true.

Solution

Use the Python `if` command.

The following example will print the message `x is big` only if `x` has a value greater than 100:

```
>>> x = 101
>>> if x > 100:
...     print("x is big")
...
x is big
```

Discussion

After the `if` keyword, there is a *condition*. This condition often, but not always, compares two values and gives an answer that is either `True` or `False`. If it is `True`, then the subsequent indented lines will all be executed.

It is quite common to want to do one thing if a condition is `True` and something different if the answer is `False`. In this case, the `else` command is used with `if`, as shown in this example:

```
x = 101
if x > 100:
    print("x is big")
else:
    print("x is small")

print("This will always print")
```

You can also chain together a long series of `elif` conditions. If any one of the conditions succeeds, then that block of code is executed and none of the other conditions are tried. For example:

```
x = 90
if x > 100:
    print("x is big")
elif x < 10:
    print("x is small")
else:
    print("x is medium")
```

This example will print `x is medium`.

See Also

See Recipe 5.19 for more information on different types of comparisons you can make.

5.19 Comparing Values

Problem

You want to compare values.

Solution

Use one of the comparison operators: <, >, <=, >=, ==, or !=.

Discussion

You used the < (less than) and > (greater than) operators in Recipe 5.18. Here's the full set of comparison operators:

<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Exactly equal to
!=	Not equal to

Some people prefer to use the <> operator in place of !=. Both work the same.

You can test these commands using the Python console (Recipe 5.3), as shown in the following exchange:

```
>>> 1 != 2
True
>>> 1 != 1
False
>>> 10 >= 10
True
>>> 10 >= 11
False
>>> 10 == 10
True
>>>
```

A common mistake is to use = (set a value) instead of == (double equals) in comparisons. This can be difficult to spot because if one half of the comparison is a variable, it is perfectly legal syntax and will run, but it will not produce the result you were expecting.

As well as comparing numbers, you can also compare strings by using these comparison operators, as shown here:

```
>>> 'aa' < 'ab'
True
>>> 'aaa' < 'aa'
False
```

The strings are compared lexicographically—that is, in the order that you would find them in a dictionary.

This is not quite correct because, for each letter, the uppercase version of the letter is considered less than the lowercase equivalent. Each letter has a value that is its ASCII code.

See Also

See also Recipes 5.18 and 5.20.

5.20 Logical Operators

Problem

You need to specify a complex condition in an `if` statement.

Solution

Use one of the logical operators: `and`, `or`, and `not`.

Discussion

As an example, you may want to check whether a variable *x* has a value between 10 and 20. For that, you would use the `and` operator:

```
>>> x = 17
>>> if x >= 10 and x <= 20:
...     print('x is in the middle')
...
x is in the middle
```

You can combine as many `and` and `or` statements as you need and also use brackets to group them if the expressions become complicated.

See Also

See Recipes 5.18 and 5.19.

5.21 Repeating Instructions an Exact Number of Times

Problem

You need to repeat some program code an exact number of times.

Solution

Use the Python `for` command and iterate over a range.

For example, to repeat a command 10 times, use the following example:

```
>>> for i in range(1, 11):
...     print(i)
...
1
2
3
4
5
6
7
8
9
10
>>>
```

Discussion

The second parameter in the `range` command is exclusive; that is, to count up to 10, you must specify a value of 11.

See Also

If the condition for stopping the loop is more complicated than simply repeating a number of times, see Recipe 5.22.

If you are trying to repeat commands for each element of a list or dictionary, see Recipes 6.7 or 6.15, respectively.

5.22 Repeating Instructions Until Some Condition Changes

Problem

You need to repeat some program code until something changes.

Solution

Use the Python `while` statement. The `while` statement repeats its nested commands until its condition becomes false. The following example will stay in the loop until the user enters **`X`** for exit:

```
>>> answer = ''
>>> while answer != 'X':
...     answer = input('Enter command:')
...
Enter command:A
```

```
Enter command:B
Enter command:X
>>>
```

Discussion

Note that the preceding example uses the `input` command as it works in Python 3. To run the example in Python 2, substitute the command `raw_input` for `input`.

See Also

If you just want to repeat some commands a number of times, see Recipe 5.21.

If you are trying to repeat commands for each element of a list or dictionary, then see Recipes 6.7 or 6.15, respectively.

5.23 Breaking Out of a Loop

Problem

You are in a loop and need to exit the loop if some condition occurs.

Solution

Use the Python `break` statement to exit either a `while` or `for` loop.

The following example behaves in exactly the same way as the example code in Recipe 5.22:

```
>>> while True:
...     answer = input('Enter command:')
...     if answer == 'X':
...             break
...
Enter command:A
Enter command:B
Enter command:X
>>>
```

Discussion

Note that this example uses the `input` command as it works in Python 3. To run the example in Python 2, substitute the command `raw_input` for `input`.

This example behaves in exactly the same way as the example of Recipe 5.22. However, in this case, the condition for the `while` loop is just `True`, so the loop will never end unless we use `break` to exit the loop when the user enters `X`.

See Also

You can also leave a `while` loop by using its condition; see Recipe 5.18.

5.24 Defining a Function in Python

Problem

You want to avoid repeating the same code over and over in a program.

Solution

Create a function that groups together lines of code, allowing it to be called from multiple places.

Creating and then calling a function in Python is illustrated in the following example:

```
def count_to_10():
    for i in range(1, 11):
        print(i)

count_to_10()
```

In this example, we have defined a function using the `def` command that will print out the numbers between 1 and 10 whenever it is called:

```
count_to_10()
```

Discussion

The conventions for naming functions are the same as for variables in Recipe 5.5; that is, they should start with a lowercase letter, and if the name consists of more than one word, the words should be separated by underscores.

The example function is a little inflexible because it can only count to 10. If we wanted to make it more flexible—so it could count up to any number—then we can include the maximum number as a *parameter* to the function, as this example illustrates:

```
def count_to_n(n):
    for i in range(1, n + 1):
        print(i)

count_to_n(5)
```

The parameter name `n` is included inside the parentheses and then used inside the `range` command, but not before 1 is added to it.

Using a parameter for the number you want to count up to means that if you usually count to 10 but sometimes count to a different number, you will always have to specify the number. You can, however, specify a default value for a parameter, and hence have the best of both worlds, as shown in this example:

```
def count_to_n(n=10):
    for i in range(1, n + 1):
        print(i)

count_to_n()
```

This will now count to 10 unless a different number is specified when you call the function.

If your function needs more than one parameter, perhaps to count between two numbers, then the parameters are separated by commas:

```
def count(from_num=1, to_num=10):
    for i in range(from_num, to_num + 1):
                print(i)

count()
count(5)
count(5, 10)
```

All these examples are functions that do not return any value; they just do something. If you need a function to return a value, you need to use the `return` command.

The following function takes a string as an argument and adds the word *please* to the end of the string.

```
def make_polite(sentence):
    return sentence + " please"

print(make_polite("Pass the cheese"))
```

When a function returns a value, you can assign the result to a variable, or as in this example, print out the result.

See Also

To return more than one value from a function, see Recipe 7.3.

CHAPTER 6

Python Lists and Dictionaries

6.0 Introduction

In Chapter 5, you looked at the basics of the Python language. In this chapter, you look at two key Python data structures: lists and dictionaries.

6.1 Creating a List

Problem

You want to use a variable to hold a series of values rather than just one value.

Solution

Use a list. In Python, a list is a collection of values stored in order so that you can access them by position.

You create a list using [and] to contain its initial contents:

```
>>> a = [34, 'Fred', 12, False, 72.3]
>>>
```

Unlike the more rigid arrays in languages like C, you don't need to specify the size of a list in Python when you declare it. You can also change the number of elements in the list any time you like.

Discussion

As this example illustrates, the items in a list do not have to be all of the same type, although they often are.

If you want to create an empty list that you can add items to later, you can write:

```
>>> a = []
>>>
```

See Also

All the recipes between Recipes 6.1 and 6.11 involve the use of lists.

6.2 Accessing Elements of a List

Problem

You want to find individual elements of a list or change them.

Solution

Use the `[]` notation to access elements of a list by their position in the list. For example:

```
>>> a = [34, 'Fred', 12, False, 72.3]
>>> a[1]
'Fred'
```

Discussion

The list positions (indices) start at 0 for the first element.

As well as using the `[]` notation to read values out of a list, you can also use it to change values at a certain position. For example:

```
>>> a = [34, 'Fred', 12, False, 72.3]
>>> a[1] = 777
>>> a
[34, 777, 12, False, 72.3]
```

If you try to change (or, for that matter, read) an element of a list using an index that is too large, you will get an "Index out of range" error:

```
>>> a[50] = 777
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list assignment index out of range
>>>
```

See Also

All the recipes between Recipes 6.1 and 6.11 involve the use of lists.

6.3 Finding the Length of a List

Problem

You need to know how many elements there are in a list.

Solution

Use the `len` Python function. For example:

```
>>> a = [34, 'Fred', 12, False, 72.3]
>>> len(a)
5
```

Discussion

The `len` command also works on strings (Recipe 5.13).

See Also

All the recipes between Recipes 6.1 and 6.11 involve the use of lists.

6.4 Adding Elements to a List

Problem

You need to add an item to a list.

Solution

Use the `append`, `insert`, or `extend` Python functions.

To add a single item to the end of a list, use `append`:

```
>>> a = [34, 'Fred', 12, False, 72.3]
>>> a.append("new")
>>> a
[34, 'Fred', 12, False, 72.3, 'new']
```

Discussion

Sometimes you don't want to add the new elements to the end of a list, but rather you want to insert them at a certain position in the list. For this, use the `insert` command. The first argument is the index where the item should be inserted, and the second argument is the item to be inserted:

```
>>> a.insert(2, "new2")
>>> a
[34, 'Fred', 'new2', 12, False, 72.3]
```

Note how all the elements after the newly inserted element are shuffled up one position.

Both `append` and `insert` add only one element to a list. The `extend` function adds all the elements of one list to the end of another:

```
>>> a = [34, 'Fred', 12, False, 72.3]
>>> b = [74, 75]
>>> a.extend(b)
>>> a
[34, 'Fred', 12, False, 72.3, 74, 75]
```

See Also

All the recipes between Recipes 6.1 and 6.11 involve the use of lists.

6.5 Removing Elements from a List

Problem

You need to remove an item from a list.

Solution

Use the `pop` Python function.

The command `pop` with no parameters removes the last element of a list:

```
>>> a = [34, 'Fred', 12, False, 72.3]
>>> a.pop()
72.3
>>> a
[34, 'Fred', 12, False]
```

Discussion

Notice that `pop` returns the value removed from the list.

To remove an item in a position rather than the last element, use `pop` with a parameter of the position from which the item will be removed:

```
>>> a = [34, 'Fred', 12, False, 72.3]
>>> a.pop(0)
34
```

If you use an index position that is beyond the end of the list, you will get an "Index out of range" exception.

See Also

All the recipes between Recipes 6.1 and 6.11 involve the use of lists.

6.6 Creating a List by Parsing a String

Problem

You need to convert a string of words separated by some character into an array of strings, with each string in the array being one of the words.

Solution

Use the `split` Python string function.

The command `split` with no parameters separates the words out of a string into individual elements of an array:

```
>>> "abc def ghi".split()
['abc', 'def', 'ghi']
```

If you supply `split` with a parameter, then it will split the string, using the parameter as a separator. For example:

```
>>> "abc--de--ghi".split('--')
['abc', 'de', 'ghi']
```

Discussion

This command can be very useful when you are, say, importing data from a file. The `split` command can optionally take an argument that is the string to use as a delimiter when you are splitting the string. So, if you were to use commas as a separator, you could split the string as follows:

```
>>> "abc,def,ghi".split(',')
['abc', 'def', 'ghi']
```

See Also

All the recipes between Recipes 6.1 and 6.11 involve the use of lists.

6.7 Iterating Over a List

Problem

You need to apply some lines of code to each item in a list in turn.

Solution

Use the `for` Python language command:

```
>>> a = [34, 'Fred', 12, False, 72.3]
>>> for x in a:
...     print(x)
...
34
Fred
12
False
72.3
>>>
```

Discussion

The `for` keyword is immediately followed by a variable name (in this case, `x`). This is called the loop variable and will be set to each of the elements of the list specified after `in`.

The indented lines that follow will be executed one time for each element in the list. Each time around the loop, `x` will be given the value of the element in the list at that position. You can then use `x` to print out the value, as shown in the previous example.

See Also

All the recipes between Recipes 6.1 and 6.11 involve the use of lists.

6.8 Enumerating a List

Problem

You need to run some lines of code for each item in a list in turn, but you also need to know the index position of each item.

Solution

Use the `for` Python language command along with the `enumerate` command.

```
>>> a = [34, 'Fred', 12, False, 72.3]
>>> for (i, x) in enumerate(a):
...     print(i, x)
...
(0, 34)
(1, 'Fred')
(2, 12)
(3, False)
(4, 72.3)
>>>
```

Discussion

It's quite common to need to know the position of something in the list while enumerating each of the values. An alternative method is to simply count with an index variable and then access the value by using the [] syntax:

```
>>> a = [34, 'Fred', 12, False, 72.3]
>>> for i in range(len(a)):
...     print(i, a[i])
...
(0, 34)
(1, 'Fred')
(2, 12)
(3, False)
(4, 72.3)
>>>
```

See Also

All the recipes between Recipes 6.1 and 6.11 involve the use of lists.

See Recipe 6.7 to iterate over a list without needing to know each item's index position.

6.9 Sorting a List

Problem

You need to sort the elements of a list.

Solution

Use the `sort` Python language command:

```
>>> a = ["it", "was", "the", "best", "of", "times"]
>>> a.sort()
>>> a
['best', 'it', 'of', 'the', 'times', 'was']
```

Discussion

When you sort a list, you'll actually be modifying it rather than returning a sorted copy of the original list. This means that if you also need the original list, you need to use the `copy` command in the standard library to make a copy of the original list before sorting it:

```
>>> import copy
>>> a = ["it", "was", "the", "best", "of", "times"]
>>> b = copy.copy(a)
>>> b.sort()
>>> a
['it', 'was', 'the', 'best', 'of', 'times']
>>> b
['best', 'it', 'of', 'the', 'times', 'was']
>>>
```

See Also

All the recipes between Recipes 6.1 and 6.11 involve the use of lists.

6.10 Cutting Up a List

Problem

You need to make a sublist of a list, using a range of the original list's elements.

Solution

Use the `[:]` Python language construction. The following example returns a list containing the elements of the original list from index position 1 to index position 2 (the number after the `:` is exclusive):

```
>>> l = ["a", "b", "c", "d"]
>>> l[1:3]
['b', 'c']
```

Note that the character positions start at 0, so a position of 1 means the second character in the string, and 5 means the sixth, but the character range is exclusive at the high end, so the letter *d* is not included in this example.

Discussion

The [:] notation is actually quite powerful. You can omit either argument, in which case the start or end of the list is assumed as appropriate. For example:

```
>>> l = ["a", "b", "c", "d"]
>>> l[:3]
['a', 'b', 'c']
```

```
>>> l[3:]
['d']
>>>
```

You can also use negative indices to count back from the end of the list. The following example returns the last two elements in the list:

```
>>> l[-2:]
['c', 'd']
```

Incidentally, `l[:-2]` returns `['a', 'b']` in the preceding example.

See Also

All the recipes between Recipes 6.1 and 6.11 involve the use of lists.

See Recipe 5.15 where the same syntax is used for strings.

6.11 Applying a Function to a List

Problem

You need to apply a function to each element of a list and collect the results.

Solution

Use the Python language feature called *comprehensions*.

The following example converts each string element of the list to uppercase and returns a new list that is the same length as the old one, but with all the strings in uppercase:

```
>>> l = ["abc", "def", "ghi", "ijk"]
>>> [x.upper() for x in l]
['ABC', 'DEF', 'GHI', 'IJK']
```

Although it can get confusing, there is no reason why you can't combine these kinds of statements, nesting one comprehension inside another.

Discussion

This is a very concise way of doing comprehensions. The whole expression has to be enclosed in brackets ([]). The first element of the comprehension is the code to be evaluated for each element of the list. The rest of the comprehension looks rather like a normal list iteration command (Recipe 6.7). The loop variable follows the `for` keyword, and the list to be used follows the `in` keyword.

See Also

All the recipes between Recipes 6.1 and 6.11 involve the use of lists.

6.12 Creating a Dictionary

Problem

You need to create a lookup table where you associate values with keys.

Solution

Use a Python dictionary.

Arrays are great when you need to access a list of items in order or you always know the index of the element that you want to use. Dictionaries are an alternative to lists for storing collections of data, but they are organized very differently.

Figure 6-1 shows how a dictionary is organized.

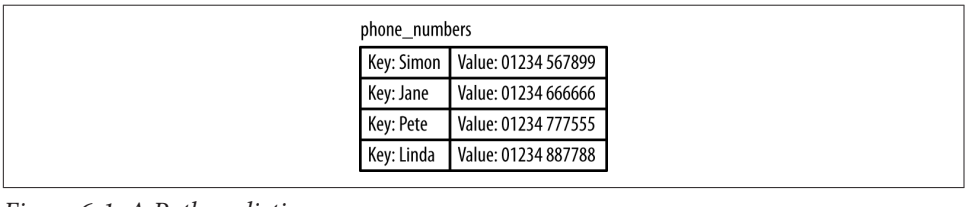

Figure 6-1. A Python dictionary

A dictionary stores key/value pairs in such a way that you can use the key to retrieve that value very efficiently and without having to search the whole dictionary.

To create a dictionary, you use the {} notation:

```
>>> phone_numbers = {'Simon':'01234 567899', 'Jane':'01234 666666'}
```

Discussion

In this example, the keys of the dictionary are strings, but they do not have to be; they could be numbers or in fact any data type, although strings are most commonly used.

The values can also be of any data type, including other dictionaries or lists. The following example creates one dictionary (`a`) and then uses it as a value in a second dictionary (`b`):

```
>>> a = {'key1':'value1', 'key2':2}
>>> a
{'key2': 2, 'key1': 'value1'}
>>> b = {'b_key1':a}
```

```
>>> b
{'b_key1': {'key2': 2, 'key1': 'value1'}}
```

When you display the contents of a dictionary, you will notice that the order of the items in the dictionary may not match the order in which they were specified when the dictionary was created and initialized with some contents:

```
>>> phone_numbers = {'Simon':'01234 567899', 'Jane':'01234 666666'}
>>> phone_numbers
{'Jane': '01234 666666', 'Simon': '01234 567899'}
```

Unlike lists, dictionaries have no concept of keeping items in order. Because of the way they are represented internally, the order of a dictionary's contents will be—for all intents and purposes—random.

The reason the order appears to be random is that the underlying data structure is a *hash table*. Hash tables use a *hashing function* to decide where to store the value; the hashing function calculates a numeric equivalent to any object.

You can find out more about hash tables at Wikipedia.

See Also

All the recipes between Recipes 6.12 and 6.15 involve the use of dictionaries.

6.13 Accessing a Dictionary

Problem

You need to find and change entries in a dictionary.

Solution

Use the Python `[]` notation. Use the key of the entry to which you need access inside the brackets:

```
>>> phone_numbers = {'Simon':'01234 567899', 'Jane':'01234 666666'}
>>> phone_numbers['Simon']
'01234 567899'
>>> phone_numbers['Jane']
'01234 666666'
```

Discussion

The lookup process is in one direction only, from key to value.

If you use a key that is not present in the dictionary, you will get a *key error*. For example:

```
{'b_key1': {'key2': 2, 'key1': 'value1'}}
>>> phone_numbers = {'Simon':'01234 567899', 'Jane':'01234 666666'}
>>> phone_numbers['Phil']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'Phil'
>>>
```

As well as using the `[]` notation to read values from a dictionary, you can also use it to add new values or overwrite existing ones.

The following example adds a new entry to the dictionary with a key of `Pete` and a value of `01234 777555`:

```
>>> phone_numbers['Pete'] = '01234 777555'
>>> phone_numbers['Pete']
'01234 777555'
```

If the key is not in use in the dictionary, a new entry is automatically added. If the key is already present, then whatever value was there before will be overwritten by the new value.

This is in contrast to trying to read a value from the dictionary, where an unknown key will cause an error.

See Also

All the recipes between Recipes 6.12 and 6.15 involve the use of dictionaries.

For information on handling errors, see Recipe 7.10.

6.14 Removing Things from a Dictionary

Problem

You need to remove an item from a dictionary.

Solution

Use the `pop` command, specifying the key for the item that you want to remove:

```
>>> phone_numbers = {'Simon':'01234 567899', 'Jane':'01234 666666'}
>>> phone_numbers.pop('Jane')
'01234 666666'
>>> phone_numbers
{'Simon': '01234 567899'}
```

Discussion

The pop command returns the value of the item removed from the dictionary.

See Also

All the recipes between Recipes 6.12 and 6.15 involve the use of dictionaries.

6.15 Iterating Over Dictionaries

Problem

You need to do something to each of the items in the dictionary in turn.

Solution

Use the for command to iterate over the keys of the dictionary:

```
>>> phone_numbers = {'Simon':'01234 567899', 'Jane':'01234 666666'}
>>> for name in phone_numbers:
...     print(name)
...
Jane
Simon
```

Discussion

There are a couple of other techniques that you can use to iterate over a dictionary. The following form can be useful if you need access to the values as well as the keys:

```
>>> phone_numbers = {'Simon':'01234 567899', 'Jane':'01234 666666'}
>>> for name, num in phone_numbers.items():
...     print(name + " " + num)
...
Jane 01234 666666
Simon 01234 567899
```

See Also

All the recipes between Recipes 6.12 and 6.15 involve the use of dictionaries.

See the for command used elsewhere in Recipes 5.21, 6.7, and 6.11.

CHAPTER 7

Advanced Python

7.0 Introduction

In this chapter, you will explore some of the more advanced concepts in the Python language—in particular, object-oriented Python, reading and writing files, handling exceptions, using modules, and Internet programming.

7.1 Formatting Numbers

Problem

You want to format numbers to a certain number of decimal places.

Solution

Apply a `format` string to the number. For example:

```
>>> x = 1.2345678
>>> "x={:.2f}".format(x)
'x=1.23'
>>>
```

Discussion

The formatting string can contain a mixture of regular text and markers delimited by `{` and `}`. The parameters to the format function (there can be as many as you like) will be substituted in place of the marker, according to the format specifier.

In the preceding example, the format specifier is `:.2f`, which means that the number will be specified with two digits after the decimal place and is a float `f`.

If you wanted the number to be formatted so that the total length of the number is always seven digits (or padding spaces), then you would add another number before the decimal place like this:

```
>>> "x={:7.2f}".format(x)
'x=   1.23'
>>>
```

In this case, since the number is only three digits long, there are four spaces of padding before the 1. If you wanted the padding to take the form of leading zeros, you would use:

```
>>> "x={:07.2f}".format(x)
'x=0001.23'
>>>
```

A more complicated example might be to display the temperature in both degrees Celsius and degrees Fahrenheit, as shown here:

```
>>> c = 20.5
>>> "Temperature {:5.2f} deg C, {:5.2f} deg F.".format(c, c * 9 / 5 + 32)
'Temperature 20.50 deg C, 68.90 deg F.'
>>>
```

See Also

Formatting in Python involves a whole formatting language.

7.2 Formatting Dates and Times

Problem

You want to convert a date into a string and format it in a certain way.

Solution

Apply `format` string to the date object.

For example:

```
>>> from datetime import datetime
>>> d = datetime.now()
>>> "{:%Y-%m-%d %H:%M:%S}".format(d)
'2015-12-09 16:00:45'
>>>
```

Discussion

The Python formatting language includes some special symbols for formatting date. `%Y`, `%m`, and `%d` correspond to year, month, and day numbers, respectively.

Other symbols useful for formatting the date are `%B`, which supplies the full name of the month and `%Y`, which gives the year in four digit format, as shown here:

```
>>> "{:%d %B %Y}".format(d)
'09 December 2015'
```

See Also

See Recipe 7.1 for formatting of numbers.

Formatting in Python actually involves a whole formatting language. You will find full details of this at *http://strftime.org/*.

7.3 Returning More Than One Value

Problem

You need to write a function that returns more than one value.

Solution

Return a Python *tuple* and use the multiple variable assignment syntax. A tuple is a Python data structure that's a little like a list, except that tuples are enclosed in parentheses rather than brackets. They are also of fixed size.

For example, you could have a function that converts a temperature in Kelvin into both Fahrenheit and Celsius. You can arrange for this function to return the temperature in both these units by separating the multiple return values by commas:

```
>>> def calculate_temperatures(kelvin):
...     celsius = kelvin - 273
...     fahrenheit = celsius * 9 / 5 + 32
...     return celsius, fahrenheit
...
>>> c, f = calculate_temperatures(340)
>>>
>>> print(c)
67
>>> print(f)
152.6
```

When you call the function, you just provide the same number of variables before the = and each of the return values will be assigned to the variables in the same position.

Discussion

Sometimes when you just have one or two values to return, this is the right way to return multiple values. However, if the data is complex, you might find that the solu-

tion is neater if you use Python's object-oriented features and define a class that contains the data. That way, you can return an instance of the class rather than a tuple.

See Also

See Recipe 7.4 for information on defining classes.

7.4 Defining a Class

Problem

You need to group together related data and functionality into a class.

Solution

Define a class and provide it with the member variables you need.

The following example defines a class to represent an address book entry:

```
class Person:
    '''This class represents a person object'''

    def __init__(self, name, tel):
        self.name = name
        self.tel = tel
```

The first line inside the class definition uses the triple quotes to denote a documentation string. This should explain the purpose of the class. Although entirely optional, adding a documentation string to a class allows others to see what the class does. This is particularly useful if the class is made available for others to use.

Doc strings are not like normal comments because, although they are not active lines of code, they do get associated with the class, so, at any time, you can read the doc string for a class by using the following command:

```
Person.__doc__
```

Inside the class definition is the constructor method, which will be called automatically whenever you create a new instance of the class. A class is like a template, so in defining a class called `Person`, we do not create any actual `Person` objects until later:

```
def __init__(self, name, tel):
    self.name = name
    self.tel = tel
```

The constructor method must be named as shown with double underscores on either side of the word `init`.

Discussion

One way in which Python differs from most object-oriented languages is that you have to include the special variable `self` as a parameter to all the methods that you define within the class. This is a reference to, in this case, the newly created instance. The variable `self` is the same concept as the special variable `this` that you find in Java and some other languages.

The code in this method transfers parameters that were supplied to it into member variables. The member variables do not need to be declared in advance, but do need to be prefixed by `self.`.

So the following line:

```
self.name = name
```

creates a variable called `name` that's accessible to every member of the class `Person` and initializes it with the value passed into the call to create an instance, which looks like this:

```
p = Person("Simon", "1234567")
```

We can then check that our new `Person` object, `p`, has a name of `"Simon"` by typing the following:

```
>>> p.name
Simon
```

In a complex program, it is good practice to put each class in its own file with a filename that matches the class name. This also makes it easy to convert the class into a module (see Recipe 7.11).

See Also

See Recipe 7.5 for information on defining methods.

7.5 Defining a Method

Problem

You need to add a method to a class.

Solution

The following example shows how you can include a method within a class definition:

```
class Person:
    '''This class represents a person object'''
```

```
    def __init__(self, first_name, surname, tel):
        self.first_name = first_name
        self.surname = surname
        self.tel = tel

    def full_name(self):
        return self.first_name + " " + self.surname
```

The `full_name` method concatenates the first name and surname attributes of the person, with a space in between.

Discussion

You can think of methods as just functions that are tied to a specific class and may or may not use member variables of that class in their processing. So, as with a function, you can write whatever code you like in a method and also have one method call another.

If one method calls another within the same class, the call to the method has to be prefixed with `self.`

See Also

See Recipe 7.4 for information on defining a class.

7.6 Inheritance

Problem

You need a specialized version of an existing class.

Solution

Use *inheritance* to create a subclass of an existing class and add new member variables and methods.

By default, all new classes that you create are subclasses of `object`. You can change this by specifying the class you want to use as a superclass in parentheses after the class name in a class definition. The following example defines a class (`Employee`) as a subclass of `Person` and adds a new member variable (`salary`) and an extra method (`give_raise`):

```
class Employee(Person):

    def __init__(self, first_name, surname, tel, salary):
        super().__init__(first_name, surname, tel)
        self.salary = salary
```

```
    def give_raise(self, amount):
        self.salary = self.salary + amount
```

Note that the preceding example is for Python 3. For Python 2, you can't use `super` the same way. Instead, you must write:

```
class Employee(Person):

    def __init__(self, first_name, surname, tel, salary):
        Person.__init__(self, first_name, surname, tel)
        self.salary = salary

    def give_raise(self, amount):
        self.salary = self.salary + amount
```

Discussion

In both of these examples, the initializer method for the subclass first uses the initializer method of the parent class (superclass) and then adds the member variable. This has the advantage that you do not need to repeat the initialization code in the new subclass.

See Also

See Recipe 7.4 for information on defining a class.

The Python inheritance mechanism is actually very powerful and supports *multiple inheritance*, where a subclass inherits from more than one superclass. For more on multiple inheritance, see the official documentation for Python.

7.7 Writing to a File

Problem

You need to write something to a file.

Solution

Use the `open`, `write`, and `close` functions to open a file, write some data, and then close the file, respectively:

```
>>> f = open('test.txt', 'w')
>>> f.write('This file is not empty')
>>> f.close()
```

Discussion

Once you have opened the file, you can make as many writes to it as you like before you close the file. Note that it is important to use `close` because although each write should update the file immediately, it may be buffered in memory and data could be lost. It could also leave the file locked so that other programs can't open it.

The `open` method takes two parameters. The first is the path to the file to be written. This can be relative to the current working directory, or if it starts with a `/`, an absolute path.

The second parameter is the mode in which the file should be opened. To overwrite an existing file or create the file with the name specified if it doesn't exist, just use `w`. Table 7-1 shows the full list of file mode characters. You can combine these using `+`. So to open a file in read and binary mode, you would use:

```
>>> f = open('test.txt', 'r+b')
```

Table 7-1. File modes

Mode	Description
r	Read
w	Write
a	Append - to the end of an existing file rather than overwriting
b	Binary mode
t	Text mode (default)
+	A shortcut for r+w

Binary mode allows you to read or write binary streams of data, such as images, rather than text.

See Also

To read the contents of a file, see Recipe 7.8. For more information on handling exceptions, see Recipe 7.10.

7.8 Reading from a File

Problem

You need to read the contents of a file into a string variable.

Solution

To read a file's contents, you need to use the file methods `open`, `read`, and `close`. The following example reads the entire contents of the file into the variable `s`:

```
f = open('test.txt')
s = f.read()
f.close()
```

Discussion

You can also read text files one line at a time using the method `readline`.

The preceding example will throw an exception if the file does not exist or is not readable for some other reason. You can handle this by enclosing the code in a `try`/`except` construction like this:

```
try:
    f = open('test.txt')
    s = f.read()
    f.close()
except IOError:
    print("Cannot open the file")
```

See Also

To write things to a file and for a list of file open modes, see Recipe 7.7.

For more information on handling exceptions, see Recipe 7.10.

7.9 Pickling

Problem

You want to save the entire contents of a data structure to a file so that it can be read the next time the program is run.

Solution

Use the Python *pickling* feature to dump the data structure to file in a format that can be automatically read back into memory as an equivalent data structure later on.

The following example saves a complex list structure to file:

```
>>> import pickle
>>> mylist = ['some text', 123, [4, 5, True]]
>>> f = open('mylist.pickle', 'w')
>>> pickle.dump(mylist, f)
>>> f.close()
```

To *unpickle* the contents of the file into a new list, use the following:

```
>>> f = open('mylist.pickle')
>>> other_array = pickle.load(f)
>>> f.close()
>>> other_array
['some text', 123, [4, 5, True]]
```

Discussion

Pickling will work on pretty much any data structure you can throw at it. It does not need to be a list.

The file is saved in a text format that is sort of human-readable, but you will not normally need to look at or edit the text file.

See Also

To write things to a file and for a list of file open modes, see Recipe 7.7.

7.10 Handling Exceptions

Problem

If something goes wrong while a program is running, you want to catch the error or exception and display a more user-friendly error message.

Solution

Use Python's `try/except` construct.

The following example, taken from Recipe 7.8, catches any problems in opening a file:

```
try:
    f = open('test.txt')
    s = f.read()
    f.close()
except IOError:
    print("Cannot open the file")
```

Because you wrapped the potentially error-prone commands to open the file in a `try/except` construction, any error that occurs will be captured before it displays any error message allowing you to handle it in your own way. In this case, that means displaying the friendly message `"Cannot open the file"`.

Discussion

A common situation where runtime exceptions can occur, in addition to during file access, is when you are accessing a list and the index you are using is outside the bounds of the list. For example, this happens if you try to access the fifth (index 4) element of a three-element list:

```
>>> list = [1, 2, 3]
>>> list[4]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

Errors and exceptions are arranged in a hierarchy, and you can be as specific or general as you like when catching the exceptions.

`Exception` is pretty near the top of that tree and will catch almost any exception. You can also have separate `except` sections for catching different types of exception and handling, each in a different way. If you do not specify any exception class, all exceptions will be caught.

Python also allows you to have `else` and `finally` clauses in your error handling:

```
list = [1, 2, 3]
try:
    list[8]
except:
    print("out of range")
else:
    print("in range")
finally:
    print("always do this")
```

The `else` clause will be run if there is no exception, and the `finally` clause will be run whether there is an exception or not.

Whenever an exception occurs, you can get more information about it by using the exception object, which is available only if you use the `as` keyword, as shown in the following example:

```
>>> list = [1, 2, 3]
>>> try:
...     list[8]
... except Exception as e:
...     print("out of range")
...     print(e)
...
out of range
list index out of range
>>>
```

This allows you to handle the error in your own way, but also keep hold of the original error message.

See Also

Python documentation for Python exception class hierarchy.

7.11 Using Modules

Problem

You want to use a Python module in your program.

Solution

Use the `import` command:

```
import random
```

Discussion

There is a very large number of modules (sometimes called *libraries*) available for Python. Many are included with Python as part of the standard library and others can be downloaded and installed into Python.

Standard Python libraries include modules for random numbers, database access, various Internet protocols, object serialization, and many more.

One consequence of having so many modules is that there is the potential for conflict—for example, if two modules have a function of the same name. To avoid such conflicts, when importing a module, you can specify how much of the module is accessible.

For example, if you just use a command like this:

```
import random
```

there is no possibility of any conflicts because you will only be able to access any functions or variables in the module by prefixing them with `random.`. (Incidentally, you'll be meeting the `random` package in the next recipe.)

If, on the other hand, you use the following command, everything in the module will be accessible without your having to put anything in front of it. Unless you know what all the functions are in all the modules you are using, there is a much greater chance of conflict.

```
from random import *
```

In between these two extremes, you can explicitly specify the components of a module that you need within a program so that they can be conveniently used without any prefix.

For example:

```
>>> from random import randint
>>> print(randint(1,6))
2
>>>
```

A third option is to use the `as` keyword to provide a more convenient or meaningful name for the module when referencing it.

```
>>> import random as R
>>> R.randint(1, 6)
```

See Also

For a definitive list of the modules included with Python, see *http://docs.python.org/2/library/*.

7.12 Random Numbers

Problem

You need to generate a random number between a range of numbers.

Solution

Use the `random` library:

```
>>> import random
>>> random.randint(1, 6)
2
>>> random.randint(1, 6)
6
>>> random.randint(1, 6)
5
```

The number generated will be between the two arguments (inclusive)—in this case, simulating a gaming dice.

Discussion

The numbers generated are not truly random, but are what is known as a *pseudorandom number sequence*; that is, they are a long sequence of numbers that, when taken in a large enough quantity, show what statisticians call a *random distribution*. For games, this is perfectly good enough, but if you were generating lottery numbers, you

would need to look at special randomizing hardware. Computers are just plain bad at being random; it's not really in their nature.

A common use of random numbers is to select something at random from a list. You can do this by generating an index position and using that, but there is also a command in the `random` module specifically for this. Try the following example:

```
>>> import random
>>> random.choice(['a', 'b', 'c'])
'a'
>>> random.choice(['a', 'b', 'c'])
'b'
>>> random.choice(['a', 'b', 'c'])
'a'
```

See Also

See the official reference for the random package for more information on this.

7.13 Making Web Requests from Python

Problem

You need to read the contents of a web page into a string using Python.

Solution

Python has an extensive library for making HTTP requests.

The following Python 2 example reads the contents of the Google home page into the string `contents`:

```
import urllib2
contents = urllib2.urlopen("https://www.google.com/").read()
print(contents)
```

Discussion

If you are using Python 3 rather than Python 2, then you will need to change your example to:

```
import urllib.request
response = urllib.request.urlopen('http://python.org/')
contents = response.read()
print(contents)
```

Having read the HTML, you are then likely to want to search it and extract the parts of the text that you really want. For this, you will need to use the string manipulation functions (see Recipes 5.14 and 5.15).

See Also

For more Internet-related examples using Python, see Chapter 15.

7.14 Command-Line Arguments in Python

Problem

You want to run a Python program from the command line and pass it parameters.

Solution

Import `sys` and use its `argv` property, as shown in the following example. This returns an array, the first element of which is the name of the program. The other elements are any parameters (separated by spaces) that were typed on the command line after the program name.

```
import sys

for (i, value) in enumerate(sys.argv):
    print("arg: %d %s " % (i, value))
```

Running the program from the command line, with some parameters after it, results in the following output:

```
$ python cmd_line.py a b c
arg: 0 cmd_line.py
arg: 1 a
arg: 2 b
arg: 3 c
```

Discussion

Being able to specify command-line arguments can be useful for automating the running of Python programs, either at startup (Recipe 3.21) or on a timed basis (Recipe 3.23).

See Also

For basic information on running Python from the command line, see Recipe 5.4.

In printing out `argv`, we used list enumerations (Recipe 6.8).

7.15 Running Linux Commands from Python

Problem

You want to run a Linux command or program from your Python program.

Solution

Use the `system` command.

For example, to delete a file called *myfile.txt* in the directory from which you started Python, you could do the following:

```
import os
os.system("rm myfile.txt")
```

Discussion

Sometimes rather than just execute a command blindly as in the example above, you need to capture the response of the command. Let's say you wanted to use the `'hostname'` command to find the IP address (see Recipe 2.2) of the Raspberry Pi. In this case, you can use the `check_output` function in the `subprocess` library.

```
import subprocess
ip = subprocess.check_output(['hostname', '-I'])
```

In this case, the variable `ip` will contain the IP address of the Raspberry Pi. Unlike `system`, `check_output` requires the command itself and any parameters to be supplied as separate elements of an array.

See Also

For documentation on the OS library, see *http://www.pythonforbeginners.com/os/pythons-os-module*.

For more information on the `subprocess` library, see *https://docs.python.org/3/library/subprocess.html*.

In Recipe 14.4, you will find an example using subprocess to display the IP address, hostname, and time of your Raspberry Pi on an LCD screen.

7.16 Sending Email from Python

Problem

You want to send an email message from a Python program.

Solution

Python has a library for the Simple Mail Transfer Protocol (SMTP) that you can use to send emails:

```
import smtplib

GMAIL_USER = 'your_name@gmail.com'
GMAIL_PASS = 'your_password'
SMTP_SERVER = 'smtp.gmail.com'
SMTP_PORT = 587

def send_email(recipient, subject, text):
    smtpserver = smtplib.SMTP(SMTP_SERVER, SMTP_PORT)
    smtpserver.ehlo()
    smtpserver.starttls()
    smtpserver.ehlo
    smtpserver.login(GMAIL_USER, GMAIL_PASS)
    header = 'To:' + recipient + '\n' + 'From: ' + GMAIL_USER
    header = header + '\n' + 'Subject:' + subject + '\n'
    msg = header + '\n' + text + ' \n\n'
    smtpserver.sendmail(GMAIL_USER, recipient, msg)
    smtpserver.close()

send_email('destination_email_address', 'sub', 'this is text')
```

To use this example to send an email to an address of your choice, first change the variables `GMAIL_USER` and `GMAIL_PASS` to match your email credentials. If you are not using Gmail, then you will also need to change the values of the `SMTP_SERVER` and possibly `SMTP_PORT`.

You also need to change the destination email address in the last line.

Discussion

The `send_email` message simplifies the use of the `smtplib` library into a single function that you can reuse in your projects.

Being able to send emails from Python opens up all sorts of project opportunities. For example, you could use a sensor such as the PIR sensor to send an email when movement is detected.

See Also

For a similar example that uses the IFTTT web service to send emails, see Recipe 15.3.

For performing HTTP requests from the Raspberry Pi, see Recipe 7.13.

For more information on the `smtplib`, see *http://docs.python.org/2/library/smtplib.html*.

For a whole load of Internet-related recipes, see Chapter 15.

7.17 Writing a Simple Web Server in Python

Problem

You need to create a simple Python web server, but don't want to have to run a full web server stack.

Solution

Use the `bottle` Python library to run a pure Python web server that will respond to HTTP requests.

To install `bottle`, use the following command:

```
sudo apt-get install python-bottle
```

The following Python program (called *bottle_test* in the book resources) simply serves up a message displaying what time the Pi thinks it is. Figure 7-1 shows the page you see if you connect to the Raspberry Pi from a browser anywhere on your network:

```
from bottle import route, run, template
from datetime import datetime

@route('/')
def index(name='time'):
    dt = datetime.now()
    time = "{:%Y-%m-%d %H:%M:%S}".format(dt)
    return template('<b>Pi thinks the date/time is: {{t}}</b>', t=time)

run(host='0.0.0.0', port=80)
```

To start the program, you need to run it with superuser privileges:

```
$ sudo python bottle_test.py
```

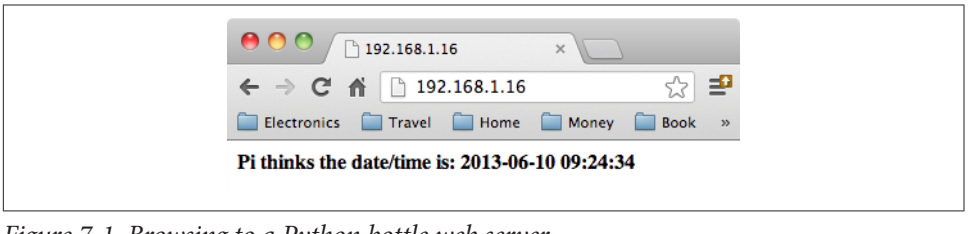

Figure 7-1. Browsing to a Python bottle web server

This example requires a little explanation.

After the `import` commands, the `@route` command links the URL path / with the handler function that follows it.

That handler function formats the date and time and then returns a string of the HTML to be rendered by the browser. In this case, it uses a template into which values can be substituted.

The final `run` line actually starts the web serving process. Port 80 is the default port for web serving, so if you wish to use a different port, then you need to add the port number with a `:` after the server address.

Discussion

You can define as many routes and handlers as you like within the program.

`bottle` is perfect for small, simple web server projects, and because it's written in Python, it's very easy to write a handler function to control hardware in response to the user interacting with the page in a browser. You will find other examples using `bottle` in Chapter 15.

See Also

For more information, see the `bottle` project documentation.

For more on formatting dates and times in Python, see Recipe 7.2.

For a whole load of Internet-related recipes, see Chapter 15.

7.18 Doing More Than One Thing at a Time

Problem

Your Python program is busy doing one thing, and you want it to do something else at the same time.

Solution

Use the Python `thread` library.

The following example, which you can download from the Downloads section of the *Raspberry Pi Cookbook* website, sets a thread running that will interrupt the counting of the main thread.

```
import thread, time, random

def annoy(message):
    while True:
        time.sleep(random.randint(1, 3))
        print(message)

thread.start_new_thread(annoy, ('BOO !!',))

x = 0
while True:
    print(x)
    x += 1
    time.sleep(1)
```

The output on the console will look something like this:

```
$ python thread_test.py
0
1
BOO !!
2
BOO !!
3
4
5
BOO !!
6
7
8
```

When you start a new *thread of execution* using the Python thread library, you must specify a function that is to be run as that thread. In this example, the function is called `annoy` and it contains a loop that will continue indefinitely printing out a message after a random interval of between 1 and 3 seconds.

To start the thread, the `start_new_thread` function in the thread module is called. This function expects two parameters: the first is the name of the function to run (in this case `annoy`) and the second is a tuple that contains any parameters that are to be passed to the function (in this case `'BOO !!'`).

You can see that the main thread that is just happily counting will be interrupted every few seconds by the thread running in the `annoy` function.

Discussion

Threads like these are also sometimes called *lightweight processes* because they are similar in effect to having more than one program or process running at the same time. They do, however, have the advantage that threads running in the same pro-

gram have access to the same variables, and when the main thread of the program exits, so do any threads that are started in it.

See Also

For a good introduction to threading in Python, see *http://www.tutorialspoint.com/python/python_multithreading.htm*.

7.19 Doing Nothing in Python

Problem

You want Python to kill time for a while. You might want to do this, say, to delay between sending messages to the terminal.

Solution

Use the `sleep` function in the `time` library as illustrated in the following code example that you can find with the downloads for the book called *sleep_test.py*.

```
import time

x = 0
while True:
    print(x)
    time.sleep(1)
    x += 1
```

The main loop of the program will delay for a second before printing the next number.

Discussion

The function `time.sleep` takes a number of seconds as its parameter. But if you want shorter delays, you can specify decimals. For example, to delay for a millisecond, you would use `time.sleep(0.001)`.

Its a good idea to put a short delay in any loop that continues indefinitely, or even just continues for more than a fraction of a second, because when sleep is being called, the processor is freed up to allow other processes to do some work.

See Also

For an interesting discussion of how `time.sleep` can reduce the CPU load of your Python program, see *http://raspberrypi.stackexchange.com/questions/8077/how-can-i-lower-the-usage-of-cpu-for-this-python-program*.

7.20 Using Python with Minecraft Pi Edition

Problem

Now that you know some Python, you want to use it with Minecraft.

Solution

Use the Python interface that comes with Minecraft Pi edition to interact with Minecraft Pi while it is running.

Although you can switch back and forth between an LXTerminal session and the Minecraft game, the game will pause each time the window loses focus, so it is a lot easier to connect to the Raspberry Pi by using SSH on a second computer (Recipe 2.7). This has the added benefit that you get to see the Python script in action, live in the game.

The following example, which you can download from the Downloads section of the *Raspberry Pi Cookbook* website, builds a staircase at your current location (Figure 7-2):

```
from mcpi import minecraft, block
mc = minecraft.Minecraft.create()

mc.postToChat("Lets Build a Staircase!")

x, y, z = mc.player.getPos()

for xy in range(1, 50):
    mc.setBlock(x + xy, y + xy, z, block.STONE)
```

The Python library comes pre-installed in Raspbian, so if it's not there, you should probably update your system (Recipe 1.5).

After the library imports, the variable `mc` is assigned to a new instance of the Minecraft class.

The `postToChat` method sends a message to the player's screen telling him or her that the staircase is about to be built.

The variables `x`, `y`, and `z` are bound to the player's position and then the `for` loop increments both the `x` and `y` (`y` is height) repeatedly to build the staircase using the `setBlock` method (Figure 7-2).

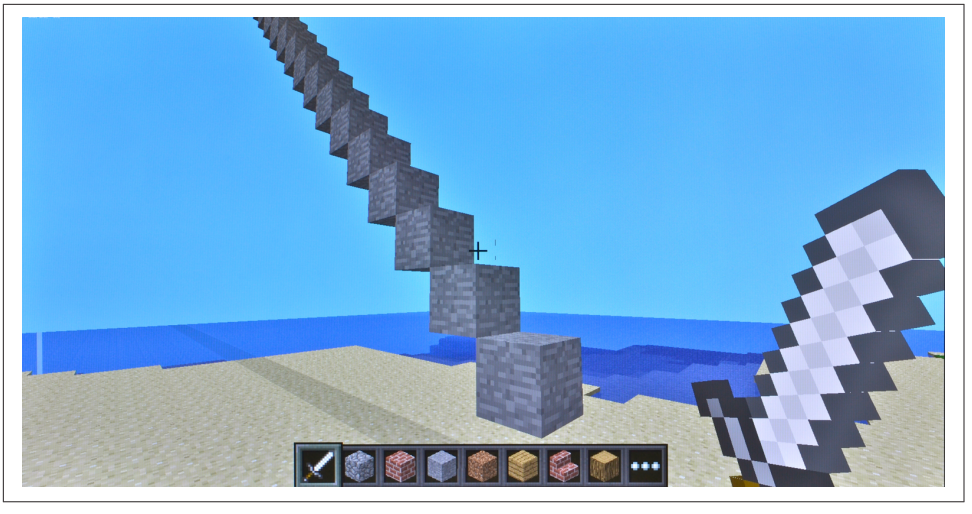

Figure 7-2. Building a staircase in Minecraft Pi with Python

Discussion

The `mcpi` library does not just allow you to post to chat, find the player's location, and place a block, it also provides a whole load of other methods that allow you to:

- Find the ID of the block at the coordinates specified
- Find out who's playing and teleport them perhaps?
- Set the position of a player
- Get the direction a player is facing

There is no definitive documentation for the class other than this useful blog post.

See Also

For more information on the Minecraft Pi Edition, see Recipe 4.7.

To learn about running a Minecraft server on a Raspberry Pi, see Recipe 4.8.