Table of Contents

Dive Into Python

Chapter 1. Installing Python
1.1. Which Python is right for you?
1.2. Python on Windows
1.3. Python on Mac OS X
1.4. Python on Mac OS 9
1.5. Python on RedHat Linux
1.6. Python on Debian GNU/Linux
1.7. Python Installation from Source
1.8. The Interactive Shell
1.9. Summary

Chapter 2. Your First Python Program
2.1. Diving In
2.2. Declaring Functions
2.3. Documenting Functions
2.4. Everything Is an Object
2.5. Indenting Code
2.6. Testing Modules

Chapter 3. Native Datatypes
3.1. Introducing Dictionaries
3.2. Introducing Lists
3.3. Introducing Tuples
3.4. Declaring Variables
3.5. Formatting Strings
3.6. Mapping Lists
3.7. Joining and Splitting Strings
3.8. Summary

Chapter 4. The Power of Introspection
4.1. Diving In
4.2. Using Optional and Named Arguments
4.3. Using type, str, dir, and Other Built-In Functions
4.4. Getting References Without Executing
4.5. Filtering Lists
4.6. The Peculiar Nature of and and or
4.7. Using lambda Functions
4.8. Putting It All Together
4.9. Summary

Chapter 5. Objects and Object-Oriented Programming
5.1. Diving In
5.2. Importing Modules Using from module import
5.3. Defining Classes
5.4. Instantiating Classes
5.5. Exploring UserDict: A Wrapper Class
5.6. Special Class Methods
5.7. Advanced Special Class Methods
5.8. Introducing Class Attributes

Table of Contents

Chapter 5. Objects and Object-Orientation
5.9. Private Functions
5.10. Summary

Chapter 6. Exceptions and File Handling
6.1. Handling Exceptions
6.2. Working with Files
6.3. Iterating with for Loops
6.4. Using sys.modules
6.5. Working with Directories
6.6. Putting It All Together
6.7. Summary

Chapter 7. Regular Expressions
7.1. Diving In
7.2. Case Study: Street Addresses
7.3. Case Study: Roman Numerals
7.4. Using the Using Syntax
7.5. Verbose Regular Expressions
7.6. Case Study: Parsing Phone Numbers
7.7. Summary

Chapter 8. HTML Processing
8.1. Diving in
8.2. Introducing sgmlib.py
8.3. Extracting data from HTML documents
8.4. HTML Processor.py
8.5. Locals and globals
8.6. Dictionary-based string formatting
8.7. Quoting attribute values
8.8. Introducing dialect.py
8.9. Putting it all together
8.10. Summary

Chapter 9. XML Processing
9.1. Diving in
9.2. Packages
9.3. Parsing XML
9.4. Unicode
9.5. Searching for elements
9.6. Accessing element attributes
9.7. Segue

Chapter 10. Scripts and Streams
10.1. Abstracting input sources
10.2. Standard input, standard, and error
10.3. Locking node lockings
10.4. Finding the children of a node
10.5. Creating separate handlers by node type
10.6. Handling command-line arguments
10.7. Putting it all together

Table of Contents

Chapter 10. Scripts and Streams

10.8. Summary

Chapter 11. HTTP Web Services

11.1. Diving in
11.2. Hoofing it over HTTP
11.3. HTTP Features
11.4. Debugging Web services
11.5. Set-Agent
11.6. Modified and ETag
11.7. Handling Last-Modified
11.8. Handling compressed data
11.9. Putting it all together
11.10. Summary

Chapter 12. SOAP Web Services

12.1. Diving in
12.2. Installing SOAP Libraries
12.3. First Steps with SOAP
12.4. Debugging SOAP Web Services
12.5. Introducing WSDL
12.6. Introspecting SOAP Web Services with WSDL
12.7. Searching Google
12.8. Probe Services
12.9. Summary

Chapter 13. Unit Testing

13.1. Introduction to Romanic numerals
13.2. Diving in
13.3. Introducing Test Driven Development
13.4. Testing for success
13.5. Testing for failure
13.6. Testing for sanity

Chapter 14. Test-First Programming

14.1. roman.py, stage 1
14.2. roman.py, stage 2
14.3. roman.py, stage 3
14.4. roman.py, stage 4
14.5. roman.py, stage 5

Chapter 15. Refactoring

15.1. Handling bugs
15.2. Handling changing requirements
15.3. Refactoring
15.4. Postscript
15.5. Summary

Chapter 16. Functional Programming

16.1. Diving in
16.2. Finding the path

Table of Contents

Chapter 16. Functional Programming
16.3. Filtering
16.4. Mapping
16.5. Data-centric programming
16.6. Dynamically modifying modules
16.7. Putting it all together
16.8. Summary

Chapter 17. Dynamic Functions
17.1. Diving
17.2. plural.py, stage 1
17.3. plural.py, stage 2
17.4. plural.py, stage 3
17.5. plural.py, stage 4
17.6. plural.py, stage 5
17.7. plural.py, stage 6
17.8. Summary

Chapter 18. Performance Tuning
18.1. Diving
18.2. Using Modules
18.3. Optimizing Regular Expressions
18.4. Optimizing Dictionary Lookups
18.5. Optimizing List Operations
18.6. Optimizing Stringy Manipulation
18.7. Summary

Appendix A. Further Reading

Appendix B. A Preview

Appendix C. Tips

Appendix D. List of examples

Appendix E. Revision history

Appendix F. About the book

Appendix G. GNU Free Documentation License
G.0. Preamble
G.1. Applicability and definitions
G.2. Verbatim copying
G.3. Copying in quantity
G.4. Modifications
G.5. Combining documents
G.6. Collections of documents
G.7. Aggregation with independent works
G.8. Translation
G.9. Termination
G.10. Future revisions of this license

Table of Contents

Appendix G. GNU Free Documentation License
G.1. How to use this License for your documents

Appendix H. Python Licenses
H.A. History of the software
H.B. Terms and conditions for accessing or otherwise using Python

Dive Into Python

20 May 2004

Copyright © 2000, 2001, 2002, 2003, 2004 Mark Pilgrim (mailto:mark@diveintopython.org)

This book lives at http://diveintopython.org/. If you're reading it somewhere else, you may not have the latest version.

Permission is granted to copy, distribute, and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1

The example programs in this book are free software; you can redistribute and/or modify them under the terms of the Python license as publis

Chapter 1. Installing Python

Welcome to Python. Let’s dive in. In this chapter, you’ll install the version of Python that’s right for you.

1.1. Which Python is right for you?

The first thing you need to do with Python is install it. Or do you?

If you’re using an account on a hosted server, your ISP may have already installed Python. Most popular Linux distributions come with Pyth

Windows does not come with any version of Python, but don’t despair! There are several ways to point-and-click your way to Python on Wi

As you can see already, Python runs on a great many operating systems. The full list includes Windows, Mac OS, Mac OS X, and all varietie

What’s more, Python programs written on one platform can, with a little care, run on *any* supported platform. For instance, I regularly develo

So back to the question that started this section, "Which Python is right for you?" The answer is whichever one runs on the computer you alre

1.2. Python on Windows

On Windows, you have a couple choices for installing Python.

ActiveState makes a Windows installer for Python called ActivePython, which includes a complete version of Python, an IDE with a Python-

ActivePython is freely downloadable, although it is not open source. It is the IDE I used to learn Python, and I recommend you try it unless y

The second option is the "official" Python installer, distributed by the people who develop Python itself. It is freely downloadable and open s

Procedure 1.1. Option 1: Installing ActivePython

Here is the procedure for installing ActivePython:

1. Download ActivePython from http://www.activestate.com/Products/ActivePython/.
2. If you are using Windows 95, Windows 98, or Windows ME, you will also need to download and install Windows Installer 2.0 (http
3. Double-click the installer,
4. Step through the installer program.
5. If space is tight, you can do a custom installation and deselect the documentation, but I don’t recommend this unless you absolutely
6. After the installation is complete, close the installer and choose Start->Programs->ActiveState ActivePython 2.2->PythonWin IDE.



>

Procedure 1.2. Option 2: Installing Python from Python.org (http://www.python.org/)

1. Download the latest Python Windows installer by going to http://www.python.org/ftp/python/ and selecting the highest version num

2. Double-click the installer, The name will depend on the version of Python available when you read this.

3. Step through the installer program.

4. If disk space is tight, you can deselect the HTMLHelp file, the utility scripts (and/or the test suite (

5. If you do not have administrative rights on your machine, you can select Advanced Options, then choose Non-Admin Install. This ju

6. After the installation is complete, close the installer and select Start->Programs->Python 2.3->IDLE (Python GUI). You'll see some

*

*

>

1.3. Python on Mac OS X

On Mac OS X, you have two choices for installing Python: install it, or don't install it. You probably want to install it.

Mac OS X 10.2 and later comes with a command-line version of Python preinstalled. If you are comfortable with the command line, you can

Rather than using the preinstalled version, you'll probably want to install the latest version, which also comes with a graphical interactive she

Procedure 1.3. Running the Preinstalled Version of Python on Mac OS X

To use the preinstalled version of Python, follow these steps:

1. Open the older.

2. Open the older.

Double-click to open a terminal window and get to a command line.3.

4.
Type at the command prompt.

Try it out:

```
>
>
>
>
>
>
>
>
```

Procedure 1.4. Installing the Latest Version of Python on Mac OS X

Follow these steps to download and install the latest version of Python:

1.
Download the disk image from http://homepages.cwi.nl/~jack/macpython/download.html.

2.
If your browser has not already done so, double-click to mount the disk image on your desktop.

3.
Double-click the installer,

4.
The installer will prompt you for your administrative username and password.

5.
Step through the installer program.

6.
After installation is complete, close the installer and open the folder.

7.
Open the folder

8.
Double-click to launch Python.

The MacPython IDE should display a splash screen, then take you to the interactive shell. If the interactive shell does not appear, select Wind

```
>
>
>
>
>
```

Note that once you install the latest version, the pre-installed version is still present. If you are running scripts from the command line, you ne

Example 1.1. Two versions of Python

```
[illegible]
```

1.4. Python on Mac OS 9

Mac OS 9 does not come with any version of Python, but installation is very simple, and there is only one choice.

Follow these steps to install Python on Mac OS 9:

1. Download the [illegible]ile from http://homepages.cwi.nl/~jack/macpython/download.html.
2. If your browser does not decompress the file automatically, double-click [illegible]o decompress the file with Stuffit Expander.
3. Double-click the installer, [illegible]
4. Step through the installer program.
5. AFter installation is complete, close the installer and open the [illegible]older.
6. Open the [illegible]older.
7. Double-click [illegible]o launch Python.

The MacPython IDE should display a splash screen, and then take you to the interactive shell. If the interactive shell does not appear, select W

```
[illegible]
>
```

1.5. Python on RedHat Linux

Installing under UNIX-compatible operating systems such as Linux is easy if you're willing to install a binary package. Pre-built binary pack

Download the latest Python RPM by going to http://www.python.org/ftp/python/ and selecting the highest version number listed, then selecti

Example 1.2. Installing on RedHat Linux 9



❶ Whoops! Just typing gives you the older version of Python -- the one that was installed by default. That's not the one you want.

❷ At the time of this writing, the newest version is called You'll probably want to change the path on the first line of the sample scripts

❸ This is the complete path of the newer version of Python that you just installed. Use this on the #ine (the first line of each script) to ens

1.6. Python on Debian GNU/Linux

If you are lucky enough to be running Debian GNU/Linux, you install Python through the **apt** command.

Example 1.3. Installing on Debian GNU/Linux





1.7. Python Installation from Source

If you prefer to build from source, you can download the Python source code from http://www.python.org/ftp/python/. Select the highest vers

Example 1.4. Installing from source



1.8. The Interactive Shell

Now that you have Python installed, what's this interactive shell thing you're running?

It's like this: Python leads a double life. It's an interpreter for scripts that you can run from the command line or run like applications, by dou

Launch the Python interactive shell in whatever way works on your platform, and let's dive in with the steps shown here:

Example 1.5. First Steps in the Interactive Shell

```
2
3
```

❶ The Python interactive shell can evaluate arbitrary Python expressions, including any basic arithmetic expression.

❷ The interactive shell can execute arbitrary Python statements, including the **print** statement.

❸ You can also assign values to variables, and the values will be remembered as long as the shell is open (but not any longer than that).

1.9. Summary

You should now have a version of Python installed that works for you.

Depending on your platform, you may have more than one version of Python intsalled. If so, you need to be aware of your paths. If simply ty

Congratulations, and welcome to Python.

Chapter 2. Your First Python Program

You know how other books go on and on about programming fundamentals and finally work up to building a complete, working program? L

2.1. Diving in

Here is a complete, working Python program.

It probably makes absolutely no sense to you. Don't worry about that, because you're going to dissect it line by line. But read through it first

Example 2.1.

If you have not already done so, you can download this and other examples (http://diveintopython.org/download/diveintopython-examples-5.

```
}
```

Now run this program and see what happens.

In the ActivePython IDE on Windows, you can run the Python program you're editing by choosing File->Run... (**Ctrl-R**). Output is displaye

In the Python IDE on Mac OS, you can run a Python program with Python->Run window... (**Cmd-R**), but there is an important option you m

On UNIX-compatible systems (including Mac OS X), you can run a Python program from the command line:

The output of will look like this:

```
```

2.2. Declaring Functions

Python has functions like most other languages, but it does not have separate header files like C++ or sections like Pascal. When you need a

```
```

Note that the keyword starts the function declaration, followed by the function name, followed by the arguments in parentheses. Multiple arg

Also note that the function doesn't define a return datatype. Python functions do not specify the datatype of their return value; they don't ever

In Visual Basic, functions (that return a value) start with and subroutines (that do not return a value) start with There are no subroutines in

The argument, doesn't specify a datatype. In Python, variables are never explicitly typed. Python figures out what type a variable is and kee

In Java, C++, and other statically-typed languages, you must specify the datatype of the function return value and each function argument. In

2.2.1. How Python's Datatypes Compare to Other Programming Languages

An erudite reader sent me this explanation of how Python compares to other programming languages:

statically typed language
: A language in which types are fixed at compile time. Most statically typed languages enforce this by requiring you to decla

dynamically typed language
: A language in which types are discovered at execution time; the opposite of statically typed. VBScript and Python are dyna

strongly typed language
: A language in which types are always enforced. Java and Python are strongly typed. If you have an integer, you can't treat

weakly typed language
: A language in which types may be ignored; the opposite of strongly typed. VBScript is weakly typed. In VBScript, you can

So Python is both *dynamically typed* (because it doesn't use explicit datatype declarations) and *strongly typed* (because once a variable has a

2.3. Documenting Functions

You can document a Python function by giving it a

Example 2.2. Defining the Function's

```

```

Triple quotes signify a multi-line string. Everything between the start and end quotes is part of a single string, including carriage returns and

Triple quotes are also an easy way to define a string with both single and double quotes, like in Perl.

Everything between the triple quotes is the function's which documents what the function does. A if it exists, must be the first thing define

Many Python IDEs use the to provide context-sensitive documentation, so that when you type a function name, its appears as a tooltip. This

Further Reading on Documenting Functions

PEP 257 (http://www.python.org/peps/pep-0257.html) defines conventions.·
Python Style Guide (http://www.python.org/doc/essays/styleguide.html) discusses how to write a good
Python Tutorial (http://www.python.org/doc/current/tut/tut.html) discusses conventions for spacing in (http://www.python.org/doc

2.4. Everything Is an Object

In case you missed it, I just said that Python functions have attributes, and that those attributes are available at runtime.

A function, like everything else in Python, is an object.

Open your favorite Python IDE and follow along:

Example 2.3. Accessing the Function's

```
```

1. The first line imports the program as a module -- a chunk of code that you can use interactively, or from a larger Python program. (Yo
2. When you want to use functions defined in imported modules, you need to include the module name. So you can't just say it must be
3. Instead of calling the function as you would expect to, you asked for one of the function's attributes, d

n Python is like n Perl. Once you a Python module, you access its functions with once you a Perl module, you access its functions with

2.4.1. The Import Search Path

Before you go any further, I want to briefly mention the library search path. Python looks in several places when you try to import a module.

Example 2.4. Import Search Path

```
```

1. Importing the module makes all of its functions and attributes available.
2. s a list of directory names that constitute the current search path. (Yours will look different, depending on your operating system, wh
3. Actually, I lied; the truth is more complicated than that, because not all modules are stored as files. Some, like the module, are "built-
4. You can add a new directory to Python's search path at runtime by appending the directory name to and then Python will look in that

2.4.2. What's an Object?

Everything in Python is an object, and almost everything has attributes and methods. All functions have a built-in attribute __doc__ which returns the

Still, this begs the question. What is an object? Different programming languages define "object" in different ways. In some, it means that *all*

This is so important that I'm going to repeat it in case you missed it the first few times: *everything in Python is an object*. Strings are objects.

Further Reading on Objects

Python Reference Manual (http://www.python.org/doc/current/ref/) explains exactly what it means to say that everything in Python
eff-bot (http://www.effbot.org/guides/) summarizes Python objects (http://www.effbot.org/guides/python-objects.htm).·

2.5. Indenting Code

Python functions have no explicit begin or end, and no curly braces to mark where the function code starts and stops. The only delimiter is a colon (:) a

Example 2.5. Indenting the buildConnectionString Function

```
[illegible]
[illegible]


[illegible]
[illegible]
```

Code blocks are defined by their indentation. By "code block", I mean functions, if statements, for loops, while loops, and so forth. Indenting starts a bl

Example 2.6, "if Statements" shows an example of code indentation with if statements.

Example 2.6. if Statements

```
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
```

1. This is a function named fib that takes one argument, n. All the code within the function is indented.
2. Printing to the screen is very easy in Python, just use print. print statements can take any data type, including strings, integers, and other native t
3. if statements are a type of code block. If the if expression evaluates to true, the indented block is executed, otherwise it falls to the else block.
4. Of course if and else blocks can contain multiple lines, as long as they are all indented the same amount. This else block has two lines of code i

After some initial protests and several snide analogies to Fortran, you will make peace with this and start seeing its benefits. One major benef

Python uses carriage returns to separate statements and a colon and indentation to separate code blocks. C++ and Java use semicolons to sepa

Further Reading on Code Indentation

Python Reference Manual (http://www.python.org/doc/current/ref/) discusses cross-platform indentation issues and shows various in
Python Style Guide (http://www.python.org/doc/essays/styleguide.html) discusses good indentation style.·

2.6. Testing Modules

Python modules are objects and have several useful attributes. You can use this to easily test your modules as you write them. Here's an exam

```
def
```

Some quick observations before you get to the good stuff. First, parentheses are not required around the if expression. Second, the if statement e



Like C, Python uses == for comparison and = for assignment. Unlike C, Python does not support in-line assignment, so there's no chance of acci

So why is this particular if statement a trick? Modules are objects, and all modules have a built-in attribute __name__. A module's __name__ depends on how you'

```
def
def
def
```

Knowing this, you can design a test suite for your module within the module itself by putting it in this if statement. When you run the module d



On MacPython, there is an additional step to make the trick work. Pop up the module's options menu by clicking the black triangle in the up

Further Reading on Importing Modules

Python Reference Manual (http://www.python.org/doc/current/ref/) discusses the low-level details of importing modules (http://www

Chapter 3. Native Datatypes

You'll get back to your first Python program in just a minute. But first, a short digression is in order, because you need to know about diction

3.1. Introducing Dictionaries

One of Python's built-in datatypes is the dictionary, which defines one-to-one relationships between keys and values.

A dictionary in Python is like a hash in Perl. In Perl, variables that store hashes always start with a % character. In Python, variables can be nar

A dictionary in Python is like an instance of the Hashtable class in Java.

A dictionary in Python is like an instance of the Scripting.Dictionary object in Visual Basic.

3.1.1. Defining Dictionaries

Example 3.1. Defining a Dictionary

```
>>> d = {"server":"mpilgrim", "database":"master"}
>>> d
{'server': 'mpilgrim', 'database': 'master'}
>>> d["server"]
'mpilgrim'
>>> d["database"]
'master'
>>> d["mpilgrim"]
Traceback (innermost last):
  File "<interactive input>", line 1, in ?
KeyError: mpilgrim
```

1. First, you create a new dictionary with two elements and assign it to the variable d. Each element is a key-value pair, and the whole set
2. is a key, and its associated value, referenced by is
3. is a key, and its associated value, referenced by is
4. You can get values by key, but you can't get keys by value. So gives but raises an exception, because is not a key.

3.1.2. Modifying Dictionaries

Example 3.2. Modifying a Dictionary

```
>>> d
{'server': 'mpilgrim', 'database': 'master'}
>>> d["database"] = "pubs"
>>> d
{'server': 'mpilgrim', 'database': 'pubs'}
>>> d["uid"] = "sa"
>>> d
{'server': 'mpilgrim', 'uid': 'sa', 'database': 'pubs'}
```

❶ You can not have duplicate keys in a dictionary. Assigning a value to an existing key will wipe out the old value.

❷ You can add new key-value pairs at any time. This syntax is identical to modifying existing values. (Yes, this will annoy you someday

Note that the new element (key d value appears to be in the middle. In fact, it was just a coincidence that the elements appeared to be in orde

Dictionaries have no concept of order among elements. It is incorrect to say that the elements are "out of order"; they are simply unordered.

When working with dictionaries, you need to be aware that dictionary keys are case-sensitive.

Example 3.3. Dictionary Keys Are Case-Sensitive

❶ Assigning a value to an existing dictionary key simply replaces the old value with a new one.

❷ This is not assigning a value to an existing dictionary key, because strings in Python are case-sensitive, so s not the same as This cre

Example 3.4. Mixing Datatypes in a Dictionary

❶ Dictionaries aren't just for strings. Dictionary values can be any datatype, including strings, integers, objects, or even other dictionarie

❷ Dictionary keys are more restricted, but they can be strings, integers, and a few other types. You can also mix and match key datatypes

3.1.3. Deleting Items From Dictionaries

Example 3.5. Deleting Items from a Dictionary

```
❷
d
I
```

❶ dets you delete individual items from a dictionary by key.

❷ deletes all items from a dictionary. Note that the set of empty curly braces signifies a dictionary without any items.

Further Reading on Dictionaries

How to Think Like a Computer Scientist (http://www.ibiblio.org/obp/thinkCSpy/) teaches about dictionaries and shows how to use d
Python Knowledge Base (http://www.faqts.com/knowledge-base/index.phtml/fid/199/) has a lot of example code using dictionaries
Python Cookbook (http://www.activestate.com/ASPN/Python/Cookbook/) discusses how to sort the values of a dictionary by key (h
Python Library Reference (http://www.python.org/doc/current/lib/) summarizes all the dictionary methods (http://www.python.org/d

3.2. Introducing Lists

Lists are Python's workhorse datatype. If your only experience with lists is arrays in Visual Basic or (God forbid) the datastore in Powerbuild

A list in Python is like an array in Perl. In Perl, variables that store arrays always start with the @ character; in Python, variables can be named

A list in Python is much more than an array in Java (although it can be used as one if that's really all you want out of life). A better analogy w

3.2.1. Defining Lists

Example 3.6. Defining a List

```
❶
i
n
❷
a
❸
n
```

❶ First, you define a list of five elements. Note that they retain their original order. This is not an accident. A list is an ordered set of elem

❷ A list can be used like a zero-based array. The first element of any non-empty list is always 0

❸ The last element of this five-element list is 4 because lists are always zero-based.

Example 3.7. Negative List Indices

```
i
n
❶
n
❷
0
```

❶ A negative index accesses elements from the end of the list counting backwards. The last element of any non-empty list is always -1

❷ If the negative index is confusing to you, think of it this way: So in this list,

Example 3.8. Slicing a List

❶ You can get a subset of a list, called a "slice", by specifying two indices. The return value is a new list containing all the elements of th

❷ Slicing works if one or both of the slice indices is negative. If it helps, you can think of it this way: reading the list from left to right, th

❸ Lists are zero-based, so returns the first three elements of the list, starting at up to but not including

Example 3.9. Slicing Shorthand

❶ If the left slice index is 0, you can leave it out, and 0 is implied. So is the same as from Example 3.8, “Slicing a List”.

❷ Similarly, if the right slice index is the length of the list, you can leave it out. So is the same as because this list has five elements.

❸ Note the symmetry here. In this five-element list, returns the first 3 elements, and returns the last two elements. In fact, will always r

❹ If both slice indices are left out, all elements of the list are included. But this is not the same as the original list; it is a new list that hap

3.2.2. Adding Elements to Lists

Example 3.10. Adding Elements to a List

❶ adds a single element to the end of the list.

❷ inserts a single element into a list. The numeric argument is the index of the first element that gets bumped out of position. Note that l

❸ concatenates lists. Note that you do not call with multiple arguments; you call it with one argument, a list. In this case, that list has tw

Example 3.11. The Difference between and

```
6
f
4
```

1. Lists have two methods, and that look like they do the same thing, but are in fact completely different. takes a single argument, whi
2. Here you started with a list of three elements (and , and you extended the list with a list of another three elements (and , so you
3. On the other hand, takes one argument, which can be any data type, and simply adds it to the end of the list. Here, you're calling the
4. Now the original list, which started as a list of three elements, contains four elements. Why four? Because the last element that you jus

3.2.3. Searching Lists

Example 3.12. Searching a List

```
5
2
```

1. finds the first occurrence of a value in the list and returns the index.
2. finds the *first* occurrence of a value in the list. In this case, occurs twice in the list, in and but will return only the first index, 2
3. If the value is not found in the list, Python raises an exception. This is notably different from most languages, which will return some i
4. To test whether a value is in the list, use which returns if the value is found or if it is not.

Before version 2.2.1, Python had no separate boolean datatype. To compensate for this, Python accepted almost anything in a boolean contex
- is false; all other numbers are true.
- An empty string ( ) is false, all other strings are true.

An empty list ([]) is false; all other lists are true.
An empty tuple ([]) is false; all other tuples are true.
An empty dictionary ([]) is false; all other dictionaries are true.

These rules still apply in Python 2.2.1 and beyond, but now you can also use an actual boolean, which has a value of [illegible] or [illegible] Note the capitalizat

3.2.4. Deleting List Elements

Example 3.13. Removing Elements from a List

[illegible]

❶ [illegible]emoves the first occurrence of a value from a list.

❷ [illegible]emoves *only* the first occurrence of a value. In this case, [illegible]ppeared twice in the list, but [illegible]emoved only the first occurrence.

❸ If the value is not found in the list, Python raises an exception. This mirrors the behavior of the [illegible]method.

❹ [illegible]s an interesting beast. It does two things: it removes the last element of the list, and it returns the value that it removed. Note that this

3.2.5. Using List Operators

Example 3.14. List Operators

[illegible]

❶ Lists can also be concatenated with the [illegible]operator. [illegible]as the same result as [illegible] But the [illegible]operator returns a new (concatenated) list as a value

❷ Python supports the [illegible]operator. [illegible]s equivalent to [illegible] The [illegible]operator works for lists, strings, and integers, and it can be overloaded to work fo

❸ The [illegible]operator works on lists as a repeater. [illegible]s equivalent to [illegible] which concatenates the three lists into one.

Further Reading on Lists

How to Think Like a Computer Scientist (http://www.ibiblio.org/obp/thinkCSpy/) teaches about lists and makes an important point a

Python Tutorial (http://www.python.org/doc/current/tut/tut.html) shows how to use lists as stacks and queues (http://www.python.or
Python Knowledge Base (http://www.faqts.com/knowledge-base/index.phtml/fid/199/) answers common questions about lists (http:
Python Library Reference (http://www.python.org/doc/current/lib/) summarizes all the list methods (http://www.python.org/doc/cur

3.3. Introducing Tuples

A tuple is an immutable list. A tuple can not be changed in any way once it is created.

Example 3.15. Defining a tuple

```
>>> t = ("a", "b", "mpilgrim", "z", "example")
>>> t
('a', 'b', 'mpilgrim', 'z', 'example')
>>> t[0]
'a'
>>> t[-1]
'example'
>>> t[1:3]
('b', 'mpilgrim')
```

1. A tuple is defined in the same way as a list, except that the whole set of elements is enclosed in parentheses instead of square brackets.
2. The elements of a tuple have a defined order, just like a list. Tuples indices are zero-based, just like a list, so the first element of a non-
3. Negative indices count from the end of the tuple, just as with a list.
4. Slicing works too, just like a list. Note that when you slice a list, you get a new list; when you slice a tuple, you get a new tuple.

Example 3.16. Tuples Have No Methods

```
>>> t
('a', 'b', 'mpilgrim', 'z', 'example')
>>> t.append("new")
Traceback (innermost last):
  File "<interactive input>", line 1, in ?
AttributeError: 'tuple' object has no attribute 'append'
>>> t.remove("z")
Traceback (innermost last):
  File "<interactive input>", line 1, in ?
AttributeError: 'tuple' object has no attribute 'remove'
>>> t.index("example")
Traceback (innermost last):
  File "<interactive input>", line 1, in ?
AttributeError: 'tuple' object has no attribute 'index'
>>> "z" in t
1
```

1. You can’t add elements to a tuple. Tuples have no `append` or `extend` method.
2. You can’t remove elements from a tuple. Tuples have no `remove` or `pop` method.
3. You can’t find elements in a tuple. Tuples have no `index` method.
4. You can, however, use `in` to see if an element exists in the tuple.

So what are tuples good for?

Tuples are faster than lists. If you’re defining a constant set of values and all you’re ever going to do with it is iterate through it, use

It makes your code safer if you "write-protect" data that does not need to be changed. Using a tuple instead of a list is like having an
Remember that I said that dictionary keys can be integers, strings, and "a few other types"? Tuples are one of those types. Tuples ca
Tuples are used in string formatting, as you'll see shortly.·

Tuples can be converted into lists, and vice-versa. The built-in function takes a list and returns a tuple with the same elements, and the func

Further Reading on Tuples

How to Think Like a Computer Scientist (http://www.ibiblio.org/obp/thinkCSpy/) teaches about tuples and shows how to concatenat
Python Knowledge Base (http://www.faqts.com/knowledge-base/index.phtml/fid/199/) shows how to sort a tuple (http://www.faqts.
Python Tutorial (http://www.python.org/doc/current/tut/tut.html) shows how to define a tuple with one element (http://www.python

3.4. Declaring variables

Now that you know something about dictionaries, tuples, and lists (oh my!), let's get back to the sample program from Chapter 2,

Python has local and global variables like most other languages, but it has no explicit variable declarations. Variables spring into existence by

Example 3.17. Defining the Variable

```
}
```

Notice the indentation. An statement is a code block and needs to be indented just like a function.

Also notice that the variable assignment is one command split over several lines, with a backslash ("\) serving as a line-continuation marker.

When a command is split among several lines with the line-continuation marker ("\), the continued lines can be indented in any manner; Pyth

Strictly speaking, expressions in parentheses, straight brackets, or curly braces (like defining a dictionary) can be split into multiple lines with

Third, you never declared the variable you just assigned a value to it. This is like VBScript without the option. Luckily, unlike VBScript, Py

3.4.1. Referencing Variables

Example 3.18. Referencing an Unbound Variable

```
1
```

You will thank Python for this one day.

3.4.2. Assigning Multiple Values at Once

One of the cooler programming shortcuts in Python is using sequences to assign multiple values at once.

Example 3.19. Assigning multiple values at once

```
v = ('a', 2, True)
(x, y, z) = v
x
'a'
y
2
z
True
```

1. `v` is a tuple of three elements, and `(x, y, z)` is a tuple of three variables. Assigning one to the other assigns each of the values of `v` to each of the va

This has all sorts of uses. I often want to assign names to a range of values. In C, you would use `#define` and manually list each constant and its assoc

Example 3.20. Assigning Consecutive Values

```
MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY = range(7)
MONDAY
0
TUESDAY
1
SUNDAY
6
```

1. The built-in `range` function returns a list of integers. In its simplest form, it takes an upper limit and returns a zero-based list counting up to b
2. `MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY` are the variables you're defining. (This example came from the `calendar` module, a fun little module that prints calendars, like the `
3. Now each variable has its value: `MONDAY` is `0`, `TUESDAY` is `1`, and so forth.

You can also use multi-variable assignment to build functions that return multiple values, simply by returning a tuple of all the values. The ca

Further Reading on Variables

Python Reference Manual (http://www.python.org/doc/current/ref/) shows examples of when you can skip the line continuation char
How to Think Like a Computer Scientist (http://www.ibiblio.org/obp/thinkCSpy/) shows how to use multi-variable assignment to sw

3.5. Formatting Strings

Python supports formatting values into strings. Although this can include very complicated expressions, the most basic usage is to insert valu

String formatting in Python uses the same syntax as the `sprintf` function in C.

Example 3.21. Introducing String Formatting

```
k = "uid"
pw = "sesame"
```

```
```

1. The whole expression evaluates to a string. The first %s is replaced by the value of k, the second %s is replaced by the value of v. All other cha

Note that k,v is a tuple. I told you they were good for something.

You might be thinking that this is a lot of work just to do simple string concatentation, and you would be right, except that string formatting i

Example 3.22. String Formatting vs. Concatenating

```
```

1. + is the string concatenation operator.
2. In this trivial case, string formatting accomplishes the same result as concatentation.
3. (k,) is a tuple with one element. Yes, the syntax is a little strange, but there's a good reason for it: it's unambiguously a tuple. In fact, you
4. String formatting works with integers by specifying %d instead of %s.
5. Trying to concatenate a string with a non-string raises an exception. Unlike string formatting, string concatenation works only when ev

As with printf in C, string formatting in Python is like a Swiss Army knife. There are options galore, and modifier strings to specially format many

Example 3.23. Formatting Numbers

```
```

1. The %f string formatting option treats the value as a decimal, and prints it to six decimal places.
2. The ".2" modifier of the %f option truncates the value to two decimal places.
3. You can even combine modifiers. Adding the + modifier displays a plus or minus sign before the value. Note that the ".2" modifier is st

Further Reading on String Formatting

Python Library Reference (http://www.python.org/doc/current/lib/) summarizes all the string formatting format characters (http://ww
Effective AWK Programming (http://www-gnats.gnu.org:8080/cgi-bin/info2www?(gawk)Top) discusses all the format characters (h

3.6. Mapping Lists

One of the most powerful features of Python is the list comprehension, which provides a compact way of mapping a list into another list by a

Example 3.24. Introducing List Comprehensions



1. To make sense of this, look at it from right to left. is the list you're mapping. Python loops through one element at a time, temporarily
2. Note that list comprehensions do not change the original list.
3. It is safe to assign the result of a list comprehension to the variable that you're mapping. Python constructs the new list in memory, an

Here are the list comprehensions in the function that you declared in Chapter 2:



First, notice that you're calling the function of the dictionary. This function returns a list of tuples of all the data in the dictionary.

Example 3.25. The and Functions



1. The method of a dictionary returns a list of all the keys. The list is not in the order in which the dictionary was defined (remember tha
2. The method returns a list of all the values. The list is in the same order as the list returned by so for all values of n
3. The method returns a list of tuples of the form The list contains all the data in the dictionary.

Now let's see what does. It takes a list, and maps it to a new list by applying string formatting to each element. The new list will have the sa

Example 3.26. List Comprehensions in Step by Step



❶ Note that you're using two variables to iterate through the list. This is another use of multi-variable assignment. The first element of

❷ Here you're doing the same thing, but ignoring the value of so this list comprehension ends up being equivalent to

❸ Combining the previous two examples with some simple string formatting, you get a list of strings that include both the key and value

Further Reading on List Comprehensions

Python Tutorial (http://www.python.org/doc/current/tut/tut.html) discusses another way to map lists using the built-in function (http

Python Tutorial (http://www.python.org/doc/current/tut/tut.html) shows how to do nested list comprehensions (http://www.python.o

3.7. Joining Lists and Splitting Strings

You have a list of key-value pairs in the form and you want to join them into a single string. To join any list of strings into a single string, us

Here is an example of joining a list from the function:

One interesting note before you continue. I keep repeating that functions are objects, strings are objects... everything is an object. You might

The method joins the elements of the list into a single string, with each element separated by a semi-colon. The delimiter doesn't need to be

works only on lists of strings; it does not do any type coercion. Joining a list that has one or more non-string elements will raise an exceptio

Example 3.27. Output of

This string is then returned from the function and printed by the calling block, which gives you the output that you marveled at when you sta

You're probably wondering if there's an analogous method to split a string into a list. And of course there is, and it's called

Example 3.28. Splitting a String

❶ reverses by splitting a string into a multi-element list. Note that the delimiter ("";) is stripped out completely; it does not appear in any

❷ takes an optional second argument, which is the number of times to split. (""Oooooh, optional arguments..." You'll learn how to do th

is a useful technique when you want to search a string for a substring and then work with everything before the substring (which ends up in

Further Reading on String Methods

Python Knowledge Base (http://www.faqts.com/knowledge-base/index.phtml/fid/199/) answers common questions about strings (ht
Python Library Reference (http://www.python.org/doc/current/lib/) summarizes all the string methods (http://www.python.org/doc/c
Python Library Reference (http://www.python.org/doc/current/lib/) documents the module (http://www.python.org/doc/current/lib/m
The Whole Python FAQ (http://www.python.org/doc/FAQ.html) explains why is a string method (http://www.python.org/cgi-bin/fa

3.7.1. Historical Note on String Methods

When I first learned Python, I expected to be a method of a list, which would take the delimiter as an argument. Many people feel the same w

3.8. Summary

The program and its output should now make perfect sense.

```
}
```

Here is the output of

```
```

Before diving into the next chapter, make sure you're comfortable doing all of these things:

- Using the Python IDE to test expressions interactively·
- Writing Python programs and running them from within your IDE, or from the command line·
- Importing modules and calling their functions·
- Declaring functions and using local variables, and proper indentation·
- Defining dictionaries, tuples, and lists·
- Accessing attributes and methods of any object, including strings, lists, dictionaries, functions, and modules·
- Concatenating values through string formatting·
- Mapping lists into other lists using list comprehensions·
- Splitting strings into lists and joining lists into strings·

Chapter 4. The Power Of Introspection

This chapter covers one of Python's strengths: introspection. As you know, everything in Python is an object, and introspection is code lookin

4.1. Diving In

Here is a complete, working Python program. You should understand a good deal about it just by looking at it. The numbered lines illustrate

Example 4.1.

If you have not already done so, you can download this and other examples (http://diveintopython.org/download/diveintopython-examples-5.

❶❷❸

❹❺

❶ This module has one function, According to its function declaration, it takes three parameters: and The last two are actually optio

❷ The function has a multi-line that succinctly describes the function's purpose. Note that no return value is mentioned; this function w

❸ Code within the function is indented.

❹ The trick allows this program do something useful when run by itself, without interfering with its use as a module for other programs.

❺ Statements use for comparison, and parentheses are not required.

The function is designed to be used by you, the programmer, while working in the Python IDE. It takes any object that has functions or meth

Example 4.2. Sample Usage of

By default the output is formatted to be easy to read. Multi-line are collapsed into a single long line, but this option can be changed by speci

Example 4.3. Advanced Usage of

4.2. Using Optional and Named Arguments

Python allows function arguments to have default values; if the function is called without the argument, the argument gets its default value. F

Here is an example of a function with two optional arguments:

and are optional, because they have default values defined. is required, because it has no default value. If is called with only one argument,

Say you want to specify a value for but want to accept the default value for In most languages, you would be out of luck, because you woul

Example 4.4. Valid Calls of

1. With only one argument, gets its default value of and gets its default value of 1
2. With two arguments, gets its default value of 1
3. Here you are naming the argument explicitly and specifying its value. still gets its default value of 0
4. Even required arguments (like which has no default value) can be named, and named arguments can appear in any order.

This looks totally whacked until you realize that arguments are simply a dictionary. The "normal" method of calling functions without argum

The only thing you need to do to call a function is specify a value (somehow) for each required argument; the manner and order in which you

Further Reading on Optional Arguments

Python Tutorial (http://www.python.org/doc/current/tut/tut.html) discusses exactly when and how default arguments are evaluated (

4.3. Using and Other Built-In Functions

Python has a small set of extremely useful built-in functions. All other functions are partitioned off into modules. This was actually a conscio

4.3.1. The Function

The function returns the datatype of any arbitrary object. The possible types are listed in the module. This is useful for helper functions that

Example 4.5. Introducing

❶ takes anything -- and I mean anything -- and returns its datatype. Integers, strings, lists, dictionaries, tuples, functions, classes, module

❷ can take a variable and return its datatype.

❸ also works on modules.

❹ You can use the constants in the module to compare types of objects. This is what the function does, as you'll see shortly.

4.3.2. The Function

The coerces data into a string. Every datatype can be coerced into a string.

Example 4.6. Introducing

❶ For simple datatypes like integers, you would expect to work, because almost every language has a function to convert an integer to a

❷ However, works on any object of any type. Here it works on a list which you've constructed in bits and pieces.

❸ also works on modules. Note that the string representation of the module includes the pathname of the module on disk, so yours will b

❹ A subtle but important behavior of is that it works on the Python null value. It returns the string You'll use this to your advantage i

At the heart of the function is the powerful function. returns a list of the attributes and methods of any object: modules, functions, strings, li

Example 4.7. Introducing

```
```

1. is a list, so returns a list of all the methods of a list. Note that the returned list contains the names of the methods as strings, not the m
2. ds a dictionary, so returns a list of the names of dictionary methods. At least one of these, should look familiar.
3. This is where it really gets interesting. s a module, so returns a list of all kinds of stuff defined in the module, including built-in attrib

Finally, the unction takes any object and returns f the object can be called, or otherwise. Callable objects include functions, class methods

Example 4.8. Introducing a

```
```

1. The functions in the module are deprecated (although many people still use the function), but the module contains a lot of useful cons
2. s a function that joins a list of strings.
3. s not callable; it is a string. (A string does have callable methods, but the string itself is not callable.)
4. s callable; it's a function that takes two arguments.
5. Any callable object may have a By using the unction on each of an object's attributes, you can determine which attributes you care

4.3.3. Built-In Functions

and all the rest of Python's built-in functions are grouped into a special module called (That's two underscores before and after.) If it hel

The advantage of thinking like this is that you can access all the built-in functions and attributes as a group by getting information about the

Example 4.9. Built-in Attributes and Functions

```
```

Python comes with excellent reference manuals, which you should peruse thoroughly to learn all the modules Python has to offer. But unlike

Further Reading on Built-In Functions

Python Library Reference (http://www.python.org/doc/current/lib/) documents all the built-in functions (http://www.python.org/doc

4.4. Getting Object References With

You already know that Python functions are objects. What you don't know is that you can get a reference to a function without knowing its n

Example 4.10. Introducing

1. This gets a reference to the method of the list. Note that this is not calling the method; that would be This is the method itself.
2. This also returns a reference to the method, but this time, the method name is specified as a string argument to the function. s an inc
3. In case it hasn't sunk in just how incredibly useful this is, try this: the return value of s the method, which you can then call just as if
4. also works on dictionaries.
5. In theory, would work on tuples, except that tuples have no methods, so will raise an exception no matter what attribute name you giv

4.4.1. with Modules

sn't just for built-in datatypes. It also works on modules.

Example 4.11. The Function in

```
```

1. This returns a reference to the function in the module, which you studied in Chapter 2, *Your First Python Program*. (The hex address
2. Using you can get the same reference to the same function. In general, is equivalent to If is a module, then can be anything defined
3. And this is what you actually use in the function. is passed into the function as an argument; is a string which is the name of a metho
4. In this case, is the name of a function, which you can prove by getting its
5. Since is a function, it is callable.

4.4.2. As a Dispatcher

A common usage pattern of is as a dispatcher. For example, if you had a program that could output data in a variety of different formats, you

For example, let's imagine a program that prints site statistics in HTML, XML, and plain text formats. The choice of output format could be

Example 4.12. Creating a Dispatcher with

```
```

1. The function takes one required argument, and one optional argument, If is not specified, it defaults to and you will end up callin
2. You concatenate the argument with "output_" to produce a function name, and then go get that function from the module. This allow
3. Now you can simply call the output function in the same way as any other function. The variable is a reference to the appropriate func

Did you see the bug in the previous example? This is a very loose coupling of strings and functions, and there is no error checking. What hap

Luckily, takes an optional third argument, a default value.

Example 4.13. Default Values

```
```

❶ This function call is guaranteed to work, because you added a third argument to the call to g The third argument is a default value that

As you can see, g is quite powerful. It is the heart of introspection, and you'll see even more powerful examples of it in later chapters.

4.5. Filtering Lists

As you know, Python has powerful capabilities for mapping lists into other lists, via list comprehensions (Section 3.6, "Mapping Lists"). This

Here is the list filtering syntax:

f

This is an extension of the list comprehensions that you know and love. The first two thirds are the same; the last part, starting with the f is th

Example 4.14. Introducing List Filtering

❶
❷
❸
❹
❺

❶ The mapping expression here is simple (it just returns the value of each element), so concentrate on the filter expression. As Python lo
❷ Here, you are filtering out a specific value, b Note that this filters all occurrences of b since each time it comes up, the filter expression
❸ is a list method that returns the number of times a value occurs in a list. You might think that this filter would eliminate duplicates fro

Let's get back to this line from p

This looks complicated, and it is complicated, but the basic structure is the same. The whole filter expression returns a list, which is assigned

The filter expression looks scary, but it's not. You already know about and i As you saw in the previous section, the expression returns a f

So this expression takes an object (named) Then it gets a list of the names of the object's attributes, methods, functions, and a few other thin

Further Reading on Filtering Lists

Python Tutorial (http://www.python.org/doc/current/tut/tut.html) discusses another way to filter lists using the built-in function (htt

4.6. The Peculiar Nature of and

In Python, and perform boolean logic as you would expect, but they do not return boolean values; instead, they return one of the actual valu

Example 4.15. Introducing and

❶
b
❷
'
❸

c

❶ When using values are evaluated in a boolean context from left to right. , , , and are false in a boolean context; everything else is t

❷ If any value is false in a boolean context, returns the first false value. In this case, is the first false value.

❸ All values are true, so returns the last value, c

Example 4.16. Introducing

a

b

a

❶ When using values are evaluated in a boolean context from left to right, just like If any value is true, returns that value immediately

❷ evaluates , which is false, then which is true, and returns b

❸ If all values are false, returns the last value. evaluates , which is false, then , which is false, then , which is false, and returns

❹ Note that evaluates values only until it finds one that is true in a boolean context, and then it ignores the rest. This distinction is impor

If you're a C hacker, you are certainly familiar with the expression, which evaluates to if is true, and otherwise. Because of the way and w

4.6.1. Using the Trick

Example 4.17. Introducing the Trick

f

d

❶ This syntax looks similar to the expression in C. The entire expression is evaluated from left to right, so the is evaluated first. evalute

❷ evalutes to and then evaluates to

However, since this Python expression is simply boolean logic, and not a special construct of the language, there is one extremely important

Example 4.18. When the Trick Fails

d

❶ Since is an empty string, which Python considers false in a boolean context, evalutes to , and then evalutes to Oops! That's not wha

The trick, but will not work like the C expression when is false in a boolean context.

The real trick behind the trick, then, is to make sure that the value of is never false. One common way of doing this is to turn into and into

Example 4.19. Using the Trick Safely

```
a
u
(1)
'
```

(1) Since is a non-empty list, it is never false. Even if is or or some other false value, the list is true because it has one element.

By now, this trick may seem like more trouble than it's worth. You could, after all, accomplish the same thing with an statement, so why go

Further Reading on the Trick

Python Cookbook (http://www.activestate.com/ASPN/Python/Cookbook/) discusses alternatives to the trick (http://www.activestate

4.7. Using Functions

Python supports an interesting syntax that lets you define one-line mini-functions on the fly. Borrowed from Lisp, these so-called unctions c

Example 4.20. Introducing Functions

```
#
a
.
3
6
(1)
g
a
6
(2)
x
6
```

(1) This is a unction that accomplishes the same thing as the normal function above it. Note the abbreviated syntax here: there are no par

(2) You can use a unction without even assigning it to a variable. This may not be the most useful thing in the world, but it just goes to s

To generalize, a unction is a function that takes any number of arguments (including optional arguments) and returns the value of a single ex

unctions are a matter of style. Using them is never required; anywhere you could use them, you could define a separate normal function and

4.7.1. Real-World Functions

Here are the unctions in

```
p
```

Notice that this uses the simple form of the trick, which is okay, because a unction is always true in a boolean context. (That doesn't mean

Also notice that you're using the function with no arguments. You've already seen it used with one or two arguments, but without any argum

Example 4.21. With No Arguments

```
1
2
3
```

1. This is a multiline string, defined by escape characters instead of triple quotes. is a carriage return, and is a tab character.
2. without any arguments splits on whitespace. So three spaces, a carriage return, and a tab character are all the same.
3. You can normalize whitespace by splitting a string with and then rejoining it with using a single space as a delimiter. This is what th

So what is the function actually doing with these functions, , and dricks?

```
is now a function, but which function it is depends on the value of the variable. If is true, will collapse whitespace; otherwise, will return i
```

To do this in a less robust language, like Visual Basic, you would probably create a function that took a string and a argument and used an

Further Reading on Functions

Python Knowledge Base (http://www.faqts.com/knowledge-base/index.phtml/fid/199/) discusses using to call functions indirectly (
Python Tutorial (http://www.python.org/doc/current/tut/tut.html) shows how to access outside variables from inside a function (http
The Whole Python FAQ (http://www.python.org/doc/FAQ.html) has examples of obfuscated one-liners using (http://www.python.o

4.8. Putting It All Together

The last line of code, the only one you haven't deconstructed yet, is the one that does all the work. But by now the work is easy, because ever

This is the meat of

```
```

Note that this is one command, split over multiple lines, but it doesn't use the line continuation character (). Remember when I said that some

Now, let's take it from the end and work backwards. The

```
```

shows that this is a list comprehension. As you know, is a list of all the methods you care about in So you're looping through that list with

Example 4.22. Getting a Dynamically

```
```

In the function, is the object you're getting help on, passed in as an argument.

As you're looping through is the name of the current method.

Using the function, you're getting a reference to the function in the module.

Now, printing the actual of the method is easy.

The next piece of the puzzle is the use of around the As you may recall, is a built-in function that coerces data into a string. But a is alway

Example 4.23. Why Use on a

You can easily define a function that has no so its attribute is Confusingly, if you evaluate the attribute directly, the Python IDE pr

You can verify that the value of the attribute is actually by comparing it directly.

The function takes the null value and returns a string representation of it,

In SQL, you must use instead of to compare a null value. In Python, you can use either or but is faster.

Now that you are guaranteed to have a string, you can pass the string to which you have already defined as a function that either does or doe

Stepping back even further, you see that you're using string formatting again to concatenate the return value of with the return value of is

Example 4.24. Introducing

pads the string with spaces to the given length. This is what the function uses to make two columns of output and line up all the in t

If the given length is smaller than the length of the string, will simply return the string unchanged. It never truncates the string.

You're almost finished. Given the padded method name from the method and the (possibly collapsed) from the call to you concatenate the

Example 4.25. Printing a List

```
a
b
c
```

❶ This is also a useful debugging trick when you're working with lists. And in Python, you're always working with lists.

That's the last piece of the puzzle. You should now understand this code.

4.9. Summary

The program and its output should now make perfect sense.

Here is the output of

Before diving into the next chapter, make sure you're comfortable doing all of these things:

- Defining and calling functions with optional and named arguments·

Using `str` to coerce any arbitrary value into a string representation·
Using `dir` to get references to functions and other attributes dynamically·
Extending the list comprehension syntax to do list filtering·
Recognizing the `and-or` trick and using it safely·
Defining `lambda` functions·
Assigning functions to variables and calling the function by referencing the variable. I can’t emphasize this enough, because this mo

Chapter 5. Objects and Object-Orientation

This chapter, and pretty much every chapter after this, deals with object-oriented Python programming.

5.1. Diving In

Here is a complete, working Python program. Read the of the module, the classes, and the functions to get an overview of what this progra

Example 5.1.

If you have not already done so, you can download this and other examples (http://diveintopython.org/download/diveintopython-examples-5.

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]



❶ This program’s output depends on the files on your hard drive. To get meaningful output, you’ll need to change the directory path to p

This is the output I got on my machine. Your output will be different, unless, by some startling coincidence, you share my exact taste in musi



[redacted]

5.2. Importing Modules Using

Python has two ways of importing modules. Both are useful, and you should know when to use each. One way, you've already seen in Secti

Here is the basic yntax:

This is similar to the yntax that you know and love, but with an important difference: the attributes and methods of the imported module

n Python is like n Perl; n Python is like n Perl.

[redacted]n Python is like [redacted]n Java; [redacted]n Python is like [redacted]n Java.

Example 5.2. [redacted]*vs.*[redacted]

[redacted]

❶ The [redacted]module contains no methods; it just has attributes for each Python object type. Note that the attribute, [redacted]must be qualified by the n

❷ [redacted]y itself has not been defined in this namespace; it exists only in the context of [redacted]

❸ This syntax imports the attribute [redacted]rom the [redacted]module directly into the local namespace.

❹ Now [redacted]an be accessed directly, without reference to [redacted]

When should you use [redacted]

If you will be accessing attributes and methods often and don't want to type the module name over and over, use [redacted]
If you want to selectively import some attributes and methods but not others, use [redacted]
If the module contains attributes or functions with the same name as ones in your module, you must use [redacted]o avoid name conflicts.·

Other than that, it's just a matter of style, and you will see Python code written both ways.

Use [redacted]paringly, because it makes it difficult to determine where a particular function or attribute came from, and that makes debugging and re

Further Reading on Module Importing Techniques

eff-bot (http://www.effbot.org/guides/) has more to say on [redacted]*vs.*[redacted]http://www.effbot.org/guides/import-confusion.htm).·
Python Tutorial (http://www.python.org/doc/current/tut/tut.html) discusses advanced import techniques, including [redacted]http://www.pyt

5.3. Defining Classes

Python is fully object-oriented: you can define your own classes, inherit from your own or built-in classes, and instantiate the classes you've

Defining a class in Python is simple. As with functions, there is no separate interface definition. Just define the class and start coding. A Pyth

Example 5.3. The Simplest Python Class

❶
❷❸

❶ The name of this class is [redacted]and it doesn't inherit from any other class. Class names are usually capitalized, [redacted]but this is only a conventio

❷ This class doesn't define any methods or attributes, but syntactically, there needs to be something in the definition, so you use [redacted]This is

❸ You probably guessed this, but everything in a class is indented, just like the code within a function, [redacted]statement, [redacted]oop, and so forth. Th

The statement in Python is like an empty set of braces (in Java or C.

Of course, realistically, most classes will be inherited from other classes, and they will define their own class methods and attributes. But as y

Example 5.4. Defining the Class

❶ In Python, the ancestor of a class is simply listed in parentheses immediately after the class name. So the class is inherited from the

In Python, the ancestor of a class is simply listed in parentheses immediately after the class name. There is no special keyword like in Java.

Python supports multiple inheritance. In the parentheses following the class name, you can list as many ancestor classes as you like, separated

5.3.1. Initializing and Coding Classes

This example shows the initialization of the class using the method.

Example 5.5. Initializing the Class

❶ Classes can (and should) have too, just like modules and functions.

❷ is called immediately after an instance of the class is created. It would be tempting but incorrect to call this the constructor of the clas

❸ The first argument of every class method, including is always a reference to the current instance of the class. By convention, this argu

❹ methods can take any number of arguments, and just like functions, the arguments can be defined with default values, making them o

By convention, the first argument of any Python class method (the reference to the current instance) is called This argument fills the role of

Example 5.6. Coding the Class

❶ Some pseudo-object-oriented languages like Powerbuilder have a concept of "extending" constructors and other events, where the anc

❷ I told you that this class acts like a dictionary, and here is the first sign of it. You’re assigning the argument as the value of this object

❸ Note that the method never returns a value.

5.3.2. Knowing When to Use and

When defining your class methods, you *must* explicitly list as the first argument for each method, including When you call a method of an a

Whew. I realize that's a lot to absorb, but you'll get the hang of it. All Python classes work the same way, so once you learn one, you've learn

methods are optional, but when you define one, you must remember to explicitly call the ancestor's method (if it defines one). This is more

Further Reading on Python Classes

Learning to Program (http://www.freenetpages.co.uk/hp/alan.gauld/) has a gentler introduction to classes (http://www.freenetpages.
How to Think Like a Computer Scientist (http://www.ibiblio.org/obp/thinkCSpy/) shows how to use classes to model compound data
Python Tutorial (http://www.python.org/doc/current/tut/tut.html) has an in-depth look at classes, namespaces, and inheritance (http:/
Python Knowledge Base (http://www.faqts.com/knowledge-base/index.phtml/fid/199/) answers common questions about classes (ht

5.4. Instantiating Classes

Instantiating classes in Python is straightforward. To instantiate a class, simply call the class as if it were a function, passing the arguments th

Example 5.7. Creating a Instance

❶ You are creating an instance of the class (defined in the module) and assigning the newly created instance to the variable . You are pa
❷ Every class instance has a built-in attribute, which is the object's class. (Note that the representation of this includes the physical add
❸ You can access the instance's just as with a function or a module. All instances of a class share the same
❹ Remember when the method assigned its argument to Well, here's the result. The arguments you pass when you create the class ins

In Python, simply call a class as if it were a function to create a new instance of the class. There is no explicit operator like C++ or Java.

5.4.1. Garbage Collection

If creating new instances is easy, destroying them is even easier. In general, there is no need to explicitly free instances, because they are free

Example 5.8. Trying to Implement a Memory Leak

❶ Every time the function is called, you are creating an instance of and assigning it to the variable , which is a local variable within the

❷ No matter how many times you call the function, it will never leak memory, because every time, Python will destroy the newly create

The technical term for this form of garbage collection is "reference counting". Python keeps a list of references to every instance created. In t

In previous versions of Python, there were situations where reference counting failed, and Python couldn't clean up after you. If you created t

As a former philosophy major, it disturbs me to think that things disappear when no one is looking at them, but that's exactly what happens in

Further Reading on Garbage Collection

Python Library Reference (http://www.python.org/doc/current/lib/) summarizes built-in attributes like (http://www.python.org/doc/
Python Library Reference (http://www.python.org/doc/current/lib/) documents the module (http://www.python.org/doc/current/lib/m

5.5. Exploring A Wrapper Class

As you've seen, is a class that acts like a dictionary. To explore this further, let's look at the class in the module, which is the ancestor of th

In the ActivePython IDE on Windows, you can quickly open any module in your library path by selecting File->Locate... (**Ctrl-L**).

Example 5.9. Defining the Class

❶
❷
❸
❹❺

❶ Note that is a base class, not inherited from any other class.
❷ This is the method that you overrode in the class. Note that the argument list in this ancestor class is different than the descendant. Th
❸ Python supports data attributes (called "instance variables" in Java and Powerbuilder, and "member variables" in C++). Data attributes
❹ The method is a dictionary duplicator: it copies all the keys and values from one dictionary to another. This does *not* clear the target d
❺ This is a syntax you may not have seen before (I haven't used it in the examples in this book). It's an statement, but instead of having

Java and Powerbuilder support function overloading by argument list, *i.e.* one class can have multiple methods with the same name but a diff

Guido, the original author of Python, explains method overriding this way: "Derived classes may override methods of their base classes. Bec

Always assign an initial value to all of an instance's data attributes in the method. It will save you hours of debugging later, tracking down

Example 5.10. Normal Methods

❶
❷❸
❹
❺
❻

1. is a normal class method; it is publicly available to be called by anyone at any time. Notice that like all class methods, has as its firs
2. The method of a real dictionary returns a new dictionary that is an exact duplicate of the original (all the same key-value pairs). But
3. You use the attribute to see if is a if so, you're golden, because you know how to copy a just create a new and give it the real dicti
4. If is not then must be some subclass of like maybe in which case life gets trickier. doesn't know how to make an exact copy of
5. The rest of the methods are straightforward, redirecting the calls to the built-in methods on

In versions of Python prior to 2.2, you could not directly subclass built-in datatypes like strings, lists, and dictionaries. To compensate for thi

In Python, you can inherit directly from the built-in datatype, as shown in this example. There are three differences here compared to the ve

Example 5.11. Inheriting Directly from Built-In Datatype

1. The first difference is that you don't need to import the module, since is a built-in datatype and is always available. The second is tha
2. The third difference is subtle but important. Because of the way works internally, it requires you to manually call its method to prope

Further Reading on

Python Library Reference (http://www.python.org/doc/current/lib/) documents the module (http://www.python.org/doc/current/lib/

5.6. Special Class Methods

In addition to normal class methods, there are a number of special methods that Python classes can define. Instead of being called directly by

As you saw in the previous section, normal methods go a long way towards wrapping a dictionary in a class. But normal methods alone are n

5.6.1. Getting and Setting Items

Example 5.12. The Special Method

1. The special method looks simple enough. Like the normal methods and it just redirects to the dictionary to return its value. But ho
2. This looks just like the syntax you would use to get a dictionary value, and in fact it returns the value you would expect. But here's the

Of course, Python has a special method to go along with as shown in the next example.

Example 5.13. The Special Method

```
[
[
[
[
[
[
[
```

1. Like the [method, [simply redirects to the real dictionary [to do its work. And like [you wouldn't ordinarily call it directly like this; Py
2. This looks like regular dictionary syntax, except of course that [is really a class that's trying very hard to masquerade as a dictionary, a [is a special class method because it gets called for you, but it's still a class method. Just as easily as the [method was defined in [you can red

This concept is the basis of the entire framework you're studying in this chapter. Each file type can have a handler class that knows how to ge

For example, [is a descendant of [When an [is set, it doesn't just set the [key (like the ancestor [does); it also looks in the file itself for MP3

Example 5.14. Overriding [in [

```
[
[
[
[
```

1. Notice that this [method is defined exactly the same way as the ancestor method. This is important, since Python will be calling the me
2. Here's the crux of the entire [class: if you're assigning a value to the [key, you want to do something extra.
3. The extra processing you do for [is encapsulated in the [method. This is another class method defined in [and when you call it, you qu
4. After doing this extra processing, you want to call the ancestor method. Remember that this is never done for you in Python; you must

When accessing data attributes within a class, you need to qualify the attribute name: [When calling other methods within a class, you need t

Example 5.15. Setting an [s [

```
[
[
[
[
[
[
[
```

2. Now the real fun begins. Setting the key of triggers the method on not which notices that you're setting the key with a real value
3. Modifying the key will go through the same process again: Python calls which calls which sets all the other keys.

5.7. Advanced Special Class Methods

Python has more special methods than just and Some of them let you emulate functionality that you may not even know about.

This example shows some of the other special methods in

Example 5.16. More Special Methods in

1. is a special method that is called when you call The function is a built-in function that returns a string representation of an object. It
2. is called when you compare class instances. In general, you can compare any two Python objects, not just class instances, by using
3. is called when you call The function is a built-in function that returns the length of an object. It works on any object that could reasc
4. is called when you call which you may remember as the way to delete individual items from a dictionary. When you use on a class i

In Java, you determine whether two string variables reference the same physical memory location by using This is called *object identity*, an

At this point, you may be thinking, "All this work just to do something in a class that I can do with a built-in datatype." And it's true that life

Special methods mean that *any class* can store key/value pairs like a dictionary, just by defining the method. *Any class* can act like a sequenc

While other object-oriented languages only let you define the physical model of an object ("this object has a method"), Python's special clas

Python has a lot of other special methods. There's a whole set of them that let classes act like numbers, allowing you to add, subtract, and do

Further Reading on Special Class Methods

Python Reference Manual (http://www.python.org/doc/current/ref/) documents all the special class methods (http://www.python.org

5.8. Introducing Class Attributes

You already know about data attributes, which are variables owned by a specific instance of a class. Python also supports class attributes, wh

Example 5.17. Introducing Class Attributes

❶ [redacted]s the class itself, not any particular instance of the class.

❷ [redacted]s a class attribute: literally, an attribute of the class. It is available before creating any instances of the class.

❸ Class attributes are available both through direct reference to the class and through any instance of the class.

In Java, both static variables (called class attributes in Python) and instance variables (called data attributes in Python) are defined immediate

Class attributes can be used as class-level constants (which is how you use them in [redacted] but they are not really constants. You can also change th

There are no constants in Python. Everything can be changed if you try hard enough. This fits with one of the core principles of Python: bad

Example 5.18. Modifying Class Attributes

[redacted]

❶ [redacted]

❷ [redacted]

❸ [redacted]

```
0
```

❹ [redacted]

```
1
[redacted]
1
```

```
2
2
2
```

1. is a class attribute of the class.
2. is a built-in attribute of every class instance (of every class). It is a reference to the class that is an instance of (in this case, the class)
3. Because is a class attribute, it is available through direct reference to the class, before you have created any instances of the class.
4. Creating an instance of the class calls the method, which increments the class attribute by 1This affects the class itself, not just the ne
5. Creating a second instance will increment the class attribute again. Notice how the class attribute is shared by the class and all instanc

5.9. Private Functions

Like most languages, Python has the concept of private elements:

- Private functions, which can't be called from outside their module·
- Private class methods, which can't be called from outside their class·
- Private attributes, which can't be accessed from outside their class.·

Unlike in most languages, whether a Python function, method, or attribute is private or public is determined entirely by its name.

If the name of a Python function, class method, or attribute starts with (but doesn't end with) two underscores, it's private; everything else is

In there are two methods: and As you have already discussed, is a special method; normally, you would call it indirectly by using the dict

In Python, all special methods (like and built-in attributes (like follow a standard naming convention: they both start with and end with tw

Example 5.19. Trying to Call a Private Method

```
1
```

1. If you try to call a private method, Python will raise a slightly misleading exception, saying that the method does not exist. Of course i

Further Reading on Private Functions

Python Tutorial (http://www.python.org/doc/current/tut/tut.html) discusses the inner workings of private variables (http://www.pyth

5.10. Summary

That's it for the hard-core object trickery. You'll see a real-world application of special class methods in Chapter 12, which uses to create a p

The next chapter will continue using this code sample to explore other Python concepts, such as exceptions, file objects, and loops.

Before diving into the next chapter, make sure you're comfortable doing all of these things:

Importing modules using either [illegible] or [illegible]
Defining and instantiating classes·
Defining [illegible]methods and other special class methods, and understanding when they are called·
Subclassing [illegible]o define classes that act like dictionaries·
Defining data attributes and class attributes, and understanding the differences between them·
Defining private attributes and methods·

Chapter 6. Exceptions and File Handling

In this chapter, you will dive into exceptions, file objects, [illegible]oops, and the [illegible]and [illegible]modules. If you've used exceptions in another programming la

6.1. Handling Exceptions

Like many other programming languages, Python has exception handling via [illegible]locks.

Python uses [illegible]o handle exceptions and [illegible]o generate them. Java and C++ use [illegible]o handle exceptions, and [illegible]o generate them.

Exceptions are everywhere in Python. Virtually every module in the standard Python library uses them, and Python itself will raise them in a

- Accessing a non-existent dictionary key will raise a [illegible]xception.
- Searching a list for a non-existent value will raise a [illegible]xception.
- Calling a non-existent method will raise an [illegible]xception.
- Referencing a non-existent variable will raise a [illegible]xception.
- Mixing datatypes without coercion will raise a [illegible]xception.

In each of these cases, you were simply playing around in the Python IDE: an error occurred, the exception was printed (depending on your I

An exception doesn't need result in a complete program crash, though. Exceptions, when raised, can be *handled*. Sometimes an exception is r

Example 6.1. Opening a Non-Existent File

❶ Using the built-in [illegible]unction, you can try to open a file for reading (more on [illegible]n the next section). But the file doesn't exist, so this raise

❷ You're trying to open the same non-existent file, but this time you're doing it within a [illegible]lock.

❸ When the [illegible]method raises an [illegible]xception, you're ready for it. The [illegible]ine catches the exception and executes your own block of code, whic

❹ Once an exception has been handled, processing continues normally on the first line after the [illegible]lock. Note that this line will always pri

Exceptions may seem unfriendly (after all, if you don't catch the exception, your entire program will crash), but consider the alternative. Wou

6.1.1. Using Exceptions For Other Purposes

There are a lot of other uses for exceptions besides handling actual error conditions. A common use in the standard Python library is to try to

You can also define your own exceptions by creating a class that inherits from the built-in [illegible]lass, and then raise your exceptions with the [illegible]on

The next example demonstrates how to use an exception to support platform-specific functionality. This code comes from the [illegible]module, a wra

Example 6.2. Supporting Platform-Specific Functionality



1. is a UNIX-specific module that provides low-level control over the input terminal. If this module is not available (because it's not on
2. OK, you didn't have so let's try which is a Windows-specific module that provides an API to many useful functions in the Microsof
3. If the first two didn't work, you try to import a function from which is a Mac OS-specific module that provides functions to pop up d
4. None of these platform-specific modules is available (which is possible, since Python has been ported to a lot of different platforms), s
5. A block can have an clause, like an statement. If no exception is raised during the block, the clause is executed afterwards. In this c

Further Reading on Exception Handling

Python Tutorial (http://www.python.org/doc/current/tut/tut.html) discusses defining and raising your own exceptions, and handling
Python Library Reference (http://www.python.org/doc/current/lib/) summarizes all the built-in exceptions (http://www.python.org/d
Python Library Reference (http://www.python.org/doc/current/lib/) documents the getpass (http://www.python.org/doc/current/lib/m
Python Library Reference (http://www.python.org/doc/current/lib/) documents the module (http://www.python.org/doc/current/lib/
Python Reference Manual (http://www.python.org/doc/current/ref/) discusses the inner workings of the block (http://www.python.o

6.2. Working with File Objects

Python has a built-in function, for opening a file on disk. returns a file object, which has methods and attributes for getting information abo

Example 6.3. Opening a File



1. The method can take up to three parameters: a filename, a mode, and a buffering parameter. Only the first one, the filename, is require
2. The function returns an object (by now, this should not surprise you). A file object has several useful attributes.
3. The attribute of a file object tells you in which mode the file was opened.
4. The attribute of a file object tells you the name of the file that the file object has open.

6.2.1. Reading Files

After you open a file, the first thing you'll want to do is read from it, as shown in the next example.

Example 6.4. Reading a File

```
>
[illegible]
❶
0
❷
❸
[illegible]
❹
[illegible]
[illegible]
[illegible]
❺
[illegible]
```

❶ A file object maintains state about the file it has open. The [illegible]method of a file object tells you your current position in the open file. Sinc

❷ The [illegible]method of a file object moves to another position in the open file. The second parameter specifies what the first one means; [illegible]mea

❸ The [illegible]method confirms that the current file position has moved.

❹ The [illegible]method reads a specified number of bytes from the open file and returns a string with the data that was read. The optional parame

❺ The [illegible]method confirms that the current position has moved. If you do the math, you'll see that after reading 128 bytes, the position has

6.2.2. Closing Files

Open files consume system resources, and depending on the file mode, other programs may not be able to access them. It's important to close

Example 6.5. Closing a File

```
>
[illegible]
❶
[illegible]
❷
>
[illegible]
❸
[illegible]
❹
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
```

[illegible]

1. The attribute of a file object indicates whether the object has a file open or not. In this case, the file is still open (as
2. To close a file, call the method of the file object. This frees the lock (if any) that you were holding on the file, flushes buffered writes
3. The attribute confirms that the file is closed.
4. Just because a file is closed doesn't mean that the file object ceases to exist. The variable will continue to exist until it goes out of sco
5. Calling on a file object whose file is already closed does *not* raise an exception; it fails silently.

6.2.3. Handling I/O Errors

Now you've seen enough to understand the file handling code in the ample code from teh previous chapter. This example shows how to saf

Example 6.6. File Objects in

[illegible]

1. Because opening and reading files is risky and may raise an exception, all of this code is wrapped in a block. (Hey, isn't standardized
2. The function may raise an (Maybe the file doesn't exist.)
3. The method may raise an (Maybe the file is smaller than 128 bytes.)
4. The method may raise an (Maybe the disk has a bad sector, or it's on a network drive and the network just went down.)
5. This is new: a block. Once the file has been opened successfully by the function, you want to make absolutely sure that you close it,
6. At last, you handle your exception. This could be the exception raised by the call to or Here, you really don't care, because all yo

6.2.4. Writing to Files

As you would expect, you can also write to files in much the same way that you read from them. There are two basic file modes:

"Append" mode will add data to the end of the file.·
"write" mode will overwrite the file.·

Either mode will create the file automatically if it doesn't already exist, so there's never a need for any sort of fiddly "if the log file doesn't ex

Example 6.7. Writing to Files

[illegible]

❶ You start boldly by creating either the new file or overwrites the existing file, and opening the file for writing. (The second parameter

❷ You can add data to the newly opened file with the method of the file object returned by

❸ is a synonym for This one-liner opens the file, reads its contents, and prints them.

❹ You happen to know that exists (since you just finished writing to it), so you can open it and append to it. (The parameter means ope

❺ As you can see, both the original line you wrote and the second line you appended are now in Also note that carriage returns are not i

Further Reading on File Handling

Python Tutorial (http://www.python.org/doc/current/tut/tut.html) discusses reading and writing files, including how to read a file on
eff-bot (http://www.effbot.org/guides/) discusses efficiency and performance of various ways of reading a file (http://www.effbot.or
Python Knowledge Base (http://www.faqts.com/knowledge-base/index.phtml/fid/199/) answers common questions about files (http:
Python Library Reference (http://www.python.org/doc/current/lib/) summarizes all the file object methods (http://www.python.org/d

6.3. Iterating with Loops

Like most other languages, Python has loops. The only reason you haven't seen them until now is that Python is good at so many other thing

Most other languages don't have a powerful list datatype like Python, so you end up doing a lot of manual work, specifying a start, end, and s

Example 6.8. Introducing the Loop

a
b
e
a
b
e

❶ The syntax for a loop is similar to list comprehensions. is a list, and will take the value of each element in turn, starting from the firs

❷ Like an statement or any other indented block, a loop can have any number of lines of code in it.

❸ This is the reason you haven't seen the loop yet: you haven't needed it yet. It's amazing how often you use loops in other languages

Doing a "normal" (by Visual Basic standards) counter loop is also simple.

Example 6.9. Simple Counters

```
0
1
2
3
4
[illegible]
[illegible]
[illegible]
a
b
c
d
e
```

1. As you saw in Example 3.20, “Assigning Consecutive Values”, [illegible] produces a list of integers, which you then loop through. I know it loo
2. Don’t ever do this. This is Visual Basic-style thinking. Break out of it. Just iterate through the list, as shown in the previous example.

[illegible]oops are not just for simple counters. They can iterate through all kinds of things. Here is an example of using a [illegible]oop to iterate through a di

Example 6.10. Iterating Through a Dictionary

```
[illegible]
[illegible][illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]

[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]

[illegible]
```

1. [illegible]s a dictionary of the environment variables defined on your system. In Windows, these are your user and system variables accessible
2. [illegible]eturns a list of tuples: [illegible]The [illegible]oop iterates through this list. The first round, it assigns [illegible]o [illegible]and [illegible]o [illegible]so [illegible]= [illegible]and [illegible]= [illegible]In the second round, [illegible]
3. With multi-variable assignment and list comprehensions, you can replace the entire [illegible]oop with a single statement. Whether you actuall

Now we can look at the [illegible]oop in [illegible]from the sample [illegible]rogram introduced in Chapter 5.

Example 6.11. [illegible]Loop in [illegible]

```
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
```

```
.
.
.
```

1. is a class attribute that defines the tags you're looking for in an MP3 file. Tags are stored in fixed-length fields. Once you read the last
2. This looks complicated, but it's not. The structure of the variables matches the structure of the elements of the list returned by Reme
3. Now that you've extracted all the parameters for a single MP3 tag, saving the tag data is easy. You slice from to to get the actual data

6.4. Using

Modules, like everything else in Python, are objects. Once imported, you can always get a reference to a module through the global dictionary

Example 6.12. Introducing

```
```

1. The module contains system-level information, such as the version of Python you're running (or and system-level options such as t
2. is a dictionary containing all the modules that have ever been imported since Python was started; the key is the module name, the valu

This example demonstrates how to use

Example 6.13. Using

```
```

❶ As new modules are imported, they are added to This explains why importing the same module twice is very fast: Python has already

❷ Given the name (as a string) of any previously-imported module, you can get a reference to the module itself through the dictionary.

The next example shows how to use the class attribute with the dictionary to get a reference to the module in which a class is defined.

Example 6.14. The Class Attribute

❶ Every Python class has a built-in class attribute which is the name of the module in which the class is defined.

❷ Combining this with the dictionary, you can get a reference to the module in which a class is defined.

Now you're ready to see how is used in the sample program introduced in Chapter 5. This example shows that portion of the code.

Example 6.15. In

❶ This is a function with two arguments; is required, but is optional and defaults to the module that contains the class. This looks ineff

❷ You'll plow through this line later, after you dive into the module. For now, take it on faith that ends up as the name of a class, like

❸ You already know about which gets a reference to an object by name. is a complementary function that checks whether an object ha

Further Reading on Modules

Python Tutorial (http://www.python.org/doc/current/tut/tut.html) discusses exactly when and how default arguments are evaluated (

Python Library Reference (http://www.python.org/doc/current/lib/) documents the (http://www.python.org/doc/current/lib/module-

6.5. Working with Directories

The module has several functions for manipulating files and directories. Here, we're looking at handling pathnames and listing the contents

Example 6.16. Constructing Pathnames

❶❷

```
```

1. s a reference to a module -- which module depends on your platform. Just as ncapsulates differences between platforms by setting
2. The unction of onstructs a pathname out of one or more partial pathnames. In this case, it simply concatenates strings. (Note that de
3. In this slightly less trivial case, will add an extra backslash to the pathname before joining it to the filename. I was overjoyed when I
4. will expand a pathname that uses to represent the current user's home directory. This works on any platform where users have a hom
5. Combining these techniques, you can easily construct pathnames for directories and files under the user's home directory.

Example 6.17. Splitting Pathnames

```
```

1. The unction splits a full pathname and returns a tuple containing the path and filename. Remember when I said you could use multi-
2. You assign the return value of the unction into a tuple of two variables. Each variable receives the value of the corresponding elemen
3. The first variable, receives the value of the first element of the tuple returned from the file path.
4. The second variable, receives the value of the second element of the tuple returned from the filename.
5. also contains a function which splits a filename and returns a tuple containing the filename and the file extension. You use the same

Example 6.18. Listing Directories

```
```

```
```

1. The function takes a pathname and returns a list of the contents of the directory.
2. returns both files and folders, with no indication of which is which.
3. You can use list filtering and the function of the module to separate the files from the folders. takes a pathname and returns 1 if the p
4. also has a function which returns 1 if the path represents a directory, and 0 otherwise. You can use this to get a list of the subdirectori

Example 6.19. Listing Directories in

```
```

1. returns a list of all the files and folders in
2. Iterating through the list with you use to normalize the case according to operating system defaults. is a useful little function that co
3. Iterating through the normalized list with again, you use to split each filename into name and extension.
4. For each file, you see if the extension is in the list of file extensions you care about (which was passed to the function).
5. For each file you care about, you use to construct the full pathname of the file, and return a list of the full pathnames.

Whenever possible, you should use the functions in and for file, directory, and path manipulations. These modules are wrappers for platform

There is one other way to get the contents of a directory. It's very powerful, and it uses the sort of wildcards that you may already be familiar

Example 6.20. Listing Directories with

```
```

❶ As you saw earlier, simply takes a directory path and lists all files and directories in that directory.

❷ The module, on the other hand, takes a wildcard and returns the full path of all files and directories matching the wildcard. Here the w

❸ If you want to find all the files in a specific directory that start with "s" and end with ".mp3", you can do that too.

❹ Now consider this scenario: you have a directory, with several subdirectories within it, with files within each subdirectory. You can g

Further Reading on the Module

Python Knowledge Base (http://www.faqts.com/knowledge-base/index.phtml/fid/199/) answers questions about the module (http://

Python Library Reference (http://www.python.org/doc/current/lib/) documents the (http://www.python.org/doc/current/lib/module-

6.6. Putting It All Together

Once again, all the dominoes are in place. You've seen how each line of code works. Now let's step back and see how it all fits together.

Example 6.21.

❶ is the main attraction of this entire module. It takes a directory (like in my case) and a list of interesting file extensions (like and it r

❷ As you saw in the previous section, this line of code gets a list of the full pathnames of all the files in that have an interesting file exte

❸ Old-school Pascal programmers may be familiar with them, but most people give me a blank stare when I tell them that Python suppor

❹ Now that you've seen the module, this line should make more sense. It gets the extension of the file (forces it to uppercase (slices

❺ Having constructed the name of the handler class that would handle this file, you check to see if that handler class actually exists in thi

❻ For each file in the "interesting files" list (you call with the filename (. Calling returns a class; you don't know exactly which class

Note that is completely generic. It doesn't know ahead of time which types of files it will be getting, or which classes are defined that could

6.7. Summary

The program introduced in Chapter 5 should now make perfect sense.

.

Before diving into the next chapter, make sure you're comfortable doing the following things:

- Catching exceptions with `try..except`
- Protecting external resources with `try..finally`
- Reading from files
- Assigning multiple values at once in a `for` loop
- Using the `os` module for all your cross-platform file manipulation needs
- Dynamically instantiating classes of unknown type by treating classes as objects and passing them around

Chapter 7. Regular Expressions

Regular expressions are a powerful and standardized way of searching, replacing, and parsing text with complex patterns of characters. If you

7.1. Diving In

Strings have methods for searching (, and) replacing (, and parsing (, but they are limited to the simplest of cases. The search methods loo

If what you're trying to do can be accomplished with string functions, you should use them. They're fast and simple and easy to read, and the

Although the regular expression syntax is tight and unlike normal code, the result can end up being *more* readable than a hand-rolled solution

7.2. Case Study: Street Addresses

This series of examples was inspired by a real-life problem I had in my day job several years ago, when I needed to scrub and standardize str

Example 7.1. Matching at the End of a String

```
1
2
3
4
5
6
```

1. My goal is to standardize a street address so that s always abbreviated as At first glance, I thought this was simple enough that I cou
2. Life, unfortunately, is full of counterexamples, and I quickly discovered this one. The problem here is that appears twice in the addres
3. To solve the problem of addresses with more than one substring, you could resort to something like this: only search and replace n th
4. It's time to move up to regular expressions. In Python, all functionality related to regular expressions is contained in the module.
5. Take a look at the first parameter: This is a simple regular expression that matches only when it occurs at the end of a string. The $n
6. Using the function, you search the string for the regular expression and replace it with This matches the t the end of the string, bu

Continuing with my story of scrubbing addresses, I soon discovered that the previous example, matching t the end of the address, was not g

Example 7.2. Matching Whole Words

```
1
2
3
4
5
6
7
8
9
```

What I *really* wanted was to match when it was at the end of the string *and* it was its own whole word, not a part of some larger word

To work around the backslash plague, you can use what is called a raw string, by prefixing the string with the letter . This tells Python

sigh Unfortunately, I soon found more cases that contradicted my logic. In this case, the street address contained the word as a who

To solve this problem, I removed the character and added another . Now the regular expression reads "match when it's a whole word

7.3. Case Study: Roman Numerals

You've most likely seen Roman numerals, even if you didn't recognize them. You may have seen them in copyrights of old movies and telev

In Roman numerals, there are seven characters that are repeated and combined in various ways to represent numbers.

```
I = 1
V = 5
X = 10
L = 50
C = 100
D = 500
M = 1000
```

The following are some general rules for constructing Roman numerals:

Characters are additive. is 1, is 2, and is 3. is 6 (literally, "5 and 1"), is 7, and is 8.
The tens characters (, , and) can be repeated up to three times. At 4, you need to subtract from the next highest fives character. You
Similarly, at 9, you need to subtract from the next highest tens character: 8 is , but 9 is (10 less than 0), not (since the character can not be
The fives characters can not be repeated. The number 0 is always represented as , never as . The number 0 is always , never .
Roman numerals are always written highest to lowest, and read left to right, so the order the of characters matters very much. is 0, 0

7.3.1. Checking for Thousands

What would it take to validate that an arbitrary string is a valid Roman numeral? Let's take it one digit at a time. Since Roman numerals are a

Example 7.3. Checking for Thousands

```
```

This pattern has three parts:
    to match what follows only at the beginning of the string. If this were not specified, the pattern would match no matter where

M to optionally match a single M character. Since this is repeated three times, you're matching anywhere from zero to three M char
$ to match what precedes only at the end of the string. When combined with the ^ character at the beginning, this means that the

❷ The essence of the e module is the b function, that takes a regular expression (p) and a string (M) to try to match against the regular expressi

❸ M matches because the first and second optional M characters match and the third M is ignored.

❹ M matches because all three M characters match.

❺ M does not match. All three M characters match, but then the regular expression insists on the string ending (because of the $ character), an

❻ Interestingly, an empty string also matches this regular expression, since all the M characters are optional.

7.3.2. Checking for Hundreds

The hundreds place is more difficult than the thousands, because there are several mutually exclusive ways it could be expressed, depending

0 = C
2 = C
3 = C
4 = D
5 = D
6 = D
7 = D
8 = D
9 = M

So there are four possible patterns:

M
D
Zero to three C characters (zero if the hundreds place is 0)·
D followed by zero to three C characters·

The last two patterns can be combined:

an optional D followed by zero to three C characters·

This example shows how to validate the hundreds place of a Roman numeral.

Example 7.4. Checking for Hundreds

```
[illegible]
```

❶ This pattern starts out the same as the previous one, checking for the beginning of the string (^), then the thousands place (M). Then it has

2. matches because the first matches, the second and third characters are ignored, and the matches (so the and patterns are never even
3. matches because the first matches, the second and third characters are ignored, and the pattern matches (each of the three character
4. matches because all three characters match, and the pattern matches (the s optional and is ignored). is the Roman numeral represe
5. does not match. The first matches, the second and third characters are ignored, and the matches, but then the does not match becaus
6. Interestingly, an empty string still matches this pattern, because all the characters are optional and ignored, and the empty string matc

Whew! See how quickly regular expressions can get nasty? And you've only covered the thousands and hundreds places of Roman numerals

7.4. Using the Syntax

In the previous section, you were dealing with a pattern where the same character could be repeated up to three times. There is another way to

Example 7.5. The Old Way: Every Character Optional

```
>
```

1. This matches the start of the string, and then the first optional but not the second and third (but that's okay because they're optional)
2. This matches the start of the string, and then the first and second optional but not the third (but that's okay because it's optional), and
3. This matches the start of the string, and then all three optional and then the end of the string.
4. This matches the start of the string, and then all three optional but then does not match the the end of the string (because there is still

Example 7.6. The New Way: From to

```
>
```

1. This pattern says: "Match the start of the string, then anywhere from zero to three characters, then the end of the string." The 0 and 3 c
2. This matches the start of the string, then one out of a possible three, then the end of the string.
3. This matches the start of the string, then two out of a possible three, then the end of the string.

4. This matches the start of the string, then three M out of a possible three, then the end of the string.
5. This matches the start of the string, then three M out of a possible three, but then *does not match* the end of the string. The regular expres

There is no way to programmatically determine that two regular expressions are equivalent. The best you can do is write a lot of test cases to

7.4.1. Checking for Tens and Ones

Now let's expand the Roman numeral regular expression to cover the tens and ones place. This example shows the check for tens.

Example 7.7. Checking for Tens

```
>
```

1. This matches the start of the string, then the first optional M then C then X then the end of the string. Remember, the $ syntax means "matc
2. This matches the start of the string, then the first optional M then C then X Of the X it matches the L and skips all three optional X characters.
3. This matches the start of the string, then the first optional M then C then the optional L and the first optional X skips the second and third op
4. This matches the start of the string, then the first optional M then C then the optional L and all three optional X characters, then the end of th
5. This matches the start of the string, then the first optional M then C then the optional L and all three optional X characters, then *fails to matc*

The expression for the ones place follows the same pattern. I'll spare you the details and show you the end result.

```
```

So what does that look like using this alternate {n} syntax? This example shows the new syntax.

Example 7.8. Validating Roman Numerals with {n}

```
```



1. This matches the start of the string, then one of a possible four characters, then . Of that, it matches the optional and zero of three pos
2. This matches the start of the string, then two of a possible four characters, then the with a and one of three possible characters; then
3. This matches the start of the string, then four out of four characters, then with a and three out of three characters; then with an and
4. Watch closely. (I feel like a magician. "Watch closely, kids, I'm going to pull a rabbit out of my hat.") This matches the start of the str

If you followed all that and understood it on the first try, you're doing better than I did. Now imagine trying to understand someone else's reg

In the next section you'll explore an alternate syntax that can help keep your expressions maintainable.

7.5. Verbose Regular Expressions

So far you've just been dealing with what I'll call "compact" regular expressions. As you've seen, they are difficult to read, and even if you f

Python allows you to do this with something called *verbose regular expressions*. A verbose regular expression is different from a compact reg

> Whitespace is ignored. Spaces, tabs, and carriage returns are not matched as spaces, tabs, and carriage returns. They're not matched
> Comments are ignored. A comment in a verbose regular expression is just like a comment in Python code: it starts with a character

This will be more clear with an example. Let's revisit the compact regular expression you've been working with, and make it a verbose regul

Example 7.9. Regular Expressions with Inline Comments



1. The most important thing to remember when using verbose regular expressions is that you need to pass an extra argument when worki
2. This matches the start of the string, then one of a possible four , then , then and three of a possible three , then then the end of the str
3. This matches the start of the string, then four of a possible four , then and three of a possible three , then and three of a possible three
4. This does not match. Why? Because it doesn't have the flag, so the function is treating the pattern as a compact regular expression, w

7.6. Case study: Parsing Phone Numbers

So far you've concentrated on matching whole patterns. Either the pattern matches, or it doesn't. But regular expressions are much more pow

This example came from another real-world problem I encountered, again from a previous day job. The problem: parsing an American phone

Here are the phone numbers I needed to be able to accept:

Quite a variety! In each of these cases, I need to know that the area code was 0, the trunk was 5, and the rest of the phone number was 2. For thos

Let's work through developing a solution for phone number parsing. This example shows the first step.

Example 7.10. Finding Numbers

```
>
```

1. Always read regular expressions from left to right. This one matches the beginning of the string, and then 3. What's 3? Well, the 3 means
2. To get access to the groups that the regular expression parser remembered along the way, use the method on the object that the functi
3. This regular expression is not the final answer, because it doesn't handle a phone number with an extension on the end. For that, you'l

Example 7.11. Finding the Extension

```
>
>
```

1. This regular expression is almost identical to the previous one. Just as before, you match the beginning of the string, then a remembere
2. The method now returns a tuple of four elements, since the regular expression now defines four groups to remember.
3. Unfortunately, this regular expression is not the final answer either, because it assumes that the different parts of the phone number are
4. Oops! Not only does this regular expression not do everything you want, it's actually a step backwards, because now you can't parse p

The next example shows the regular expression to handle separators between the different parts of the phone number.

Example 7.12. Handling Different Separators

```
>
>
```

1. Hang on to your hat. You're matching the beginning of the string, then a group of three digits, then What the heck is that? Well, hat
2. Using instead of means you can now match phone numbers where the parts are separated by spaces instead of hyphens.
3. Of course, phone numbers separated by hyphens still work too.
4. Unfortunately, this is still not the final answer, because it assumes that there is a separator at all. What if the phone number is entered
4. Oops! This still hasn't fixed the problem of requiring extensions. Now you have two problems, but you can solve both of them with th

The next example shows the regular expression for handling phone numbers *without* separators.

Example 7.13. Handling Numbers Without Separators

```
>
```

1. The only change you've made since that last step is changing all the to . Instead of between the parts of the phone number, you now r
2. Lo and behold, it actually works. Why? You matched the beginning of the string, then a remembered group of three digits (, then zero
3. Other variations work now too: dots instead of hyphens, and both a space and an x before the extension.
4. Finally, you've solved the other long-standing problem: extensions are optional again. If no extension is found, the method still return
5. I hate to be the bearer of bad news, but you're not finished yet. What's the problem here? There's an extra character before the area co

The next example shows how to handle leading characters in phone numbers.

Example 7.14. Handling Leading Characters

```
```

```
>
```

1. This is the same as in the previous example, except now you're matching zero or more non-numeric characters, before the first remer
2. You can successfully parse the phone number, even with the leading left parenthesis before the area code. (The right parenthesis after
3. Just a sanity check to make sure you haven't broken anything that used to work. Since the leading characters are entirely optional, this
4. This is where regular expressions make me want to gouge my eyes out with a blunt object. Why doesn't this phone number match? Be

Let's back up for a second. So far the regular expressions have all matched from the beginning of the string. But now you see that there may

Example 7.15. Phone Number, Wherever I May Find Ye

```

```

1. Note the lack of ^ in this regular expression. You are not matching the beginning of the string anymore. There's nothing that says you n
2. Now you can successfully parse a phone number that includes leading characters and a leading digit, plus any number of any kind of s
3. Sanity check. this still works.
4. That still works too.

See how quickly a regular expression can get out of control? Take a quick glance at any of the previous iterations. Can you tell the difference

While you still understand the final answer (and it is the final answer; if you've discovered a case it doesn't handle, I don't want to know abo

Example 7.16. Parsing Phone Numbers (Final Version)

```

```

❶ Other than being spread out over multiple lines, this is exactly the same regular expression as the last step, so it’s no surprise that it pa

❷ Final sanity check. Yes, this still works. You’re done.

Further Reading on Regular Expressions

Regular Expression HOWTO (http://py-howto.sourceforge.net/regex/regex.html) teaches about regular expressions and how to use t

Python Library Reference (http://www.python.org/doc/current/lib/) summarizes the re module (http://www.python.org/doc/current/lib

7.7. Summary

This is just the tiniest tip of the iceberg of what regular expressions can do. In other words, even though you’re completely overwhelmed by t

You should now be familiar with the following techniques:

- ^ matches the beginning of a string.
- $ matches the end of a string.
- \b matches a word boundary.
- \d matches any numeric digit.
- \D matches any non-numeric character.
- x? matches an optional x character (in other words, it matches an x zero or one times).
- x* matches x zero or more times.
- x+ matches x one or more times.
- x{n,m} matches an x character at least n times, but not more than m times.
- (a|b|c) matches either a or b or c
- (x) in general is a *remembered group*. You can get the value of what matched by using the groups() method of the object returned by re

Regular expressions are extremely powerful, but they are not the correct solution for every problem. You should learn enough about them to

> Some people, when confronted with a problem, think "I know, I’ll use regular expressions." Now they have two problems.
>
> --Jamie Zawinski, in c

Chapter 8. HTML Processing

8.1. Diving in

I often see questions on comp.lang.python (http://groups.google.com/groups?group=comp.lang.python) like "How can I list all the [headers|in

Here is a complete, working Python program in two parts. The first part, is a generic tool to help you process HTML files by walking throug

Example 8.1.

If you have not already done so, you can download this and other examples (http://diveintopython.org/download/diveintopython-examples-5.

Example 8.2.

Example 8.3. Output of

Running this script will translate Section 3.2, “Introducing Lists” into mock Swedish Chef-speak (../native_data_types/chef.html) (from The

8.2. Introducing

HTML processing is broken into three steps: breaking down the HTML into its constituent pieces, fiddling with the pieces, and reconstructing

The key to understanding this chapter is to realize that HTML is not just text, it is structured text. The structure is derived from the more-or-l

contains one important class: parses HTML into useful pieces, like start tags and end tags. As soon as it succeeds in breaking down some d
parses HTML into 8 kinds of data, and calls a separate method for each of them:

Start tag
: An HTML tag that starts a block, like or for a standalone tag like or When it finds a start tag will look for a method

End tag
: An HTML tag that ends a block, like or When it finds an end tag, will look for a method called If found, calls this m

Character reference

An escaped character referenced by its decimal or hexadecimal equivalent, like When found, calls with the text of the de

Entity reference

An HTML entity, like When found, calls with the name of the HTML entity.

Comment

An HTML comment, enclosed in When found, calls with the body of the comment.

Processing instruction

An HTML processing instruction, enclosed in When found, calls with the body of the processing instruction.

Declaration

An HTML declaration, such as a enclosed in When found, calls with the body of the declaration.

Text data

A block of text. Anything that doesn't fit into the other 7 categories. When found, calls with the text.

Python 2.0 had a bug where would not recognize declarations at all (would never be called), which meant that were silently ignored. This comes with a test suite to illustrate this. You can run passing the name of an HTML file on the command line, and it will print out the tags a

In the ActivePython IDE on Windows, you can specify command line arguments in the "Run script" dialog. Separate multiple arguments wit

Example 8.4. Sample test of

Here is a snippet from the table of contents of the HTML version of this book. Of course your paths may vary. (If you haven't downloaded th

Running this through the test suite of yields this output:

Here’s the roadmap for the rest of the chapter:

Subclass to create classes that extract interesting data out of HTML documents.·
Subclass to create which overrides all 8 handler methods and uses them to reconstruct the original HTML from the pieces.·
Subclass to create which adds some methods to process specific HTML tags specially, and overrides the method to provide a fram
Subclass to create classes that define text processing rules used by
Write a test suite that grabs a real web page from and processes it.·

Along the way, you’ll also learn about and dictionary-based string formatting.

8.3. Extracting data from HTML documents

To extract data from HTML documents, subclass the class and define methods for each tag or entity you want to capture.

The first step to extracting data from an HTML document is getting some HTML. If you have some HTML lying around on your hard drive,

Example 8.5. Introducing

```
1
2
3
4
5
```

1. The module is part of the standard Python library. It contains functions for getting information about and actually retrieving data from
2. The simplest use of is to retrieve the entire text of a web page using the function. Opening a URL is similar to opening a file. The retu
3. The simplest thing to do with the file-like object returned by is which reads the entire HTML of the web page into a single string. Th
4. When you’re done with the object, make sure to it, just like a normal file object.
5. You now have the complete HTML of the home page of in a string, and you’re ready to parse it.

Example 8.6. Introducing [redacted]

If you have not already done so, you can download this and other examples (http://diveintopython.org/download/diveintopython-examples-5.

[redacted]

[redacted]

❶ [redacted]s called by the [redacted]method of [redacted]and it can also be called manually once an instance of the parser has been created. So if you need to do an

❷ [redacted]s called by [redacted]whenever it finds an [redacted]ag. The tag may contain an [redacted]ttribute, and/or other attributes, like [redacted]or [redacted]The [redacted]parameter is a list of tu

❸ You can find out whether this [redacted]ag has an [redacted]ttribute with a simple multi-variablelist comprehension.

❹ String comparisons like [redacted]re always case-sensitive, but that's safe in this case, because [redacted]onverts attribute names to lowercase while bu

Example 8.7. Using [redacted]

[redacted]

[redacted]

❶ [redacted]ll the [redacted]method, which is [redacted]to [redacted]get HTML into the parser.

❷ Like files, you should [redacted]your URL objects as soon as you're done with them.

❸ You should [redacted]your parser object, too, but for a different reason. You've read all the data and fed it to the parser, but the [redacted]method isn't gu

❹ Once the parser is [redacted], the parsing is complete, and [redacted]ontains a list of all the linked URLs in the HTML document. (Your output may loo

8.4. Introducing

doesn't produce anything by itself. It parses and parses and parses, and it calls a method for each interesting thing it finds, but the methods d ubclasses and provides all 8 essential handler methods: and

Example 8.8. Introducing

1. called by initializes as an empty list before calling the ancestor method. s a data attribute which will hold the pieces of the HTML
2. Since loes not define any methods for specific tags (like the method in will call or every start tag. This method takes the tag (ar
3. Reconstructing end tags is much simpler; just take the tag name and wrap it in the rackets.
4. When inds a character reference, it calls with the bare reference. If the HTML document contains the reference will be Reconstru
5. Entity references are similar to character references, but without the hash mark. Reconstructing the original entity reference requires w
6. Blocks of text are simply appended to unaltered.
7. HTML comments are wrapped in haracters.
8. Processing instructions are wrapped in haracters.

The HTML specification requires that all non-HTML (like client-side JavaScript) must be enclosed in HTML comments, but not all web pag

Example 8.9. [redacted]output

[redacted]

❶ This is the one method in [redacted]hat is never called by the ancestor [redacted]Since the other handler methods store their reconstructed HTML in [redacted]th

❷ If you prefer, you could use the [redacted]method of the [redacted]module instead: [redacted]

Further reading

W3C (http://www.w3.org/) discusses character and entity references (http://www.w3.org/TR/REC-html40/charset.html#entities).·
Python Library Reference (http://www.python.org/doc/current/lib/) confirms your suspicions that the [redacted]module (http://www.python.o

8.5. [redacted]and [redacted]

Let's digress from HTML processing for a minute and talk about how Python handles variables. Python has two built-in functions, [redacted]and [redacted]whi

Remember [redacted] You first saw it here:

[redacted]

No, wait, you can't learn about [redacted]yet. First, you need to learn about namespaces. This is dry stuff, but it's important, so pay attention.

Python uses what are called namespaces to keep track of variables. A namespace is just like a dictionary where the keys are names of variabl

At any particular point in a Python program, there are several namespaces available. Each function has its own namespace, called the local na

When a line of code asks for the value of a variable [redacted]Python will search for that variable in all the available namespaces, in order:

local namespace - specific to the current function or class method. If the function defines a local variable [redacted]or has an argument [redacted]Pyth
global namespace - specific to the current module. If the module has defined a variable, function, or class called [redacted]Python will use th
built-in namespace - global to all modules. As a last resort, Python will assume that [redacted]is the name of built-in function or variable.3.

If Python doesn't find [redacted]in any of these namespaces, it gives up and raises a [redacted]with the message [redacted]which you saw back in Example 3.18, "Refere

Python 2.2 introduced a subtle but important change that affects the namespace search order: nested scopes. In versions of Python prior to 2.

[redacted]

Are you confused yet? Don't despair! This is really cool, I promise. Like many things in Python, namespaces are *directly accessible at run-ti*

Example 8.10. Introducing [redacted]

[redacted]

.

[redacted]

[redacted]

[redacted]

❶ The function has two variables in its local namespace: whose value is passed in to the function, and which is defined within the fun

❷ returns a dictionary of name/value pairs. The keys of this dictionary are the names of the variables as strings; the values of the diction

❸ Remember, Python has dynamic typing, so you could just as easily pass a string in for the function (and the call to would still work

What does for the module's namespace include? Module-level variables and constants, it includes all the functions, and classes defined in the

Remember the difference between and With the module itself is imported, but it retains its own namespace, which is why you need to use

Example 8.11. Introducing

Look at the following block of code at the bottom of

❶ Just so you don't get intimidated, remember that you've seen all this before. The function returns a dictionary, and you're iterating thr

Now running the script from the command line gives this output (note that your output may be slightly different, depending on your platform

❶ was imported from using That means that it was imported directly into the module's namespace, and here it is.

❷ Contrast this with which was imported using That means that the module itself is in the namespace, but the variable defined within

❸ This module only defines one class, and here it is. Note that the value here is the class itself, not a specific instance of the class.

❹ Remember the trick? When running a module (as opposed to importing it from another module), the built-in attribute is a special valu

Using the and functions, you can get the value of arbitrary variables dynamically, providing the variable name as a string. This mirrors the f

There is one other important difference between the and functions, which you should learn now before it bites you. It will bite you anyway,

Example 8.12. is read-only, is not

❶ ❷ ❸ ❹ ❺

❶ Since [illegible] is called with [illegible], this will print [illegible]. This should not be a surprise.

❷ [illegible] is a function that returns a dictionary, and here you are setting a value in that dictionary. You might think that this would change the v

❸ This prints [illegible] not [illegible].

❹ After being burned by [illegible], you might think that this *wouldn't* change the value of [illegible], but it does. Due to internal differences in how Python

❺ This prints [illegible] not [illegible].

8.6. Dictionary-based string formatting

Why did you learn about [illegible] and [illegible]? So you can learn about dictionary-based string formatting. As you recall, regular string formatting provides a

There is an alternative form of string formatting that uses dictionaries instead of tuples of values.

Example 8.13. Introducing dictionary-based string formatting

[illegible]

❶ Instead of a tuple of explicit values, this form of string formatting uses a dictionary, [illegible]. And instead of a simple [illegible] marker in the string, the

❷ Dictionary-based string formatting works with any number of named keys. Each key must exist in the given dictionary, or the formatti

❸ You can even specify the same key twice; each occurrence will be replaced with the same value.

So why would you use dictionary-based string formatting? Well, it does seem like overkill to set up a dictionary of keys and values simply to

Example 8.14. Dictionary-based string formatting in [illegible]

[illegible]

❶ Using the built-in [illegible] function is the most common use of dictionary-based string formatting. It means that you can use the names of loca

Example 8.15. More dictionary-based string formatting

[illegible]

❶ When this method is called, [illegible] is a list of key/value tuples, just like the [illegible] of a dictionary, which means you can use multi-variable assignm

Suppose [illegible] is [illegible].

In the first round of the list comprehension, [illegible] will get [illegible] and [illegible] will get [illegible].

The string formatting [illegible]

[illegible] will resolve to [illegible]

This string becomes the first element of the list comprehension's return value.

In the second round, [illegible] will get [illegible] and [illegible] will get [illegible].

The string formatting will resolve to [illegible].

The list comprehension returns a list of these two resolved strings, and [illegible] will join both elements of this list together to form [illegible]

❷ Now, using dictionary-based string formatting, you insert the value of [illegible] and [illegible] into a string. So if [illegible] is [illegible], the final result would be [illegible] and that i

Using dictionary-based string formatting with [illegible] is a convenient way of making complex string formatting expressions more readable, but it co

8.7. Quoting attribute values

[illegible] (This is generally pronounced by a project (http://groups.google.com/groups/sig-is-a-standard-religion) in the large project of HTML documen

[illegible] consumes HTML (since it's descended from [illegible] and produces equivalent HTML, but the HTML output is not identical to the input. Tags and a

Example 8.16. Quoting attribute values

❶ [illegible]

❷ [illegible]

❸ [illegible]

❶ Note that the attribute values of the [illegible] attributes in the [illegible] tags are not properly quoted. (Also note that you're using triple quotes for someth

❷ Feed the parser.

❸ Using the [illegible] function defined in [illegible] you get the output as a single string, complete with quoted attribute values. While this may seem anti-c

8.8. Introducing [illegible]

[illegible] is a simple (and silly) descendant of [illegible] It runs blocks of text through a series of substitutions, but it makes sure that anything within a [illegible] block p

To handle the [illegible] blocks, you define two methods in [illegible] and [illegible]

Example 8.17. Handling specific tags

❶ is called every time finds a tag in the HTML source. (In a minute, you'll see exactly how this happens.) The method takes a single pa

❷ In the method, you initialize a data attribute that serves as a counter for tags. Every time you hit a tag, you increment the counter; ev

❸ That's it, that's the only special processing you do for tags. Now you pass the list of attributes along to so it can do the default proces

❹ is called every time finds a tag. Since end tags can not contain attributes, the method takes no parameters.

❺ First, you want to do the default processing, just like any other end tag.

❻ Second, you decrement your counter to signal that this block has been closed.

At this point, it's worth digging a little further into I've claimed repeatedly (and you've taken it on faith so far) that looks for and calls spec

Example 8.18.

❶ At this point, has already found a start tag and parsed the attribute list. The only thing left to do is figure out whether there is a specifi

❷ The "magic" of is nothing more than your old friend, What you may not have realized before is that will find methods defined in de

❸ raises an if the method it's looking for doesn't exist in the object (or any of its descendants), but that's okay, because you wrapped th

❹ Since you didn't find a method, you'll also look for a method before giving up. This alternate naming scheme is generally used for st

❺ Another which means that the call to failed with Since you found neither a nor a method for this tag, you catch the exception and

❻ Remember, blocks can have an clause, which is called if no exception is raised during the block. Logically, that means that you *did* f

❼ By the way, don't worry about these different return values; in theory they mean something, but they're never actually used. Don't wo

❽ and methods are not called directly; the tag, method, and attributes are passed to this function, so that descendants can override it an

Now back to our regularly scheduled program: When you left, you were in the process of defining specific handler methods for and tags. T

Example 8.19. Overriding the method

1
2

1. is called with only one argument, the text to process.
2. In the ancestor the method simply appended the text to the output buffer, Here the logic is only slightly more complicated. If you're

You're close to completely understanding The only missing link is the nature of the text substitutions themselves. If you know any Perl, you

8.9. Putting it all together

It's time to put everything you've learned so far to good use. I hope you were paying attention.

Example 8.20. The function, part 1

1
2
1
1
1

1. The function has an optional argument which is a string that specifies the dialect you'll be using. You'll see how this is used in a min
2. Hey, wait a minute, there's an statement in this function! That's perfectly legal in Python. You're used to seeing statements at the top
3. Now you get the source of the given URL.

Example 8.21. The function, part 2: curiouser and curiouser

1
2
3

1. is a string method you haven't seen before; it simply capitalizes the first letter of a string and forces everything else to lowercase. Con
2. You have the name of a class as a string (and you have the global namespace as a dictionary ( ). Combined, you can get a reference
3. Finally, you have a class object (and you want an instance of the class. Well, you already know how to do that: call the class like a f

Why bother? After all, there are only 3 classes; why not just use a statement? (Well, there's no statement in Python, but why not just use a s

Even better, imagine putting in a separate module, and importing it with You've already seen that this includes it in , so would still work

Now imagine that the name of the dialect is coming from somewhere outside the program, maybe from a database or from a user-inputted val

Finally, imagine a framework with a plug-in architecture. You could put each class in a separate file, leaving only the function in Assumin

Example 8.22. The function, part 3

1
2
3

1. After all that imagining, this is going to seem pretty boring, but the function is what does the entire transformation. You had the entire
2. Because maintains an internal buffer, you should always call the parser's method when you're done (even if you fed it all at once, lik
3. Remember, is the function you defined on that joins all the pieces of output you've buffered and returns them in a single string.

And just like that, you've "translated" a web page, given nothing but a URL and the name of a dialect.

Further reading

You thought I was kidding about the server-side scripting idea. So did I, until I found this web-based dialectizer (http://rinkworks.co

8.10. Summary

Python provides you with a powerful tool, to manipulate HTML by turning its structure into an object model. You can use this tool in many

- parsing the HTML looking for something specific·
- aggregating the results, like the URL lister·
- altering the structure along the way, like the attribute quoter·
- transforming the HTML into something else by manipulating the text while leaving the tags alone, like the

Along with these examples, you should be comfortable doing all of the following things:

- Using and to access namespaces·
- Formatting strings using dictionary-based substitutions·

---

[1] The technical term for a parser like is a *consumer*: it consumes HTML and breaks it down. Presumably, the name was chosen to fit into the
[2] The reason Python is better at lists than strings is that lists are mutable but strings are immutable. This means that appending to a list just add
[3] I don't get out much.
[4] All right, it's not that common a question. It's not up there with "What editor should I use to write Python code?" (answer: Emacs) or "Is Pyt

Chapter 9. XML Processing

9.1. Diving in

These next two chapters are about XML processing in Python. It would be helpful if you already knew what an XML document looks like, th

If you’re not particularly interested in XML, you should still read these chapters, which cover important topics like Python packages, Unicod

Being a philosophy major is not required, although if you have ever had the misfortune of being subjected to the writings of Immanuel Kant,

There are two basic ways to work with XML. One is called SAX ("Simple API for XML"), and it works by reading the XML a little bit at a t

The following is a complete Python program which generates pseudo-random output based on a context-free grammar defined in an XML fo

Example 9.1.

If you have not already done so, you can download this and other examples (http://diveintopython.org/download/diveintopython-examples-5.

Example 9.2.

Run the program by itself, and it will parse the default XML-based grammar, in and print several paragraphs worth of philosophy in the sty

Example 9.3. Sample output of

This is, of course, complete gibberish. Well, not complete gibberish. It is syntactically and grammatically correct (although very verbose -- K

Let me repeat that this is much, much funnier if you are now or have ever been a philosophy major.

The interesting thing about this program is that there is nothing Kant-specific about it. All the content in the previous example was derived fr

Example 9.4. Simpler output from

You will take a closer look at the structure of the grammar file later in this chapter. For now, all you need to know is that the grammar file de

9.2. Packages

Actually parsing an XML document is very simple: one line of code. However, before you get to that line of code, you need to take a short de

Example 9.5. Loading an XML document (a sneak peek)

❶ This is a syntax you haven't seen before. It looks almost like the you know and love, but the gives it away as something above and be

That sounds complicated, but it's really not. Looking at the actual implementation may help. Packages are little more than directories of mod

Example 9.6. File layout of a package



So when you say Python figures out that that means "look in the directory for a directory, and look in *that* for the module, and import it as

Example 9.7. Packages are modules, too



❶ Here you're importing a module (from a nested package (The result is that s imported into your namespace, and in order to refere
❷ Here you are importing a class (from a module (from a nested package (The result is that s imported directly into your namespac
❸ Here you are importing the package (a nested package of as a module in and of itself. Any level of a package can be treated as a mo
❹ Here you are importing the root level package as a module.

So how can a package (which is just a directory on disk) be imported and treated as a module (which is always a file on disk)? The answer is

A package is a directory with the special file in it. The file defines the attributes and methods of the package. It doesn't need to define anyth

So why bother with packages? Well, they provide a way to logically group related modules. Instead of having an package with and package

If you ever find yourself writing a large subsystem in Python (or, more likely, when you realize that your small subsystem has grown into a la

9.3. Parsing XML

As I was saying, actually parsing an XML document is very simple: one line of code. Where you go from there is up to you.

Example 9.8. Loading an XML document (for real this time)



1. As you saw in the previous section, this imports the module from the package.
2. Here is the one line of code that does all the work: takes one argument and returns a parsed representation of the XML document. The
3. The object returned from is a object, a descendant of the class. This object is the root level of a complex tree-like structure of interlo
4. is a method of the class (and is therefore available on the object you got from . prints out the XML that this represents. For the no

Now that you have an XML document in memory, you can start traversing through it.

Example 9.9. Getting child nodes



1. Every has a attribute, which is a list of the objects. A always has only one child node, the root element of the XML document (in thi
2. To get the first (and in this case, the only) child node, just use regular list syntax. Remember, there is nothing special going on here; th
3. Since getting the first child node of a node is a useful and common activity, the class has a attribute, which is synonymous with (Th

Example 9.10. works on any node



```
```

❶ Since the method is defined in the class, it is available on any XML node, not just the element.

Example 9.11. Child nodes can be text

```
```

```
```

❶ Looking at the XML in you might think that the has only two child nodes, the two elements. But you're missing something: the carr
❷ The first child is a object representing the carriage return after the tag and before the first tag.
❸ The second child is an object representing the first element.
❹ The fourth child is an object representing the second element.
❺ The last child is a object representing the carriage return after the end tag and before the end tag.

Example 9.12. Drilling down all the way to text

```
```

```
```

1. The real advantage of unicode, of course, is its ability to store non-ASCII characters, like the Spanish "ñ" (n with a tilde over it). The u
2. Remember I said that the function attempts to convert a unicode string to ASCII so it can print it? Well, that's not going to work here,
3. Here's where the conversion-from-unicode-to-other-encoding-schemes comes in. is a unicode string, but can only print a regular strin

Remember I said Python usually converted unicode to ASCII whenever it needed to make a regular string out of a unicode string? Well, this

Example 9.15.

```
```

1. is a special script; Python will try to import it on startup, so any code in it will be run automatically. As the comment mentions, it can
2. function sets, well, the default encoding. This is the encoding scheme that Python will try to use whenever it needs to auto-coerce a u

Example 9.16. Effects of setting the default encoding

```
```

1. This example assumes that you have made the changes listed in the previous example to your file, and restarted Python. If your defau
2. Now that the default encoding scheme includes all the characters you use in your string, Python has no problem auto-coercing the strin

Example 9.17. Specifying encoding in files

If you are going to be storing non-ASCII strings within your Python code, you'll need to specify the encoding of each individual file by putti

```
```

Now, what about XML? Well, every XML document is in a specific encoding. Again, ISO-8859-1 is a popular encoding for data in Western

Example 9.18.

```
```

```
```

❶ This is a sample extract from a real Russian XML document; it's part of a Russian translation of this very book. Note the encoding, s
❷ These are Cyrillic characters which, as far as I know, spell the Russian word for "Preface". If you open this file in a regular text editor,

Example 9.19. Parsing

```
❶
❷
❸
❹
❺
```

Предисловие

❶ I'm assuming here that you saved the previous example as in the current directory. I am also, for the sake of completeness, assuming
❷ Note that the text data of the tag (now in the variable, thanks to that long concatenation of Python functions which I hastily skipped o
❸ Printing the title is not possible, because this unicode string contains non-ASCII characters, so Python can't convert it to ASCII becaus
❹ You can, however, explicitly convert it to in which case you get a (regular, not unicode) string of single-byte characters (0x25 and so f
❺ Printing the encoded string will probably show gibberish on your screen, because your Python IDE is interpreting those characters as

To sum up, unicode itself is a bit intimidating if you've never seen it before, but unicode data is really very easy to handle in Python. If your

Further reading

Unicode.org (http://www.unicode.org/) is the home page of the unicode standard, including a brief technical introduction (http://ww
Unicode Tutorial (http://www.reportlab.com/i18n/python_unicode_tutorial.html) has some more examples of how to use Python's u
PEP 263 (http://www.python.org/peps/pep-0263.html) goes into more detail about how and when to define a character encoding in y

9.5. Searching for elements

Traversing XML documents by stepping through each node can be tedious. If you're looking for something in particular, buried deep within

For this section, you'll be using the grammar file, which looks like this:

Example 9.20.

```
```

It has two [illegible], [illegible] and [illegible]A [illegible] is either a [illegible] or [illegible], and a [illegible] is 8 [illegible]

Example 9.21. Introducing [illegible]

❶ [illegible] takes one argument, the name of the element you wish to find. It returns a list of [illegible] objects, corresponding to the XML elements that ha

Example 9.22. Every element is searchable

❶ Continuing from the previous example, the first object in your [illegible] is the [illegible] element.

❷ You can use the same [illegible] method on this [illegible] to find all the [illegible] elements within the [illegible] element.

❸ Just as before, the [illegible] method returns a list of all the elements it found. In this case, you have two, one for each bit.

Example 9.23. Searching is actually recursive

```
```

❶ Note carefully the difference between this and the previous example. Previously, you were searching for elements within but here yo
❷ The first two elements are within the first the
❸ The last element is the one within the second the

9.6. Accessing element attributes

XML elements can have one or more attributes, and it is incredibly simple to access them once you have parsed an XML document.

For this section, you'll be using the grammar file that you saw in the previous section.

This section may be a little confusing, because of some overlapping terminology. Elements in an XML document have attributes, and Python

Example 9.24. Accessing element attributes

```
```

❶ Each object has an attribute called which is a object. This sounds scary, but it's not, because a is an object that acts like a dictionary
❷ Treating the as a dictionary, you can get a list of the names of the attributes of this element by using This element has only one attrib
❸ Attribute names, like all other text in an XML document, are stored in unicode.
❹ Again treating the as a dictionary, you can get a list of the values of the attributes by using The values are themselves objects, of typ
❺ Still treating the as a dictionary, you can access an individual attribute by name, using normal dictionary syntax. (Readers who have b

Example 9.25. Accessing individual attributes

❶ The `Attr` object completely represents a single XML attribute of a single XML element. The name of the attribute (the same name as you u

❷ The actual text value of this XML attribute is stored in `value`.

Like a dictionary, attributes of an XML element have no ordering. Attributes may *happen to be* listed in a certain order in the original XML d

9.7. Segue

OK, that's it for the hard-core XML stuff. The next chapter will continue to use these same example programs, but focus on other aspects tha

Before moving on to the next chapter, you should be comfortable doing all of these things:

- Parsing XML documents using `minidom` and searching through the parsed document, and accessing arbitrary element attributes and element chil
- Organizing complex libraries into packages
- Converting unicode strings to different character encodings

---

[1] This, sadly, is *still* an oversimplification. Unicode now has been extended to handle ancient Chinese, Korean, and Japanese texts, which had

Chapter 10. Scripts and Streams

10.1. Abstracting input sources

One of Python's greatest strengths is its dynamic binding, and one powerful use of dynamic binding is the *file-like object*.

Many functions which require an input source could simply take a filename, go open the file for reading, read it, and close it when they're do

In the simplest case, a *file-like object* is any object with a method with an optional parameter, which returns a string. When called with no

This is how reading from real files works; the difference is that you're not limiting yourself to real files. The input source could be anything:

In case you were wondering how this relates to XML processing, is one such function which can take a file-like object.

Example 10.1. Parsing XML from a file

```
```

1. First, you open the file on disk. This gives you a file object.
2. You pass the file object to which calls the method of and reads the XML document from the file on disk.
3. Be sure to call the method of the file object after you're done with it. will not do this for you.
4. Calling the method on the returned XML document prints out the entire thing.

Well, that all seems like a colossal waste of time. After all, you've already seen that can simply take the filename and do all the opening and

Example 10.2. Parsing XML from a URL

```
```

```
```

1. As you saw in a previous chapter, takes a web page URL and returns a file-like object. Most importantly, this object has a method wh
2. Now you pass the file-like object to which obediently calls the method of the object and parses the XML data that the method return
3. As soon as you're done with it, be sure to close the file-like object that gives you.
4. By the way, this URL is real, and it really is XML. It's an XML representation of the current headlines on Slashdot (http://slashdot.org

Example 10.3. Parsing XML from a string (the easy but inflexible way)

```
```

1. has a method, which takes an entire XML document as a string and parses it. You can use this instead of if you know you already ha

OK, so you can use the function for parsing both local files and remote URLs, but for parsing strings, you use... a different function. That me

If there were a way to turn a string into a file-like object, then you could simply pass this object to And in fact, there is a module specifically

Example 10.4. Introducing

```
```

1. The module contains a single class, also called which allows you to turn a string into a file-like object. The class takes the string as
2. Now you have a file-like object, and you can do all sorts of file-like things with it. Like which returns the original string.
3. Calling again returns an empty string. This is how real file objects work too; once you read the entire file, you can't read any more wi
4. You can explicitly seek to the beginning of the string, just like seeking through a file, by using the method of the object.
5. You can also read the string in chunks, by passing a parameter to the method.
6. At any time, will return the rest of the string that you haven't read yet. All of this is exactly how file objects work; hence the term *file-*

Example 10.5. Parsing XML from a string (the file-like object way)

1. Now you can pass the file-like object (really a to which will call the object's method and happily parse away, never knowing that it

So now you know how to use a single function, to parse an XML document stored on a web page, in a local file, or in a hard-coded string. F

Example 10.6.

1. The function takes a single parameter, and returns a file-like object. is a string of some sort; it can either be a URL (like a full or p
2. First, you see if is a URL. You do this through brute force: you try to open it as a URL and silently ignore errors caused by trying to o
3. On the other hand, if yelled at you and told you that wasn't a valid URL, you assume it's a path to a file on disk and try to open it. Ag

4 By this point, you need to assume that s a string that has hard-coded data in it (since nothing else worked), so you use o create a file

Now you can use this unction in conjunction with o make a function that takes a hat refers to an XML document somehow (either as a UR

Example 10.7. Using in

10.2. Standard input, output, and error

UNIX users are already familiar with the concept of standard input, standard output, and standard error. This section is for the rest of you.

Standard output and standard error (commonly abbreviated and) are pipes that are built into every UNIX system. When you omething, it g

Example 10.8. Introducing and

1 As you saw in Example 6.9, “Simple Counters”, you can use Python’s built-in unction to build simple counter loops that repeat some

2 s a file-like object; calling its unction will print out whatever string you give it. In fact, this is what the unction really does; it adds

3 In the simplest case, and end their output to the same place: the Python IDE (if you’re in one), or the terminal (if you’re running Pyt
and are both file-like objects, like the ones you discussed in Section 10.1, “Abstracting input sources”, but they are both write-only. They ha

Example 10.9. Redirecting output

(On Windows, you can use nstead of o display the contents of a file.)

If you have not already done so, you can download this and other examples (http://diveintopython.org/download/diveintopython-examples-5.

```
```

1. This will print to the IDE "Interactive Window" (or the terminal, if running the script from the command line).
2. Always save before redirecting it, so you can set it back to normal later.
3. Open a file for writing. If the file doesn't exist, it will be created. If the file does exist, it will be overwritten.
4. Redirect all further output to the new file you just opened.
5. This will be "printed" to the log file only; it will not be visible in the IDE window or on the screen.
6. Set back to the way it was before you mucked with it.
7. Close the log file.

Redirecting works exactly the same way, using instead of

Example 10.10. Redirecting error information

```
```

If you have not already done so, you can download this and other examples (http://diveintopython.org/download/diveintopython-examples-5.

```
```

1. Open the log file where you want to store debugging information.
2. Redirect standard error by assigning the file object of the newly-opened log file to
3. Raise an exception. Note from the screen output that this does *not* print anything on screen. All the normal traceback information has b
4. Also note that you're not explicitly closing your log file, nor are you setting back to its original value. This is fine, since once the prog

Since it is so common to write error messages to standard error, there is a shorthand syntax that can be used instead of going through the hass

Example 10.11. Printing to

```
```

❶ This shorthand syntax of the print statement can be used to write to any open file, or file-like object. In this case, you can redirect a single p

Standard input, on the other hand, is a read-only file object, and it represents the data flowing into the program from some previous program.

Example 10.12. Chaining commands

❶ As you saw in Section 9.1, “Diving in”, this will print a string of eight random bits, 0 or 1.
❷ This simply prints out the entire contents of p (Windows users should use p instead of p).
❸ This prints the contents of p, but the "|" character, called the "pipe" character, means that the contents will not be printed to the screen. I
❹ Instead of specifying a module (like p), you specify "-", which causes your script to load the grammar from standard input instead of fro

So how does the script "know" to read from standard input when the grammar file is "-"? It’s not magic; it’s just code.

Example 10.13. Reading from standard input in p

❶ This is the function from which you previously examined in Section 10.1, “Abstracting input sources”. All you’ve done is add three

10.3. Caching node lookups

employs several tricks which may or may not be useful to you in your XML processing. The first one takes advantage of the consistent struc

A grammar file defines a series of elements. Each contains one or more elements, which can contain a lot of different things, including W

This is how you build up the grammar: define elements for the smallest pieces, then define elements which "include" the first elements by u

This is all very flexible, but there is one downside: performance. When you find an and need to find the corresponding element, you have a

Example 10.14.

❶
❷
❸
❹

❶ Start by creating an empty dictionary,
❷ As you saw in Section 9.5, “Searching for elements”, returns a list of all the elements of a particular name. You easily can get a list of
❸ As you saw in Section 9.6, “Accessing element attributes”, you can access individual attributes of an element by name, using standard
❹ The values of the dictionary will be the elements themselves. As you saw in Section 9.3, “Parsing XML”, each element, each node, e

Once you build this cache, whenever you come across an and need to find the element with the same attribute, you can simply look it up in

Example 10.15. Using the element cache

❶
❷
❸

You’ll explore the function in the next section.

10.4. Finding direct children of a node

Another useful techique when parsing XML documents is finding all the direct child elements of a particular element. For instance, in the gra

You might think you could simply use for this, but you can’t. searches recursively and returns a single list for all the elements it finds. Since

Example 10.16. Finding direct child elements

❶
❷
❸
❹
❺

❶ As you saw in Example 9.9, “Getting child nodes”, the attribute returns a list of all the child nodes of an element.
❷ However, as you saw in Example 9.11, “Child nodes can be text”, the list returned by contains all different types of nodes, including t
❸ Each node has a attribute, which can be or any number of other values. The complete list of possible values is in the file in the pa

❹ Once you have a list of actual elements, choosing a random one is easy. Python comes with a module called which includes several us

10.5. Creating separate handlers by node type

The third useful XML processing tip involves separating your code into logical functions, based on node types and element names. Parsed XI

Example 10.17. Class names of parsed XML objects



❶ Assume for a moment that is in the current directory.

❷ As you saw in Section 9.2, “Packages”, the object returned by parsing an XML document is a object, as defined in the in the package

❸ Furthermore, is a built-in attribute of every Python class, and it is a string. This string is not mysterious; it’s the same as the class nam

Fine, so now you can get the class name of any particular XML node (since each XML node is represented as a Python object). How can you

Example 10.18. a generic XML node dispatcher



❶ First off, notice that you’re constructing a larger string based on the class name of the node you were passed (in the argument). So if y

❷ Now you can treat that string as a function name, and get a reference to the function itself using

❸ Finally, you can call that function and pass the node itself as an argument. The next example shows the definitions of each of these fun

Example 10.19. Functions called by the dispatcher



```
```

1. is only ever called once, since there is only one node in an XML document, and only one object in the parsed XML representation. I
2. is called on nodes that represent bits of text. The function itself does some special processing to handle automatic capitalization of the
3. is just a since you don't care about embedded comments in the grammar files. Note, however, that you still need to define the functio
4. The method is actually itself a dispatcher, based on the name of the element's tag. The basic idea is the same: take what distinguishes

In this example, the dispatch functions and simply find other methods in the same class. If your processing is very complex (or you have ma

10.6. Handling command-line arguments

Python fully supports creating programs that can be run on the command line, complete with command-line arguments and either short- or lo

It's difficult to talk about command-line processing without understanding how command-line arguments are exposed to your Python progra

Example 10.20. Introducing

If you have not already done so, you can download this and other examples (http://diveintopython.org/download/diveintopython-examples-5.

```
```

1. Each command-line argument passed to the program will be in which is just a list. Here you are printing each argument on a separate

Example 10.21. The contents of

```
```

1. The first thing to know about is that it contains the name of the script you're calling. You will actually use this knowledge to your adv
2. Command-line arguments are separated by spaces, and each shows up as a separate element in the list.
3. Command-line flags, like also show up as their own element in the list.
4. To make things even more interesting, some command-line flags themselves take arguments. For instance, here you have a flag (whi

So as you can see, you certainly have all the information passed on the command line, but then again, it doesn't look like it's going to be all t

Example 10.22. Introducing

1. First off, look at the bottom of the example and notice that you're calling the function with Remember, is the name of the script that
2. This is where all the interesting processing happens. The function of the module takes three parameters: the argument list (which you
3. If anything goes wrong trying to parse these command-line flags, will raise an exception, which you catch. You told all the flags you
4. As is standard practice in the UNIX world, when the script is passed flags it doesn't understand, you print out a summary of proper us

So what are all those parameters you pass to the function? Well, the first one is simply the raw list of command-line flags and arguments (no

h
: print usage summary

g
: use specified grammar file or URL

d
: show debugging information while parsing

The first and third flags are simply standalone flags; you specify them or you don't, and they do things (print help) or change state (turn on de

To further complicate things, the script accepts either short flags (like or long flags (like and you want them to do the same thing. This is

help
: print usage summary

g
: use specified grammar file or URL

Three things of note here:

- All long flags are preceded by two dashes on the command line, but you don't include those dashes when calling They are understo
- The flag must always be followed by an additional argument, just like the flag. This is notated by an equals sign,
- The list of long flags is shorter than the list of short flags, because the flag does not have a corresponding long version. This is fine;

Confused yet? Let's look at the actual code and see if it makes sense in context.

Example 10.23. Handling command-line arguments in

```
y
                                                
```

Oh, and along the way, you take advantage of your knowledge of the structure of the XML document to set up a little cache of references, wh

If you specified some source material on the command line, you use that; otherwise you rip through the grammar looking for the "top-level" r

Now you rip through the source material. The source material is also XML, and you parse it one node at a time. To keep the code separated a

You bounce through the grammar, parsing all the children of each p element,

replacing e elements with a random child,

and replacing f elements with a random child of the corresponding e element, which you previously cached.

Eventually, you parse your way down to plain text,

which you print out.

10.8. Summary

Python comes with powerful libraries for parsing and manipulating XML documents. The takes an XML file and parses it into Python object

Before moving on to the next chapter, you should be comfortable doing all of these things:

- Chaining programs with standard input and output·
- Defining dynamic dispatchers with
- Using command-line flags and validating them with

Chapter 11. HTTP Web Services

11.1. Diving in

You've learned about HTML processing and XML processing, and along the way you saw how to download a web page and how to parse XI

Simply stated, HTTP web services are programmatic ways of sending and receiving data from remote servers using the operations of HTTP d

The main advantage of this approach is simplicity, and its simplicity has proven popular with a lot of different sites. Data -- usually XML dat

Examples of pure XML-over-HTTP web services:

- Amazon API (http://www.amazon.com/webservices) allows you to retrieve product information from the Amazon.com online store.
- National Weather Service (http://www.nws.noaa.gov/alerts/) (United States) and Hong Kong Observatory (http://demo.xml.weather
- Atom API (http://atomenabled.org/) for managing web-based content.·
- Syndicated feeds (http://syndic8.com/) from weblogs and news sites bring you up-to-the-minute news from a variety of sites.·

In later chapters, you'll explore APIs which use HTTP as a transport for sending and receiving data, but don't map application semantics to t

Here is a more advanced version of the [illegible] module that you saw in the previous chapter:

Example 11.1. [illegible]

If you have not already done so, you can download this and other examples (http://diveintopython.org/download/diveintopython-examples-5.

[illegible]

Further reading

Paul Prescod believes that pure HTTP web services are the future of the Internet (http://webservices.xml.com/pub/a/ws/2002/02/06/

11.2. How not to fetch data over HTTP

Let’s say you want to download a resource over HTTP, such as a syndicated Atom feed. But you don’t just want to download it once; you wa

Example 11.2. Downloading a feed the quick-and-dirty way

❶ Downloading anything over HTTP is incredibly easy in Python; in fact, it’s a one-liner. The module has a handy function that takes t

So what’s wrong with this? Well, for a quick one-off during testing or development, there’s nothing wrong with it. I do it all the time. I wante

Let’s talk about some of the basic features of HTTP.

11.3. Features of HTTP

There are five important features of HTTP which you should support.

11.3.1.

The is simply a way for a client to tell a server who it is when it requests a web page, a syndicated feed, or any sort of web service over HTT

By default, Python sends a generic In the next section, you’ll see how to change this to something more specific.

11.3.2. Redirects

Sometimes resources move around. Web sites get reorganized, pages move to new addresses. Even web services can reorganize. A syndicate

Every time you request any kind of resource from an HTTP server, the server includes a status code in its response. Status code 0 means "ever

HTTP has two different ways of signifying that a resource has moved. Status code 0 is a *temporary redirect*; it means "oops, that got moved o

will automatically "follow" redirects when it receives the appropriate status code from the HTTP server, but unfortunately, it doesn't tell you

11.3.3.

Some data changes all the time. The home page of CNN.com is constantly updating every few minutes. On the other hand, the home page of

If you ask for the same data a second time (or third, or fourth), you can tell the server the last-modified date that you got last time: you send a

All modern web browsers support last-modified date checking. If you've ever visited a page, re-visited the same page a day later and found th

Python's URL library has no built-in support for last-modified date checking, but since you can add arbitrary headers to each request and rea

11.3.4.

ETags are an alternate way to accomplish the same thing as the last-modified date checking: don't re-download data that hasn't changed. The

Python's URL library has no built-in support for ETags, but you'll see how to add it later in this chapter.

11.3.5. Compression

The last important HTTP feature is gzip compression. When you talk about HTTP web services, you're almost always talking about moving

Python's URL library has no built-in support for gzip compression per se, but you can add arbitrary headers to the request. And Python come

Note that our little one-line script to download a syndicated feed did not support any of these HTTP features. Let's see how you can improve

11.4. Debugging HTTP web services

First, let's turn on the debugging features of Python's HTTP library and see what's being sent over the wire. This will be useful throughout th

Example 11.3. Debugging HTTP



```
```

1. relies on another standard Python library, Normally you don't need to directly (does that automatically), but you will here so you ca
2. Now that the debugging flag is set, information on the the HTTP request and response is printed out in real time. The first thing it tells
3. When you request the Atom feed, sends three lines to the server. The first line specifies the HTTP verb you're using, and the path of t
4. The second line is the header, which specifies the domain name of the service you're accessing. This is important, because a single H
5. The third line is the header. What you see here is the generic that the library adds by default. In the next section, you'll see how to cu
6. The server replies with a status code and a bunch of headers (and possibly some data, which got stored in the variable). The status coc
7. The server tells you when this Atom feed was last modified (in this case, about 13 minutes ago). You can send this date back to the ser
8. The server also tells you that this Atom feed has an ETag hash of The hash doesn't mean anything by itself; there's nothing you can d

11.5. Setting the

The first step to improving your HTTP web services client is to identify yourself properly with a To do that, you need to move beyond the b

Example 11.4. Introducing

```
```

1. If you still have your Python IDE open from the previous section's example, you can skip this, but this turns on HTTP debugging so y
2. Fetching an HTTP resource with s a three-step process, for good reasons that will become clear shortly. The first step is to create a
3. The second step is to build a URL opener. This can take any number of handlers, which control how responses are handled. But you ca

❹ The final step is to tell the opener to open the URL, using the object you created. As you can see from all the debugging information t

Example 11.5. Adding headers with the

❶
❷
❸
❹

❶ You're continuing from the previous example; you've already created a object with the URL you want to access.
❷ Using the method on the object, you can add arbitrary HTTP headers to the request. The first argument is the header, the second is th
❸ The object you created before can be reused too, and it will retrieve the same feed again, but with your custom header.
❹ And here's you sending your custom in place of the generic one that Python sends by default. If you look closely, you'll notice that y

11.6. Handling and

Now that you know how to add custom HTTP headers to your web service requests, let's look at adding support for and headers.

These examples show the output with debugging turned off. If you still have it turned on from the previous section, you can turn it off by sett

Example 11.6. Testing

❶

```
```

1. Remember all those HTTP headers you saw printed out when you turned on debugging? This is how you can get access to them progr
2. On the second request, you add the header with the last-modified date from the first request. If the data hasn't changed, the server sho
3. Sure enough, the data hasn't changed. You can see from the traceback that throws a special exception, in response to the status code

also raises an exception for conditions that you would think of as errors, such as (page not found). In fact, it will raise for *any* status code o

Example 11.7. Defining URL handlers

This custom URL handler is part of

```
```

1. is designed around URL handlers. Each handler is just a class that can define any number of methods. When something happens -- lik
2. searches through the defined handlers and calls the method when it encounters a status code from the server. By defining a custom e
3. This is the key part: before returning, you save the status code returned by the HTTP server. This will allow you easy access to it from

Example 11.8. Using custom URL handlers

```
```

```
```

1. You’re continuing the previous example, so the object is already set up, and you’ve already added the header.
2. This is the key: now that you’ve defined your custom URL handler, you need to tell to use it. Remember how I said that broke up the
3. Now you can quietly open the resource, and what you get back is an object that, along with the usual headers (use to acess them), also
4. Note that when the server sends back a status code, it doesn’t re-send the data. That’s the whole point: to save bandwidth by not re-do

Handling works much the same way, but instead of checking for and sending you check for and send Let’s start with a fresh IDE session

Example 11.9. Supporting

```
```

1. Using the pseudo-dictionary, you can get the returned from the server. (What happens if the server didn’t send back an Then this lin
2. OK, you got the data.
3. Now set up the second call by setting the header to the you got from the first call.
4. The second call succeeds quietly (without throwing an exception), and once again you see that the server has sent back a status code.
5. Regardless of whether the is triggered by date checking or hash matching, you’ll never get the data along with the That’s the whole

In these examples, the HTTP server has supported both [illegible] and [illegible] headers, but not all servers do. As a web services client, you should be prepared

11.7. Handling redirects

You can support permanent and temporary redirects using a different kind of custom URL handler.

First, let’s see why a redirect handler is necessary in the first place.

Example 11.10. Accessing web services without a redirect handler



```
```

1. You'll be better able to see what's happening if you turn on debugging.
2. This is a URL which I have set up to permanently redirect to my Atom feed at
3. Sure enough, when you try to download the data at that address, the server sends back a status code, telling you that the resource has
4. The server also sends back a header that gives the new address of this data.
5. notices the redirect status code and automatically tries to retrieve the data at the new location specified in the header.
6. The object you get back from the contains the new permanent address and all the headers returned from the second request (retrieved

This is suboptimal, but easy to fix. doesn't behave exactly as you want it to when it encounters a or so let's override its behavior. How? W

Example 11.11. Defining the redirect handler

This class is defined in

```
```

```
```

1. Redirect behavior is defined in in a class called You don't want to completely override the behavior, you just want to extend it a littl
2. When it encounters a status code from the server, will search through its handlers and call the method. The first thing ours does is ju
3. Here's the key: before you return, you store the status code (so that the calling program can access it later.
4. Temporary redirects (status code work the same way: override the method, call the ancestor, and save the status code before returnin

So what has this bought us? You can now build a URL opener with the custom redirect handler, and it will still automatically follow redirects

Example 11.12. Using the redirect handler to detect permanent redirects

```
```

```
[illegible]
[illegible]
[illegible]
```

❶ This is a sample URL I've set up that is configured to tell clients to *temporarily* redirect to [redacted]

❷ The server sends back a [redacted]tatus code, indicating a temporary redirect. The temporary new location of the data is given in the [redacted]eader.

❸ [redacted]alls your [redacted]method, which calls the ancestor method of the same name in [redacted]which follows the redirect to the new location. Then your [redacted]

❹ And here you are, having successfully followed the redirect to [redacted]ells you that this was a temporary redirect, which means that you sho

11.8. Handling compressed data

The last important HTTP feature you want to support is compression. Many web services have the ability to send data compressed, which ca

Servers won't give you compressed data unless you tell them you can handle it.

Example 11.14. Telling the server you would like compressed data

```
[illegible]
```

1. This is the key: once you've created your [illegible]bject, add an [illegible]eader to tell the server you can accept gzip-encoded data. [illegible]s the name of th
2. There's your header going across the wire.
3. And here's what the server sends back: the [illegible]eader means that the data you're about to receive has been gzip-compressed.
4. The [illegible]eader is the length of the compressed data, not the uncompressed data. As you'll see in a minute, the actual length of the uncom

Example 11.15. Decompressing the data

```
[illegible]
```

1. Continuing from the previous example, [illegible]s the file-like object returned from the URL opener. Using its [illegible]method would ordinarily get y
2. OK, this step is a little bit of messy workaround. Python has a [illegible]module, which reads (and actually writes) gzip-compressed files on dis
3. Now you can create an instance of [illegible]and tell it that its "file" is the file-like object [illegible]
4. This is the line that does all the actual work: "reading" from [illegible]will decompress the data. Strange? Yes, but it makes sense in a twisted k
5. Look ma, real data. (15955 bytes of it, in fact.)

"But wait!" I hear you cry. "This could be even easier!" I know what you're thinking. You're thinking that returns a file-like object, so why

Example 11.16. Decompressing the data directly from the server



1. Continuing from the previous example, you already have a object set up with an header.
2. Simply opening the request will get you the headers (though not download any data yet). As you can see from the returned header, thi
3. Since returns a file-like object, and you know from the headers that when you read it, you're going to get gzip-compressed data, why

11.9. Putting it all together

You've seen all the pieces for building an intelligent HTTP web services client. Now let's see how they all fit together.

Example 11.17. The function

This function is defined in



1. is a handy utility module for, you guessed it, parsing URLs. It's primary function, also called takes a URL and splits it into a tuple o
2. You identify yourself to the HTTP server with the passed in by the calling function. If no was specified, you use a default one define
3. If an hash was given, send it in the header.
4. If a last-modified date was given, send it in the header.
5. Tell the server you would like compressed data if possible.
6. Build a URL opener that uses *both* of the custom URL handlers: for handling and redirects, and for handling and other error cond

❼ That’s it! Open the URL and return a file-like object to the caller.

Example 11.18. The function

This function is defined in

```
```

❶ First, you call the function with a URL, hash, date, and
❷ Read the actual data returned from the server. This may be compressed; if so, you’ll decompress it later.
❸ Save the hash returned from the server, so the calling application can pass it back to you next time, and you can pass it on to which c
❹ Save the date too.
❺ If the server says that it sent compressed data, decompress it.
❻ If you got a URL back from the server, save it, and assume that the status code is until you find out otherwise.
❼ If one of the custom URL handlers captured a status code, then save that too.

Example 11.19. Using

```
```

```
[
  {
    'status': 200,
    'data': '<feed xmlns="http://www.w3.org/2005/Atom" xml:lang="en">\n<title>dive into mark</title>\n<subtitle type="html">A lot of these entries have been unwritten for years, sometimes decades, waiting for just the right moment when I found the right thing to say. Now seems as good a time as any.</subtitle>\n<updated>2009-03-27T21:56:07Z</updated>\n<id>tag:diveintomark.org,2001-07-29:/</id>\n<generator uri="http://www.sixapart.com/movabletype/" version="4.01-en">Movable Type</generator>\n<link rel="alternate" type="text/html" hreflang="en" href="http://diveintomark.org/" />\n<link rel="self" type="application/atom+xml" href="http://diveintomark.org/feed/" />\n<entry>\n<author>\n<name>Mark</name>\n<uri>http://diveintomark.org/</uri>\n</author>\n<title type="html">Dive into history, 2009 edition</title>\n<link rel="alternate" type="text/html" href="http://diveintomark.org/archives/2009/03/27/dive-into-history-2009-edition" />\n<id>tag:diveintomark.org,2009-03-27:/archives/20090327172042</id>\n<updated>2009-03-27T21:56:07Z</updated>\n<published>2009-03-27T17:20:42Z</published>\n<category scheme="http://diveintomark.org" term="diveintopython" />\n<category scheme="http://diveintomark.org" term="docbook" />\n<category scheme="http://diveintomark.org" term="html" />\n<summary type="html">Putting the finishing touches on the 10th edition of my favorite Unix book.</summary>\n</entry>\n<entry>\n<author>\n<name>Mark</name>\n<uri>http://diveintomark.org/</uri>\n</author>\n<title type="html">Accessibility is a harsh mistress</title>\n<link rel="alternate" type="text/html" href="http://diveintomark.org/archives/2009/03/21/accessibility-is-a-harsh-mistress" />\n<id>tag:diveintomark.org,2009-03-21:/archives/20090321184019</id>\n<updated>2009-03-22T01:05:09Z</updated>\n<published>2009-03-21T18:40:19Z</published>\n<category scheme="http://diveintomark.org" term="accessibility" />\n<summary type="html">Thanks to everyone who has given me feedback about the site redesign. Please keep it cards and letters, I can take it.</summary>\n</entry>\n<entry>\n<author>\n<name>Mark</name>\n<uri>http://diveintomark.org/</uri>\n</author>\n<title type="html">Tech support cat</title>\n<link rel="alternate" type="text/html" href="http://diveintomark.org/archives/2009/03/21/tech-support-cat" />\n<id>tag:diveintomark.org,2009-03-21:/archives/20090321172431</id>\n<updated>2009-03-21T22:45:31Z</updated>\n<published>2009-03-21T17:24:31Z</published>\n<category scheme="http://diveintomark.org" term="cats" />\n<category scheme="http://diveintomark.org" term="meta" />\n<summary type="html">Sent in by readers around the world.</summary>\n</entry>\n<entry>\n<author>\n<name>Mark</name>\n<uri>http://diveintomark.org/</uri>\n</author>\n<title type="html">Tools of the trade</title>\n<link rel="alternate" type="text/html" href="http://diveintomark.org/archives/2009/03/21/tools-of-the-trade" />\n<id>tag:diveintomark.org,2009-03-21:/archives/20090321170915</id>\n<updated>2009-03-21T22:45:31Z</updated>\n<published>2009-03-21T17:09:15Z</published>\n<category scheme="http://diveintomark.org" term="diveintopython" />\n<category scheme="http://diveintomark.org" term="docbook" />\n<category scheme="http://diveintomark.org" term="meta" />\n<summary type="html">Hello, fellow Dive Into Python reader. Are you using Windows? Are you using Python? Are you using XML? Are you using DocBook XML? Are you using Windows Python XML DocBook? Then I have the book for you.</summary>\n</entry>\n<entry>\n<author>\n<name>Mark</name>\n<uri>http://diveintomark.org/</uri>\n</author>\n<title type="html">Thank you for not reading my book</title>\n<link rel="alternate" type="text/html" href="http://diveintomark.org/archives/2009/03/21/thank-you-for-not-reading-my-book" />\n<id>tag:diveintomark.org,2009-03-21:/archives/20090321165054</id>\n<updated>2009-03-21T22:45:31Z</updated>\n<published>2009-03-21T16:50:54Z</published>\n<category scheme="http://diveintomark.org" term="diveintopython" />\n<category scheme="http://diveintomark.org" term="meta" />\n<summary type="html">Dear Publishing Industry: Please stop emailing me to tell me that you have published my book. Sincerely, Mark</summary>\n</entry>\n</feed>\n',
    'hash': '"e7a6bb1ab26efdd8c5fd0dfcf20ba957"',
    'modified': 'Fri, 27 Mar 2009 21:56:07 GMT',
    'status': 200
  }
]
```

1. The very first time you fetch a resource, you don't have an `etag` hash or `last-modified` date, so you'll leave those out. (They're optional parameters.)
2. What you get back is a dictionary of several useful headers, the HTTP status code, and the actual data returned from the server. `feedparser` handl
3. If you ever get a `301` status code, that's a permanent redirect, and you need to update your URL to the new address.
4. The second time you fetch the same resource, you have all sorts of information to pass back: a (possibly updated) URL, the `etag` from the l
5. What you get back is again a dictionary, but the data hasn't changed, so all you got was a `304` status code and no data.

11.10. Summary

The `feedparser` and its functions should now make perfect sense.

There are 5 important features of HTTP web services that every client should support:

- Identifying your application by setting a proper `User-Agent`
- Handling permanent redirects properly.
- Supporting `Last-Modified` date checking to avoid re-downloading data that hasn't changed.
- Supporting `ETag` hashes to avoid re-downloading data that hasn't changed.
- Supporting gzip compression to reduce bandwidth even when data *has* changed.

Chapter 12. SOAP Web Services

Chapter 11 focused on document-oriented web services over HTTP. The "input parameter" was the URL, and the "return value" was an actua

This chapter will focus on SOAP web services, which take a more structured approach. Rather than dealing with HTTP requests and XML do

SOAP is a complex specification, and it is somewhat misleading to say that SOAP is all about calling remote functions. Some people would p

12.1. Diving In

You use Google, right? It’s a popular search engine. Have you ever wished you could programmatically access Google search results? Now y

Example 12.1.

```
```

You can import this as a module and use it from a larger program, or you can run the script from the command line. On the command line, yo

Here is the sample output for a search for the word "python".

Example 12.2. Sample Usage of

```
```

Further Reading on SOAP

http://www.xmethods.net/ is a repository of public access SOAP web services.·
The SOAP specification (http://www.w3.org/TR/soap/) is surprisingly readable, if you like that sort of thing.·

12.2. Installing the SOAP Libraries

Unlike the other code in this book, this chapter relies on libraries that do not come pre-installed with Python.

Before you can dive into SOAP web services, you'll need to install three libraries: PyXML, fpconst, and SOAPpy.

12.2.1. Installing PyXML

The first library you need is PyXML, an advanced set of XML libraries that provide more functionality than the built-in XML libraries we stu

Procedure 12.1.

Here is the procedure for installing PyXML:

1.
Go to http://pyxml.sourceforge.net/, click Downloads, and download the latest version for your operating system.

2.
If you are using Windows, there are several choices. Make sure to download the version of PyXML that matches the version of Pyth

3.
Double-click the installer. If you download PyXML 0.8.3 for Windows and Python 2.3, the installer program will be

4.
Step through the installer program.

5.
After the installation is complete, close the installer. There will not be any visible indication of success (no programs installed on the

To verify that you installed PyXML correctly, run your Python IDE and check the version of the XML libraries you have installed, as shown

Example 12.3. Verifying PyXML Installation

```
[illegible]
[illegible]
0
```

This version number should match the version number of the PyXML installer program you downloaded and ran.

12.2.2. Installing fpconst

The second library you need is fpconst, a set of constants and functions for working with IEEE754 double-precision special values. This prov

Procedure 12.2.

Here is the procedure for installing fpconst:

1. Download the latest version of fpconst from http://www.analytics.washington.edu/statcomp/projects/rzope/fpconst/.
2. There are two downloads available, one in [illegible] format, the other in [illegible] format. If you are using Windows, download the [illegible] file; otherwise, do
3. Decompress the downloaded file. On Windows XP, you can right-click on the file and choose Extract All; on earlier versions of Wi
4. Open a command prompt and navigate to the directory where you decompressed the fpconst files.
5. Type [illegible] to run the installation program.

To verify that you installed fpconst correctly, run your Python IDE and check the version number.

Example 12.4. Verifying fpconst Installation

```
[illegible]
[illegible]
0
```

This version number should match the version number of the fpconst archive you downloaded and installed.

12.2.3. Installing SOAPpy

The third and final requirement is the SOAP library itself: SOAPpy.

Procedure 12.3.

Here is the procedure for installing SOAPpy:

Go to http://pywebsvcs.sourceforge.net/ and select Latest Official Release under the SOAPpy section.1.

2.
There are two downloads available. If you are using Windows, download the file; otherwise, download the file.

3.
Decompress the downloaded file, just as you did with fpconst.

4.
Open a command prompt and navigate to the directory where you decompressed the SOAPpy files.

5.
Type to run the installation program.

To verify that you installed SOAPpy correctly, run your Python IDE and check the version number.

Example 12.5. Verifying SOAPpy Installation

This version number should match the version number of the SOAPpy archive you downloaded and installed.

12.3. First Steps with SOAP

The heart of SOAP is the ability to call remote functions. There are a number of public access SOAP servers that provide simple functions fo

The most popular public access SOAP server is http://www.xmethods.net/. This example uses a demonstration function that takes a United S

Example 12.6. Getting the Current Temperature

❶ You access the remote SOAP server through a proxy class, The proxy handles all the internals of SOAP for you, including creating t

❷ Every SOAP service has a URL which handles all the requests. The same URL is used for all function calls. This particular service on

❸ You're creating the with the service URL and the service namespace. This doesn't make any connection to the SOAP server; it simpl

❹ Now with everything configured properly, you can actually call remote SOAP methods as if they were local functions. You pass argu

Let's peek under those covers.

12.4. Debugging SOAP Web Services

The SOAP libraries provide an easy way to see what's going on behind the scenes.

Turning on debugging is a simple matter of setting two flags in the configuration.

Example 12.7. Debugging SOAP Web Services

```
                                                
```

Most of the XML request document that gets sent to the server is just boilerplate. Ignore all the namespace declarations; they're going to be t



1. The element name is the function name, uses as a dispatcher. Instead of calling separate local methods based on the method name, i
2. The function's XML element is contained in a specific namespace, which is the namespace you specified when you created the object
3. The arguments of the function also got translated into XML. introspects each argument to determine its datatype (in this case it's a str

The XML return document is equally easy to understand, once you know what to ignore. Focus on this fragment within the



1. The server wraps the function return value within a element. By convention, this wrapper element is the name of the function, plus
2. The server returns the response in the same namespace we used in the request, the same namespace we specified when we first create
3. The return value is specified, along with its datatype (it's a float). uses this explicit datatype to create a Python object of the correct na

12.5. Introducing WSDL

The class proxies local method calls and transparently turns then into invocations of remote SOAP methods. As you've seen, this is a lot of w

Consider this: the previous two sections showed an example of calling a simple remote SOAP method with one argument and one return valu

That shouldn't come as a big surprise. If I wanted to call a local function, I would need to know what package or module it was in (the equiva

The big difference is introspection. As you saw in Chapter 4, Python excels at letting you discover things about modules and functions at run

WSDL lets you do that with SOAP web services. WSDL stands for "Web Services Description Language". Although designed to be flexible

A WSDL file is just that: a file. More specifically, it's an XML file. It usually lives on the same server you use to access the SOAP web servi

A WSDL file contains a description of everything involved in calling a SOAP web service:

- The service URL and namespace
- The type of web service (probably function calls using SOAP, although as I mentioned, WSDL is flexible enough to describe a wide
- The list of available functions
- The arguments for each function
- The datatype of each argument
- The return values of each function, and the datatype of each return value

In other words, a WSDL file tells you everything you need to know to be able to call a SOAP web service.

12.6. Introspecting SOAP Web Services with WSDL

Like many things in the web services arena, WSDL has a long and checkered history, full of political strife and intrigue. I will skip over this h

The most fundamental thing that WSDL allows you to do is discover the available methods offered by a SOAP server.

Example 12.8. Discovering The Available Methods



1. SOAPpy includes a WSDL parser. At the time of this writing, it was labeled as being in the early stages of development, but I had no p
2. To use a WSDL file, you again use a proxy class, which takes a single argument: the WSDL file. Note that in this case you are passin
3. The WSDL proxy class exposes the available functions as a Python dictionary, So getting the list of available methods is as simple as

Okay, so you know that this SOAP server offers a single method: But how do you call it? The WSDL proxy object can tell you that too.

Example 12.9. Discovering A Method's Arguments



1. The dictionary is filled with a SOAPpy-specific structure called A object contains information about one specific function, includin
2. The function arguments are stored in which is a Python list of objects that hold information about each parameter.
3. Each object contains a attribute, which is the argument name. You are not required to know the argument name to call the function th
4. Each parameter is also explicitly typed, using datatypes defined in XML Schema. You saw this in the wire trace in the previous sectio

WSDL also lets you introspect into a function's return values.

Example 12.10. Discovering A Method's Return Values



❶ The adjunct to for function arguments is for return value. It is also a list, because functions called through SOAP can return multiple

❷ Each object contains and This function returns a single value, named which is a float.

Let’s put it all together, and call a SOAP web service through a WSDL proxy.

Example 12.11. Calling A Web Service Through A WSDL Proxy

```
[illegible]
```

❶ The configuration is simpler than calling the SOAP service directly, since the WSDL file contains the both service URL and namespac

❷ Once the object is created, you can call a function as easily as you did with the object. This is not surprising; the is just a wrapper arc

❸ You can access the s with This is useful to turning on debugging, so that when you can call functions through the WSDL proxy, its

12.7. Searching Google

Let’s finally turn to the sample code that you saw that the beginning of this chapter, which does something more useful and exciting than get

Google provides a SOAP API for programmatically accessing Google search results. To use it, you will need to sign up for Google Web Serv

Procedure 12.4. Signing Up for Google Web Services

1. Go to http://www.google.com/apis/ and create a Google account. This requires only an email address. After you sign up you will rec

2. Also on http://www.google.com/apis/, download the Google Web APIs developer kit. This includes some sample code in several pr

3. Decompress the developer kit file and find . Copy this file to some permanent location on your local drive. You will need it later in t

Once you have your developer key and your Google WSDL file in a known place, you can start poking around with Google Web Services.

Example 12.12. Introspecting Google Web Services



1. Getting started with Google web services is easy: just create a object and point it at your local copy of Google’s WSDL file.
2. According to the WSDL file, Google offers three functions: , and . These do exactly what they sound like: perform a Google search a
3. The function takes a number of parameters of various types. Note that while the WSDL file can tell you what the arguments are called

Here is a brief synopsis of all the parameters to the function:

- : Your Google API key, which you received when you signed up for Google web services.
- : The search word or phrase you’re looking for. The syntax is exactly the same as Google’s web form, so if you know any advanced
- : The index of the result to start on. Like the interactive web version of Google, this function returns 10 results at a time. If you wan
- : The number of results to return. Currently capped at 10, although you can specify fewer if you are only interested in a few results
- : If , Google will filter out duplicate pages from the results.
- : Set this to plus a country code to get results only from a particular country. Example: to search pages in the United Kingdom. Yo

If Google will filter out porn sites.·
("language restrict") - Set this to a language code to get results only in a particular language.·
and ("input encoding" and "output encoding") - Deprecated, both must be

Example 12.13. Searching Google

```
```

1. After setting up the object, you can call with all ten parameters. Remember to use your own Google API key that you received when
2. There's a lot of information returned, but let's look at the actual search results first. They're stored in and you can access them just li
3. Each element in the is an object that has a and other useful attributes. At this point you can use normal Python introspection techn

The object contains more than the actual search results. It also contains information about the search itself, such as how long it took and how

Example 12.14. Accessing Secondary Information From Google

```
```

1. This search took 0.224919 seconds. That does not include the time spent sending and receiving the actual SOAP XML documents. It's
2. In total, there were approximately 30 million results. You can access them 10 at a time by changing the parameter and calling again.
3. For some queries, Google also returns a list of related categories in the Google Directory (http://directory.google.com/). You can appe

12.8. Troubleshooting SOAP Web Services

Of course, the world of SOAP web services is not all happiness and light. Sometimes things go wrong.

As you've seen throughout this chapter, SOAP involves several layers. There's the HTTP layer, since SOAP is sending XML documents to,

Beyond the underlying HTTP layer, there are a number of things that can go wrong. SOAPpy does an admirable job hiding the SOAP syntax

Here are a few examples of common mistakes that I've made in using SOAP web services, and the errors they generated.

Example 12.15. Calling a Method With an Incorrectly Configured Proxy



1. Did you spot the mistake? You're creating a manually, and you've correctly specified the service URL, but you haven't specified the
2. The server responds by sending a SOAP Fault, which SOAPpy turns into a Python exception of type All errors returned from any SO

Misconfiguring the basic elements of the SOAP service is one of the problems that WSDL aims to solve. The WSDL file contains the service

Example 12.16. Calling a Method With the Wrong Arguments



1. Did you spot the mistake? It's a subtle one: you're calling with an integer instead of a string. As you saw from introspecting the WSD
2. Again, the server returns a SOAP Fault, and the human-readable part of the error gives a clue as to the problem: you're calling a funct

It’s also possible to write Python code that expects a different number of return values than the remote function actually returns.

Example 12.17. Calling a Method and Expecting the Wrong Number of Return Values

```
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
```

1 Did you spot the mistake? [illegible]nly returns one value, a float, but you’ve written code that assumes you’re getting two values and trying t[illegible]

What about Google’s web service? The most common problem I’ve had with it is that I forget to set the application key properly.

Example 12.18. Calling a Method With An Application-Specific Error

```
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
```



❶ Can you spot the mistake? There's nothing wrong with the calling syntax, or the number of arguments, or the datatypes. The problem

❷ The Google server responds with a SOAP Fault and an incredibly long error message, which includes a complete Java stack trace. Rer

Further Reading on Troubleshooting SOAP

New developments for SOAPpy (http://www-106.ibm.com/developerworks/webservices/library/ws-pyth17.html) steps through tryin

12.9. Summary

SOAP web services are very complicated. The specification is very ambitious and tries to cover many different use cases for web services. Th

Before diving into the next chapter, make sure you're comfortable doing all of these things:

- Connecting to a SOAP server and calling remote methods
- Loading a WSDL file and introspecting remote methods
- Debugging SOAP calls with wire traces
- Troubleshooting common SOAP-related errors

Chapter 13. Unit Testing

13.1. Introduction to Roman numerals

In previous chapters, you "dived in" by immediately looking at code and trying to understand it as quickly as possible. Now that you have so

In the next few chapters, you're going to write, debug, and optimize a set of utility functions to convert to and from Roman numerals. You sa

The rules for Roman numerals lead to a number of interesting observations:

1. There is only one correct way to represent a particular number as Roman numerals.
2. The converse is also true: if a string of characters is a valid Roman numeral, it represents only one number (*i.e.* it can only be read o
3. There is a limited range of numbers that can be expressed as Roman numerals, specifically 1 through 9 (The Romans did have several
4. There is no way to represent 0 in Roman numerals. (Amazingly, the ancient Romans had no concept of 0 as a number. Numbers were f
5. There is no way to represent negative numbers in Roman numerals.
6. There is no way to represent fractions or non-integer numbers in Roman numerals.

Given all of this, what would you expect out of a set of functions to convert to and from Roman numerals?

Requirements

1. Should return the Roman numeral representation for all integers 1 to 9.
2. Should fail when given an integer outside the range 1 to 9.
3. Should fail when given a non-integer number.
4. Should take a valid Roman numeral and return the number that it represents.
5. Should fail when given an invalid Roman numeral.
6. If you take a number, convert it to Roman numerals, then convert that back to a number, you should end up with the number you sta
7. Should always return a Roman numeral using uppercase letters.
8. Should only accept uppercase Roman numerals (*i.e.* it should fail when given lowercase input).

Further reading

This site (http://www.wilkiecollins.demon.co.uk/roman/front.htm) has more on Roman numerals, including a fascinating history (ht

13.2. Diving in

Now that you've completely defined the behavior you expect from your conversion functions, you're going to do something a little unexpecte

This is called unit testing, since the set of two conversion functions can be written and tested as a unit, separate from any larger program they

s included with Python 2.1 and later. Python 2.0 users can download it from (http://pyunit.sourceforge.net/).

Unit testing is an important part of an overall testing-centric development strategy. If you write unit tests, it is important to write them early (

- Before writing code, it forces you to detail your requirements in a useful fashion.
- While writing code, it keeps you from over-coding. When all the test cases pass, the function is complete.
- When refactoring code, it assures you that the new version behaves the same way as the old version.
- When maintaining code, it helps you cover your ass when someone comes screaming that your latest change broke their old code. ("
- When writing code in a team, it increases confidence that the code you're about to commit isn't going to break other peoples' code,

13.3. Introducing

This is the complete test suite for your Roman numeral conversion functions, which are yet to be written but will eventually be in It is not in

Example 13.1.

If you have not already done so, you can download this and other examples (http://diveintopython.org/download/diveintopython-examples-5.

Further reading

- The PyUnit home page (http://pyunit.sourceforge.net/) has an in-depth discussion of using the framework (http://pyunit.sourceforge
- The PyUnit FAQ (http://pyunit.sourceforge.net/pyunit.html) explains why test cases are stored separately (http://pyunit.sourceforge.
- *Python Library Reference* (http://www.python.org/doc/current/lib/) summarizes the http://www.python.org/doc/current/lib/module
- ExtremeProgramming.org (http://www.extremeprogramming.org/) discusses why you should write unit tests (http://www.extremepr
- The Portland Pattern Repository (http://www.c2.com/cgi/wiki) has an ongoing discussion of unit tests (http://www.c2.com/cgi/wiki?

13.4. Testing for success

The most fundamental part of unit testing is constructing individual test cases. A test case answers a single question about the code it is testin

A test case should be able to...

- ...run completely by itself, without any human input. Unit testing is about automation.·
- ...determine by itself whether the function it is testing has passed or failed, without a human interpreting the results.·
- ...run in isolation, separate from any other test cases (even if they test the same functions). Each test case is an island.·

Given that, let's build the first test case. You have the following requirement:

> Should return the Roman numeral representation for all integers 1 to 9.

Example 13.2.



```
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]

[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
```

1. To write a test case, first subclass the [illegible] class of the [illegible] module. This class provides many useful methods which you can use in your test ca
2. This is a list of integer/numeral pairs that I verified manually. It includes the lowest ten numbers, the highest number, every number th
3. Every individual test is its own method, which must take no parameters and return no value. If the method exits normally without raisi
4. Here you call the actual [illegible] function. (Well, the function hasn't be written yet, but once it is, this is the line that will call it.) Notice that y
5. Also notice that you are not trapping any exceptions when you call [illegible] This is intentional. [illegible] shouldn't raise an exception when you call it
6. Assuming the [illegible] function was defined correctly, called correctly, completed successfully, and returned a value, the last step is to check w

13.5. Testing for failure

It is not enough to test that functions succeed when given good input; you must also test that they fail when given bad input. And not just any

Remember the other requirements for [illegible]

> [illegible] should fail when given an integer outside the range 1 to 9[illegible]2.
> [illegible] should fail when given a non-integer number.3.

In Python, functions indicate failure by raising exceptions, and the [illegible] module provides methods for testing whether a function raises a particula

Example 13.3. Testing bad input to [illegible]

```
[illegible]
[illegible]
[illegible]
[illegible]

[illegible]
[illegible]
[illegible]

[illegible]
[illegible]
[illegible]

[illegible]
[illegible]
[illegible]
```

❶ The class of the provides the method, which takes the following arguments: the exception you're expecting, the function you're testi

❷ Along with testing numbers that are too large, you need to test numbers that are too small. Remember, Roman numerals cannot expres

❸ Requirement #3 specifies that cannot accept a non-integer number, so here you test to make sure that raises a exception when called

The next two requirements are similar to the first three, except they apply to instead of

should take a valid Roman numeral and return the number that it represents.4.
should fail when given an invalid Roman numeral.5.

Requirement #4 is handled in the same way as requirement #1, iterating through a sampling of known values and testing each in turn. Require

Example 13.4. Testing bad input to

❶ Not much new to say about these; the pattern is exactly the same as the one you used to test bad input to I will briefly note that you h

13.6. Testing for sanity

Often, you will find that a unit of code contains a set of reciprocal functions, usually in the form of conversion functions where one converts

Consider this requirement:

If you take a number, convert it to Roman numerals, then convert that back to a number, you should end up with the number you sta

Example 13.5. Testing against

❶ You've seen the function before, but here it is called with two arguments, which returns a list of integers starting at the first argument

❷ I just wanted to mention in passing that is not a keyword in Python; here it's just a variable name like any other.

❸ The actual testing logic here is straightforward: take a number (convert it to a Roman numeral (then convert it back to a number (

The last two requirements are different from the others because they seem both arbitrary and trivial:

- should always return a Roman numeral using uppercase letters.7.
- should only accept uppercase Roman numerals (*i.e.* it should fail when given lowercase input).8.

In fact, they are somewhat arbitrary. You could, for instance, have stipulated that accept lowercase and mixed case input. But they are not co

Example 13.6. Testing for case

❶ [illegible]

❷ There's a similar lesson to be learned here: even though "you know" that always returns uppercase, you are explicitly converting its re

❸ Note that you're not assigning the return value of to anything. This is legal syntax in Python; if a function returns a value but nobody'

❹ This is a complicated line, but it's very similar to what you did in the and tests. You are testing to make sure that calling a particular

In the next chapter, you'll see how to write code that passes these tests.

---

[6] I can resist everything except temptation." --Oscar Wilde

Chapter 14. Test-First Programming

14.1. [redacted]stage 1

Now that the unit tests are complete, it’s time to start writing the code that the test cases are attempting to test. You’re going to do this in stag

Example 14.1. [redacted]

This file is available in [redacted]n the examples directory.

If you have not already done so, you can download this and other examples (http://diveintopython.org/download/diveintopython-examples-5.

[redacted]

[redacted]

❶ This is how you define your own custom exceptions in Python. Exceptions are classes, and you create your own by subclassing existin

❷ The [redacted]nd [redacted]xceptions will eventually be used by [redacted]o flag various forms of invalid input, as specified in [redacted]

❸ The [redacted]xception will eventually be used by [redacted]o flag invalid input, as specified in [redacted]

❹ At this stage, you want to define the API of each of your functions, but you don’t want to code them yet, so you stub them out using th

Now for the big moment (drum roll please): you’re finally going to run the unit test against this stubby little module. At this point, every test

Run [redacted]with the [redacted]command-line option, which will give more verbose output so you can see exactly what’s going on as each test case runs. Wit

Example 14.2. Output of [redacted]against [redacted]

[redacted]

1. Running the script runs which runs each test case, which is to say each method defined in each class within For each test case, it pri
2. For each failed test case, displays the trace information showing exactly what happened. In this case, the call to also called raised a
3. After the detail, displays a summary of how many tests were performed and how long it took.
4. Overall, the unit test failed because at least one test case did not pass. When a test case doesn't pass, distinguishes between failures an

14.2. stage 2

Now that you have the framework of the module laid out, it's time to start writing code and passing test cases.

Example 14.3.

This file is available in in the examples directory.

If you have not already done so, you can download this and other examples (http://diveintopython.org/download/diveintopython-examples-5.

```
```

1. is a tuple of tuples which defines three things:
   - The character representations of the most basic Roman numerals. Note that this is not just the single-character Roman numera
   - The order of the Roman numerals. They are listed in descending value order, from all the way down to .
   - The value of each Roman numeral. Each inner tuple is a pair of .
2. Here's where your rich data structure pays off, because you don't need any special logic to handle the subtraction rule. To convert to R

Example 14.4. How works

If you're not clear how works, add a statement to the end of the loop:

```
```

```
```

So appears to work, at least in this manual spot check. But will it pass the unit testing? Well no, not entirely.

Example 14.5. Output of against

Remember to run with the command-line flag to enable verbose mode.

```
```

1. does, in fact, always return uppercase, because defines the Roman numeral representations as uppercase. So this test passes already.
2. Here's the big news: this version of the function passes the known values test. Remember, it's not comprehensive, but it does put the
3. However, the function does not "work" for bad values; it fails every single bad input test. That makes sense, because you didn't includ

Here’s the rest of the output of the unit test, listing the details of all the failures. You’re down to 10.





14.3. stage 3

Now that behaves correctly with good input (integers from to), it's time to make it behave correctly with bad input (everything else).

Example 14.6.

This file is available in in the examples directory.

If you have not already done so, you can download this and other examples (http://diveintopython.org/download/diveintopython-examples-5.

❶ This is a nice Pythonic shortcut: multiple comparisons at once. This is equivalent to but it's much easier to read. This is the range che

❷ You raise exceptions yourself with the statement. You can raise any of the built-in exceptions, or you can raise any of your custom ex

❸ This is the non-integer check. Non-integers can not be converted to Roman numerals.

❹ The rest of the function is unchanged.

Example 14.7. Watching handle bad input

[redacted]

Example 14.8. Output of [redacted]against [redacted]

[redacted]

❶ [redacted]till passes the known values test, which is comforting. All the tests that passed in stage 2 still pass, so the latest code hasn't broken a[redacted]

❷ More exciting is the fact that all of the bad input tests now pass. This test, [redacted]passes because of the [redacted]check. When a non-integer is passed[redacted]

❸ This test, [redacted]passes because of the [redacted]check, which raises an [redacted]xception, which is what [redacted]s looking for.

[redacted]

[redacted]

[redacted]

❶ You're down to 6 failures, and all of them involve [redacted]the known values test, the three separate bad input tests, the case check, and the sa

The most important thing that comprehensive unit testing can tell you is when to stop coding. When all the unit tests for a function pass, stop

14.4. [redacted]stage 4

Now that [redacted]s done, it's time to start coding [redacted]Thanks to the rich data structure that maps individual Roman numerals to integer values, this is n

Example 14.9. [redacted]

This file is available in [redacted]n the examples directory.

If you have not already done so, you can download this and other examples (http://diveintopython.org/download/diveintopython-examples-5.

❶ The pattern here is the same as You iterate through your Roman numeral data structure (a tuple of tuples), and instead of matching th

Example 14.10. How works

If you're not clear how works, add a statement to the end of the loop:

**Example 14.11. Output of against **

```
[illegible]
[illegible]
[illegible]
```

14.5. stage 5

Now that works properly with good input, it's time to fit in the last piece of the puzzle: making it work properly with bad input. That means

If you're not familiar with regular expressions and didn't read Chapter 7, *Regular Expressions*, now would be a good time.

As you saw in Section 7.3, "Case Study: Roman Numerals", there are several simple rules for constructing a Roman numeral, using the letter

> Characters are additive. is , is , and is is (literally, "5 and 1"), is , and is .
> The tens characters (, , and) can be repeated up to three times. At , you need to subtract from the next highest fives character. You
> Similarly, at , you need to subtract from the next highest tens character: is but is ("1 less than "), not since the character can not b
> The fives characters can not be repeated. is always represented as , never as is always , never .
> Roman numerals are always written highest to lowest, and read left to right, so order of characters matters very much. is , is a comp

Example 14.12.

This file is available in in the examples directory.

If you have not already done so, you can download this and other examples (http://diveintopython.org/download/diveintopython-examples-5.

```
```

```
```

```
```

1. One thing I didn’t mention about regular expressions is that, by default, they are case-sensitive. Since the regular expression was expr
2. More importantly, the bad input tests pass. For instance, the malformed antecedents test checks cases like As you’ve seen, this does n
3. In fact, all the bad input tests pass. This regular expression catches everything you could think of when you made your test cases.
4. And the anticlimax award of the year goes to the word " which is printed by the module when all the tests pass.

When all of your tests pass, stop coding.

Chapter 15. Refactoring

15.1. Handling bugs

Despite your best efforts to write comprehensive unit tests, bugs happen. What do I mean by "bug"? A bug is a test case you haven't written y

Example 15.1. The bug

```
0
```

1 Remember in the previous section when you kept seeing that an empty string would match the regular expression you were using to ch

After reproducing the bug, and before fixing it, you should write a test case that fails, thus illustrating the bug.

Example 15.2. Testing for the bug (

```
```

1 Pretty simple stuff here. Call with an empty string and make sure it raises an xception. The hard part was finding the bug; now that y

Since your code has a bug, and you now have a test case that tests this bug, the test case will fail:

Example 15.3. Output of against

```
=
-
```

Now you can fix the bug.

Example 15.4. Fixing the bug ([redacted]

This file is available in [redacted]n the examples directory.

[redacted]

❶ Only two lines of code are required: an explicit check for an empty string, and a [redacted]tatement.

Example 15.5. Output of [redacted]against [redacted]

[redacted]

❶ The blank string test case now passes, so the bug is fixed.

❷ All the other test cases still pass, which means that this bug fix didn't break anything else. Stop coding.

Coding this way does not make fixing bugs any easier. Simple bugs (like this one) require simple test cases; complex bugs will require compl

15.2. Handling changing requirements

Despite your best efforts to pin your customers to the ground and extract exact requirements from them on pain of horrible nasty things involv

Suppose, for instance, that you wanted to expand the range of the Roman numeral conversion functions. Remember the rule that said that no

Example 15.6. Modifying test cases for new requirements (

This file is available in in the examples directory.

If you have not already done so, you can download this and other examples (http://diveintopython.org/download/diveintopython-examples-5.





❶ The existing known values don’t change (they’re all still reasonable values to test), but you need to add a few more in the range. Here

❷ The definition of "large input" has changed. This test used to call with and expect an error; now that are good values, you need to bu

❸ The definition of "too many repeated numerals" has also changed. This test used to call with and expect an error; now that is conside

❹ The sanity check and case checks loop through every number in the range, from to Since the range has now expanded, these loops n

Now your test cases are up to date with the new requirements, but your code is not, so you expect several of the test cases to fail.

Example 15.7. Output of against

```
```

❶ Our case checks now fail because they loop from to but only accepts numbers from to so it will fail as soon the test case hits

❷ The known values test will fail as soon as it hits because still thinks this is an invalid Roman numeral.

❸ The known values test will fail as soon as it hits because still thinks this is out of range.

❹ The sanity check will also fail as soon as it hits because still thinks this is out of range.

```
```

Now that you have test cases that fail due to the new requirements, you can think about fixing the code to bring it in line with the test cases. (

Example 15.8. Coding the new requirements (

This file is available in the examples directory.

❶ only needs one small change, in the range check. Where you used to check you now check And you change the error message that

❷ You don’t need to make any changes to at all. The only change is to if you look closely, you’ll notice that you added another optiona

You may be skeptical that these two small changes are all that you need. Hey, don’t take my word for it; see for yourself:

Example 15.9. Output of against

```
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]

-
[illegible]

[illegible]
```

1. All the test cases pass. Stop coding.

Comprehensive unit testing means never having to rely on a programmer who says "Trust me."

15.3. Refactoring

The best thing about comprehensive unit testing is not the feeling you get when all your test cases finally pass, or even the feeling you get wh

Refactoring is the process of taking working code and making it work better. Usually, "better" means "faster", although it can also mean "usin

Here, "better" means "faster". Specifically, the [illegible]unction is slower than it needs to be, because of that big nasty regular expression that you us

Example 15.10. Compiling regular expressions

```
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
```

1. This is the syntax you've seen before: [illegible]akes a regular expression as a string ([illegible]and a string to match against it ([illegible] If the pattern matches
2. This is the new syntax: [illegible]akes a regular expression as a string and returns a pattern object. Note there is no string to match here. Compi
3. The compiled pattern object returned from [illegible]as several useful-looking functions, including several (like [illegible]and [illegible]that are available direct
4. Calling the compiled pattern object's [illegible]unction with the string [illegible]ccomplishes the same thing as calling [illegible]with both the regular expressio

Whenever you are going to use a regular expression more than once, you should compile it to get a pattern object, then call the methods on th

Example 15.11. Compiled regular expressions in [illegible]

This file is available in [illegible]n the examples directory.

If you have not already done so, you can download this and other examples (http://diveintopython.org/download/diveintopython-examples-5.

1. This looks very similar, but in fact a lot has changed. is no longer a string; it is a pattern object which was returned from
2. That means that you can call methods on directly. This will be much, much faster than calling every time. The regular expression is c

So how much faster is it to compile regular expressions? See for yourself:

Example 15.12. Output of against

1.
2.
3.

1. Just a note in passing here: this time, I ran the unit test *without* the option, so instead of the full for each test, you only get a dot for ea
2. You ran tests in seconds, compared to seconds without precompiling the regular expressions. That's an improvement overall, and re
3. Oh, and in case you were wondering, precompiling the regular expression didn't break anything, and you just proved it.

There is one other performance optimization that I want to try. Given the complexity of regular expression syntax, it should come as no surpr

Example 15.13.

This file is available in in the examples directory.

If you have not already done so, you can download this and other examples (http://diveintopython.org/download/diveintopython-examples-5.

```
```

1. You have replaced with Both mean the same thing: "match 0 to 4 characters". Similarly, became ("match 0 to 3 characters") and s

This form of the regular expression is a little shorter (though not any more readable). The big question is, is it any faster?

Example 15.14. Output of against

```
```

1.

2.

1. Overall, the unit tests run 2% faster with this form of regular expression. That doesn't sound exciting, but remember that the function
2. More important than any performance boost is the fact that the module still works perfectly. This is the freedom I was talking about ea

One other tweak I would like to make, and then I promise I'll stop refactoring and put this module to bed. As you've seen repeatedly, regular

Example 15.15.

This file is available in in the examples directory.

If you have not already done so, you can download this and other examples (http://diveintopython.org/download/diveintopython-examples-5.

```
```

```
```

```
```

1. The function can take an optional second argument, which is a set of one or more flags that control various options about the compile

Example 15.16. Output of against

```
```

❶

❷

❶ This new, "verbose" version runs at exactly the same speed as the old version. In fact, the compiled pattern objects are the same, since

❷ This new, "verbose" version passes all the same tests as the old version. Nothing has changed, except that the programmer who comes

15.4. Postscript

A clever reader read the previous section and took it to the next level. The biggest headache (and performance drain) in the program as it is cu

And best of all, he already had a complete set of unit tests. He changed over half the code in the module, but the unit tests stayed the same, so

Example 15.17. ❶

This file is available in ❶n the examples directory.

If you have not already done so, you can download this and other examples (http://diveintopython.org/download/diveintopython-examples-5.



So how fast is it?

Example 15.18. Output of against

Remember, the best performance you ever got in the original version was 13 tests in 3.315 seconds. Of course, it’s not entirely a fair compari

The moral of the story?

- Simplicity is a virtue.·

Especially when regular expressions are involved.·
And unit tests can give you the confidence to do large-scale refactoring... even if you didn’t write the original code.·

15.5. Summary

Unit testing is a powerful concept which, if properly implemented, can both reduce maintenance costs and increase flexibility in any long-ter

This chapter covered a lot of ground, and much of it wasn’t even Python-specific. There are unit testing frameworks for many languages, all

Designing test cases that are specific, automated, and independent·
Writing test cases *before* the code they are testing·
Writing tests that test good input and check for proper results·
Writing tests that test bad input and check for proper failures·
Writing and updating test cases to illustrate bugs or reflect new requirements·
Refactoring mercilessly to improve performance, scalability, readability, maintainability, or whatever other -ility you’re lacking·

Additionally, you should be comfortable doing all of the following Python-specific things:

Subclassing [redacted]and writing methods for individual test cases·
Using [redacted]to check that a function returns a known value·
Using [redacted]to check that a function raises a known exception·
Calling [redacted]in your [redacted]clause to run all your test cases at once·
Running unit tests in verbose or regular mode·

Further reading

XProgramming.com (http://www.xprogramming.com/) has links to download unit testing frameworks (http://www.xprogramming.c

Chapter 16. Functional Programming

16.1. Diving in

In Chapter 13, *Unit Testing*, you learned about the philosophy of unit testing. In Chapter 14, *Test-First Programming*, you stepped through th

The following is a complete Python program that acts as a cheap and simple regression testing framework. It takes unit tests that you've writt

Example 16.1.

If you have not already done so, you can download this and other examples (http://diveintopython.org/download/diveintopython-examples-5.

Running this script in the same directory as the rest of the example scripts that come with this book will find all the unit tests, named run the

Example 16.2. Sample output of

❶ The first 5 tests are from [redacted] which tests the example script from Chapter 4, *The Power Of Introspection*.

❷ The next 5 tests are from [redacted] which tests the example script from Chapter 2, *Your First Python Program*.

❸ The rest are from [redacted] which you studied in depth in Chapter 13, *Unit Testing*.

16.2. Finding the path

When running Python scripts from the command line, it is sometimes useful to know where the currently running script is located on disk.

This is one of those obscure little tricks that is virtually impossible to figure out on your own, but simple to remember once you see it. The ke

Example 16.3. [redacted]

If you have not already done so, you can download this and other examples (http://diveintopython.org/download/diveintopython-examples-5.

[redacted]

❶ Regardless of how you run a script, [redacted] will always contain the name of the script, exactly as it appears on the command line. This may o

❷ [redacted] takes a filename as a string and returns the directory path portion. If the given filename does not include any path information, [redacted] return

❸ [redacted] as the key here. It takes a pathname, which can be partial or even blank, and returns a fully qualified pathname.

leserves further explanation. It is very flexible; it can take any kind of pathname.

Example 16.4. Further explanation of

```
```

1. eturns the current working directory.
2. Calling with an empty string returns the current working directory, same as
3. Calling with a partial pathname constructs a fully qualified pathname out of it, based on the current working directory.
4. Calling with a full pathname simply returns it.
5. lso *normalizes* the pathname it returns. Note that this example worked even though I don't actually have a 'foo' directory. never che

The pathnames and filenames you pass to do not need to exist.

not only constructs full path names, it also normalizes them. That means that if you are in the directory, will return It normalizes the path

Example 16.5. Sample output from

```
```

1. In the first case, ncludes the full path of the script. You can then use the unction to strip off the script name and return the full direct
2. If the script is run by using a partial pathname, will still contain exactly what appears on the command line. will then give you a parti
3. If the script is run from the current directory without giving any path, will simply return an empty string. Given an empty string, retu

Like the other functions in the sand modules, is cross-platform. Your results will look slightly different than my examples if you're running
Addendum. One reader was dissatisfied with this solution, and wanted to be able to run all the unit tests in the current directory, not the direc

Example 16.6. Running scripts in the current directory

❶ Instead of setting to the directory where the currently running script is located, you set it to the current working directory instead. This
❷ Append this directory to the Python library search path, so that when you dynamically import the unit test modules later, Python can fi
❸ The rest of the function is the same.

This technique will allow you to re-use this script on multiple projects. Just put the script in a common directory, then change to the project's

16.3. Filtering lists revisited

You're already familiar with using list comprehensions to filter lists. There is another way to accomplish this same thing, which some people

The function passed as the first argument to itself take one argument, and the return value. Python has a built-in function which takes two arguments, a function and a list, and returns a list. The function passed as the first argument to itself take one argument, and the return value. Returns will contain all the elements from the list

Got all that? It's not as difficult as it sounds.

Example 16.7. Introducing

❶ uses the built-in mod function "% to return if is odd and if is even.
❷ takes two arguments, a function (and a list (. It loops through the list and calls with each element. If returns a true value (remembe
❸ You could accomplish the same thing using list comprehensions, as you saw in Section 4.5, "Filtering Lists".
❹ You could also accomplish the same thing with a loop. Depending on your programming background, this may seem more "straightfo

Example 16.8. In

```
```

1. As you saw in Section 16.2, “Finding the path”, may contain the full or partial pathname of the directory of the currently running scri
2. This is a compiled regular expression. As you saw in Section 15.3, “Refactoring”, if you’re going to use the same regular expression o
3. For each element in the list, you’re going to call the method of the compiled regular expression object, If the regular expression mat

Historical note. Versions of Python prior to 2.0 did not have list comprehensions, so you couldn’t filter using list comprehensions; the funct

Example 16.9. Filtering using list comprehensions instead

```
```

1. This will accomplish exactly the same result as using the function. Which way is more expressive? That’s up to you.

16.4. Mapping lists revisited

You’re already familiar with using list comprehensions to map one list into another. There is another way to accomplish the same thing, using

Example 16.10. Introducing

```
```

1. Calls the function with each element of the list in order. In this case, the function simply multiplies each el
2. You could accomplish the same thing with a list comprehension. List comprehensions were first introduced in Python 2.0; has been a
3. You could, if you insist on thinking like a Visual Basic programmer, use a loop to accomplish the same thing.

Example 16.11. with lists of mixed datatypes

```
```

1. As a side note, I’d like to point out that works just as well with lists of mixed datatypes, as long as the function you’re using correctly

All right, enough play time. Let’s look at some real code.

Example 16.12. in

1. As you saw in Section 4.7, “Using lambda Functions”, defines an inline function. And as you saw in Example 6.17, “Splitting Pathna
2. Calling makes each filename listed in passes it to the function and returns a list of the return values of each of those function calls. In

As you’ll see in the rest of the chapter, you can extend this type of data-centric thinking all the way to the final goal, which is to define and ex

16.5. Data-centric programming

By now you’re probably scratching your head wondering why this is better than using loops and straight function calls. And that’s a perfectly

In this case, you started with no data at all; the first thing you did was get the directory path of the current script, and got a list of files in that

However, you knew you didn’t care about all of those files, only the ones that were actually test suites. You had *too much data*, so you neede

Now you had the filenames of each of the test suites (and only the test suites, since everything else had been filtered out), but you really want

Instead of you could have used a loop with an statement. Instead of you could have used a loop with a function call. But using loops like

Oh sure, you could try to be fancy and delete elements in place without creating a new list. But you’ve been burned by that before. Trying to

I resisted list comprehensions when I first learned Python, and I resisted and even longer. I insisted on making my life more difficult, stickin

Let it all go. Busywork code is not important. Data is important. And data is not difficult. It’s only data. If you have too much, filter it. If it’s

16.6. Dynamically importing modules

OK, enough philosophizing. Let’s talk about dynamically importing modules.

First, let’s look at how you normally import modules. The syntax looks in the search path for the named module and imports it by name. You

Example 16.13. Importing multiple modules at once

1. This imports four modules at once: for system functions and access to the command line parameters), for operating system function

Now let’s do the same thing, but with dynamic imports.

Example 16.14. Importing modules dynamically

1. The built-in function accomplishes the same goal as using the statement, but it's an actual function, and it takes a string as an argume
2. The variable is now the module, just as if you had said The variable is now the module, and so forth.

So imports a module, but takes a string argument to do it. In this case the module you imported was just a hard-coded string, but it could just

Example 16.15. Importing a list of modules dynamically



1. is just a list of strings. Nothing fancy, except that the strings happen to be names of modules that you could import, if you wanted to.
2. Surprise, you wanted to import them, and you did, by mapping the function onto the list. Remember, this takes each element of the lis
3. So now from a list of strings, you've created a list of actual modules. (Your paths may be different, depending on your operating syste
4. To drive home the point that these are real modules, let's look at some module attributes. Remember, is the module, so is All the otl

Now you should be able to put this all together and figure out what most of this chapter's code sample is doing.

16.7. Putting it all together

You've learned enough now to deconstruct the first seven lines of this chapter's code sample: reading a directory and importing selected mod

Example 16.16. The function



Let's look at it line by line, interactively. Assume that the current directory is which contains the examples that come with this book, includi

Example 16.17. Step 1: Get all the files

❶ [redacted]s a list of all the files and directories in the script's directory. (If you've been running some of the examples already, you may also se

Example 16.18. Step 2: Filter to find the files you care about

❶ This regular expression will match any string that ends with [redacted]Note that you need to escape the period, since a period in a regular expre

❷ The compiled regular expression acts like a function, so you can use it to filter the large list of files and directories, to find the ones tha

❸ And you're left with the list of unit testing scripts, because they were the only ones named [redacted]

Example 16.19. Step 3: Map filenames to module names

❶ As you saw in Section 4.7, "Using lambda Functions", [redacted]s a quick-and-dirty way of creating an inline, one-line function. This one take

❷ [redacted]s a function. There's nothing magic about [redacted]unctions as opposed to regular functions that you define with a [redacted]tatement. You can call t

❸ Now you can apply this function to each file in the list of unit test files, using [redacted]

❹ And the result is just what you wanted: a list of modules, as strings.

Example 16.20. Step 4: Mapping module names to modules

```
[
]
```

1. As you saw in Section 16.6, “Dynamically importing modules”, you can use a combination of and to map a list of module names (as
2. is now a list of modules, fully accessible like any other module.
3. The last module in the list *is* the module, just as if you had said

Example 16.21. Step 5: Loading the modules into a test suite

```
.
]
]
```

1. These are real module objects. Not only can you access them like any other module, instantiate classes and call functions, you can also
2. Finally, you wrap the list of objects into one big test suite. The module has no problem traversing this tree of nested test suites within

This introspection process is what the module usually does for us. Remember that magic-looking function that our individual test modules c

Example 16.22. Step 6: Telling to use your test suite

```
```

1. Instead of letting the module do all its magic for us, you’ve done most of it yourself. You’ve created a function (that imports the mo

16.8. Summary

The program and its output should now make perfect sense.

You should now feel comfortable doing all of these things:

- Manipulating path information from the command line.
- Filtering lists using instead of list comprehensions.
- Mapping lists using instead of list comprehensions.
- Dynamically importing modules.

[1] Technically, the second argument to can be any sequence, including lists, tuples, and custom classes that act like lists by defining the specia

[2] Again, I should point out that map can take a list, a tuple, or any object that acts like a sequence. See previous footnote about filter.

Chapter 17. Dynamic functions

17.1. Diving in

I want to talk about plural nouns. Also, functions that return other functions, advanced regular expressions, and generators. Generators are ne

If you haven't read Chapter 7, *Regular Expressions*, now would be a good time. This chapter assumes you understand the basics of regular ex

English is a schizophrenic language that borrows from a lot of other languages, and the rules for making singular nouns into plural nouns are

If you grew up in an English-speaking country or learned English in a formal school setting, you're probably familiar with the basic rules:

> If a word ends in S, X, or Z, add ES. "Bass" becomes "basses", "fax" becomes "faxes", and "waltz" becomes "waltzes".1.
> If a word ends in a noisy H, add ES; if it ends in a silent H, just add S. What's a noisy H? One that gets combined with other letters t
> If a word ends in Y that sounds like I, change the Y to IES; if the Y is combined with a vowel to sound like something else, just add
> If all else fails, just add S and hope for the best.4.

(I know, there are a lot of exceptions. "Man" becomes "men" and "woman" becomes "women", but "human" becomes "humans". "Mouse" be

Other languages are, of course, completely different.

Let's design a module that pluralizes nouns. Start with just English nouns, and just these four rules, but keep in mind that you'll inevitably ne

17.2. [illegible] stage 1

So you're looking at words, which at least in English are strings of characters. And you have rules that say you need to find different combina

Example 17.1. [illegible]

```
[illegible]

[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
```

1. OK, this is a regular expression, but it uses a syntax you didn't see in Chapter 7, *Regular Expressions*. The square brackets mean "mat
2. This [illegible]unction performs regular expression-based string substitutions. Let's look at it in more detail.

Example 17.2. Introducing [illegible]

```
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
```

❶ Does the string contain a, b, or c? Yes, it contains a.

❷ OK, now find a, b, or c, and replace it with o. Abbecomes Ab.

❸ The same function turns Ab into Ab.

❹ You might think this would turn Ab into Ab, but it doesn't. Ab replaces *all* of the matches, not just the first one. So this regular expression turn

Example 17.3. Back to

❶ Back to the function. What are you doing? You're replacing the end of string with s. In other words, adding s to the string. You could a

❷ Look closely, this is another new variation. The ^ as the first character inside the square brackets means something special: negation. Ab

❸ Same pattern here: match words that end in Y, where the character before the Y is *not* a, e, i, o, or u. You're looking for words that end in Y

Example 17.4. More on negation regular expressions

```
>
>
>
```

❶ matches this regular expression, because it ends in y, and c is not a, e, i, o, or u.

❷ does not match, because it ends in y, and you specifically said that the character before the y could not be o. does not match, because it e

❸ does not match, because it does not end in y.

Example 17.5. More on

❶ This regular expression turns into and into which is what you wanted. Note that it would also turn into but that will never happen

❷ Just in passing, I want to point out that it is possible to combine these two regular expressions (one to find out if the rule applies, and a

Regular expression substitutions are extremely powerful, and the syntax makes them even more powerful. But combining the entire operatio

17.3. stage 2

Now you're going to add a level of abstraction. You started by defining a list of rules: if this, then do that, otherwise go to the next rule. Let's

Example 17.6.

❶ This version looks more complicated (it's certainly longer), but it does exactly the same thing: try to match four different rules, in orde

❷ Using a [redacted]oop, you can pull out the match and apply rules two at a time (one match, one apply) from the [redacted]uple. On the first iteration of

❸ Remember that everything in Python is an object, including functions. [redacted]ontains actual functions; not names of functions, but actual fu

❹ On the first iteration of the [redacted]oop, this is equivalent to calling [redacted]and so forth.

If this additional level of abstraction is confusing, try unrolling the function to see the equivalence. This [redacted]oop is equivalent to the following:

Example 17.7. Unrolling the [redacted]function

[redacted]

The benefit here is that that [redacted]unction is now simplified. It takes a list of rules, defined elsewhere, and iterates through them in a generic fashi

Now, was adding this level of abstraction worth it? Well, not yet. Let’s consider what it would take to add a new rule to the function. Well, in

This is really just a stepping stone to the next section. Let’s move on.

17.4. [redacted]stage 3

Defining separate named functions for each match and apply rule isn’t really necessary. You never call them directly; you define them in the

Example 17.8. [redacted]

[redacted]

[redacted]
(
(
[redacted]
[redacted]
)
(
[redacted]
[redacted]
)
(
[redacted]
[redacted]
)
(
[redacted]
[redacted]
)
❶

[redacted]

```
[illegible]
[illegible]
[illegible]
```

1. This is the same set of rules as you defined in stage 2. The only difference is that instead of defining named functions like [illegible] and [illegible] you h
2. Note that the [illegible] function hasn't changed at all. It iterates through a set of rule functions, checks the first rule, and if it returns a true value

Now to add a new rule, all you need to do is define the functions directly in the [illegible] list itself: one match rule, and one apply rule. But defining th

17.5. [illegible] stage 4

Let's factor out the duplication in the code so that defining new rules can be easier.

Example 17.9. [illegible]

```
[illegible]


[illegible]
[illegible]
[illegible]
[illegible]
```

1. [illegible] is a function that builds other functions dynamically. It takes [illegible] and [illegible] (actually it takes a tuple, but more on that in a minute), and you ca
2. Building the apply function works the same way. The apply function is a function that takes one parameter, and calls [illegible] with the [illegible] and [illegible] pa
3. Finally, the [illegible] function returns a tuple of two values: the two functions you just created. The constants you defined within those function

If this is incredibly confusing (and it should be, this is weird stuff), it may become clearer when you see how to use it.

Example 17.10. [illegible] continued

```
[illegible]
(
[illegible]
[illegible]
[illegible]
$
[illegible]
[illegible]
```

1. Our pluralization rules are now defined as a series of strings (not functions). The first string is the regular expression that you would u
2. This line is magic. It takes the list of strings in [illegible] and turns them into a list of functions. How? By mapping the strings to the [illegible] function, w

I swear I am not making this up: [illegible] ends up with exactly the same list of functions as the previous example. Unroll the [illegible] definition, and you'll ge

Example 17.11. Unrolling the rules definition

```
[illegible]
(
(
[illegible]
[illegible]
)
```

```
(
[illegible]
[illegible]
)
(
[illegible]
[illegible]
)
(
[illegible]
[illegible]
)
)
```

Example 17.12. [illegible]finishing up

```
[illegible]
[illegible]
[illegible]
[illegible]
```

❶ Since the [illegible]ist is the same as the previous example, it should come as no surprise that the [illegible]unction hasn't changed. Remember, it's con

Just in case that wasn't mind-blowing enough, I must confess that there was a subtlety in the definition of [illegible]hat I skipped over. Let's go back a

Example 17.13. Another look at [illegible]

```
[illegible]
```

❶ Notice the double parentheses? This function doesn't actually take three parameters; it actually takes one parameter, a tuple of three el

Example 17.14. Expanding tuples when calling functions

```
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
```

❶ The proper way to call the function [illegible]s with a tuple of three elements. When the function is called, the elements are assigned to differen

Now let's go back and see why this auto-tuple-expansion trick was necessary. [illegible]was a list of tuples, and each tuple had three elements. When y

17.6. [illegible]stage 5

You've factored out all the duplicate code and added enough abstractions so that the pluralization rules are defined in a list of strings. The ne

First, let's create a text file that contains the rules you want. No fancy data structures, just space- (or tab-)delimited strings in three columns.

Example 17.15.

```
$
```

Now let’s see how you can use this rules file.

Example 17.16.

```
❶
❷
❸
❹
❺
❻
```

❶ You’re still using the closures technique here (building a function dynamically that uses variables defined outside the function), but no

❷ Our function now takes an optional second parameter, which defaults to

❸ You use the parameter to construct a filename, then open the file and read the contents into a list. If is , then you’ll open the file, read

❹ As you saw, each line in the file really has three values, but they’re separated by whitespace (tabs or spaces, it makes no difference). M

❺ If is a list of tuples, then will be a list of the functions created dynamically by each call to Calling returns a function that takes a sin

❻ Because you’re now building a combined match-and-apply function, you need to call it differently. Just call the function, and if it retu

So the improvement here is that you’ve completely separated the pluralization rules into an external file. Not only can the file be maintained s

The downside here is that you’re reading that file every time you call the function. I thought I could get through this entire book without usin

17.7. stage 6

Now you’re ready to talk about generators.

Example 17.17.

```
❶

❷
❸
❹
❺

❻
❼
❽
```

This uses a technique called generators, which I'm not even going to try to explain until you look at a simpler example first.

Example 17.18. Introducing generators

```
1
2
3
4
5
6
```

1. The presence of the keyword in means that this is not a normal function. It is a special kind of function which generates values one at
2. To create an instance of the generator, just call it like any other function. Note that this does not actually execute the function code. Y
3. The function returns a generator object.
4. The first time you call the method on the generator object, it executes the code in up to the first statement, and then returns the value
5. Repeatedly calling on the generator object *resumes where you left off* and continues until you hit the next statement. The next line of
6. The second time you call you do all the same things again, but this time is now 4. And so forth. Since sets up an infinite loop, you co

Example 17.19. Using generators instead of recursion

```
1
2
3
```

1. The Fibonacci sequence is a sequence of numbers where each number is the sum of the two numbers before it. It starts with 0 and 1, goes
2. is the current number in the sequence, so yield it.
3. is the next number in the sequence, so assign that to but also calculate the next value (and assign that to for later use. Note that this

So you have a function that spits out successive Fibonacci numbers. Sure, you could do that with recursion, but this way is easier to read. Als

Example 17.20. Generators in loops

1. You can use a generator like in a for loop directly. The for loop will create the generator object and successively call the next method to get valu
2. Each time through the for loop, n gets a new value from the yield statement in and all you do is print it out. Once runs out of numbers (a gets b

OK, let's go back to the def function and see how you're using this.

Example 17.21. Generators that generate dynamic functions

1.
2.
3.
4.

1.
2.
3.
4.

1. is a common idiom for reading lines from a file, one line at a time. It works because *actually returns a generator* whose next method retur
2. No magic here. Remember that the lines of the rules file have three values separated by whitespace, so returns a tuple of 3 values, and
3. *And then you yield.* What do you yield? A function, built dynamically with that is actually a closure (it uses the local variables and
4. Since is a generator, you can use it directly in a for loop. The first time through the for loop, you will call the def function, which will open the

What have you gained over stage 5? In stage 5, you read the entire rules file and built a list of all the possible rules before you even tried the f

Further reading

- PEP 255 (http://www.python.org/peps/pep-0255.html) defines generators.
- Python Cookbook (http://www.activestate.com/ASPN/Python/Cookbook/) has many more examples of generators (http://www.goog

17.8. Summary

You talked about several different advanced techniques in this chapter. Not all of them are appropriate for every situation.

You should now be comfortable with all of these techniques:

- Performing string substitution with regular expressions.
- Treating functions as objects, storing them in lists, assigning them to variables, and calling them through those variables.
- Building dynamic functions with
- Building closures, dynamic functions that contain surrounding variables as constants.
- Building generators, resumable functions that perform incremental logic and return different values each time you call them.

Adding abstractions, building functions dynamically, building closures, and using generators can all make your code simpler, more readable,

Chapter 18. Performance Tuning

Performance tuning is a many-splendored thing. Just because Python is an interpreted language doesn't mean you shouldn't worry about code

18.1. Diving in

There are so many pitfalls involved in optimizing your code, it's hard to know where to start.

Let's start here: *are you sure you need to do it at all?* Is your code really so bad? Is it worth the time to tune it? Over the lifetime of your appl

Second, *are you sure you're done coding?* Premature optimization is like spreading frosting on a half-baked cake. You spend hours or days (

This is not to say that code optimization is worthless, but you need to look at the whole system and decide whether it's the best use of your ti

Oh yes, unit tests. It should go without saying that you need a complete set of unit tests before you begin performance tuning. The last thing y

With these caveats in place, let's look at some techniques for optimizing Python code. The code in question is an implementation of the Soun

There are several subtle variations of the Soundex algorithm. This is the one used in this chapter:

1. Keep the first letter of the name as-is.
2. Convert the remaining letters to digits, according to a specific table:
   - B, F, P, and V become 1.
   - C, G, J, K, Q, S, X, and Z become 2.
   - D and T become 3.
   - L becomes 4.
   - M and N become 5.
   - R becomes 6.
   - All other letters become 9.
3. Remove consecutive duplicates.
4. Remove all 9s altogether.
5. If the result is shorter than four characters (the first letter plus three digits), pad the result with trailing zeros.
6. if the result is longer than four characters, discard everything after the fourth character.

For example, my name, becomes P942695. That has no consecutive duplicates, so nothing to do there. Then you remove the 9s, leaving P42

Another example: becomes W99, which becomes W9, which becomes W, which gets padded with zeros to become W000.

Here's a first attempt at a Soundex function:

Example 18.1.

If you have not already done so, you can download this and other examples (http://diveintopython.org/download/diveintopython-examples-5.

Further Reading on Soundex

Soundexing and Genealogy (http://www.avotaynu.com/soundex.html) gives a chronology of the evolution of the Soundex and its re

18.2. Using the Module

The most important thing you need to know about optimizing Python code is that you shouldn't write your own timing function.

Timing short pieces of code is incredibly complex. How much processor time is your computer devoting to running this code? Are there thing

And then there's the matter of the variations introduced by the timing framework itself. Does the Python interpreter cache method name look

The Python community has a saying: "Python comes with batteries included." Don't write your own timing framework. Python 2.3 comes wi

Example 18.2. Introducing

If you have not already done so, you can download this and other examples (http://diveintopython.org/download/diveintopython-examples-5.

```
1
2
3
```

1. The module defines one class, which takes two arguments. Both arguments are strings. The first argument is the statement you wish
2. Once you have the object, the easiest thing to do is call which calls your function 1 million times and returns the number of seconds
3. The other major method of the object is which takes two optional arguments. The first argument is the number of times to repeat the

You can use the module on the command line to test an existing Python program, without modifying the code. See http://docs.python.org/lib

Note that returns a list of times. The times will almost never be identical, due to slight variations in how much processor time the Python inte

In fact, that's almost certainly wrong. The tests that took longer didn't take longer because of variations in your code or in the Python interpre

Python has a handy function that takes a list and returns the smallest value:

The module only works if you already know what piece of code you need to optimize. If you have a larger Python program and don't know

18.3. Optimizing Regular Expressions

The first thing the Soundex function checks is whether the input is a non-empty string of letters. What's the best way to do this?

If you answered "regular expressions", go sit in the corner and contemplate your bad instincts. Regular expressions are almost never the right

This code fragment from checks whether the function argument is a word made entirely of letters, with at least one letter (not the empty stri

How does perform? For convenience, the section of the script contains this code that calls the module, sets up a timing test with three differ

So how does perform with this regular expression?

As you might expect, the algorithm takes significantly longer when called with longer names. There will be a few things we can do to narrow

The other thing to keep in mind is that we are testing a representative sample of names. is a kind of trivial case, in that it gets shorted down t

So what about that regular expression? Well, it's inefficient. Since the expression is testing for ranges of characters (A–Z uppercase, and a–z lo

says is slightly faster than but nothing to get terribly excited about:

We saw in Section 15.3, “Refactoring” that regular expressions can be compiled and reused for faster results. Since this regular expression ne

```
0
```

Using a compiled regular expression in s significantly faster:

```
```

But is this the wrong path? The logic here is simple: the input needs to be non-empty, and it needs to be composed entirely of letters. Would

Here is

```
0
```

It turns out that this technique in s *not* faster than using a compiled regular expression (although it is faster than using a non-compiled regula

```
```

Why isn’t aster? The answer lies in the interpreted nature of Python. The regular expression engine is written in C, and compiled to run nati

It turns out that Python offers an obscure string method. You can be excused for not knowing about it, since it’s never been mentioned in this

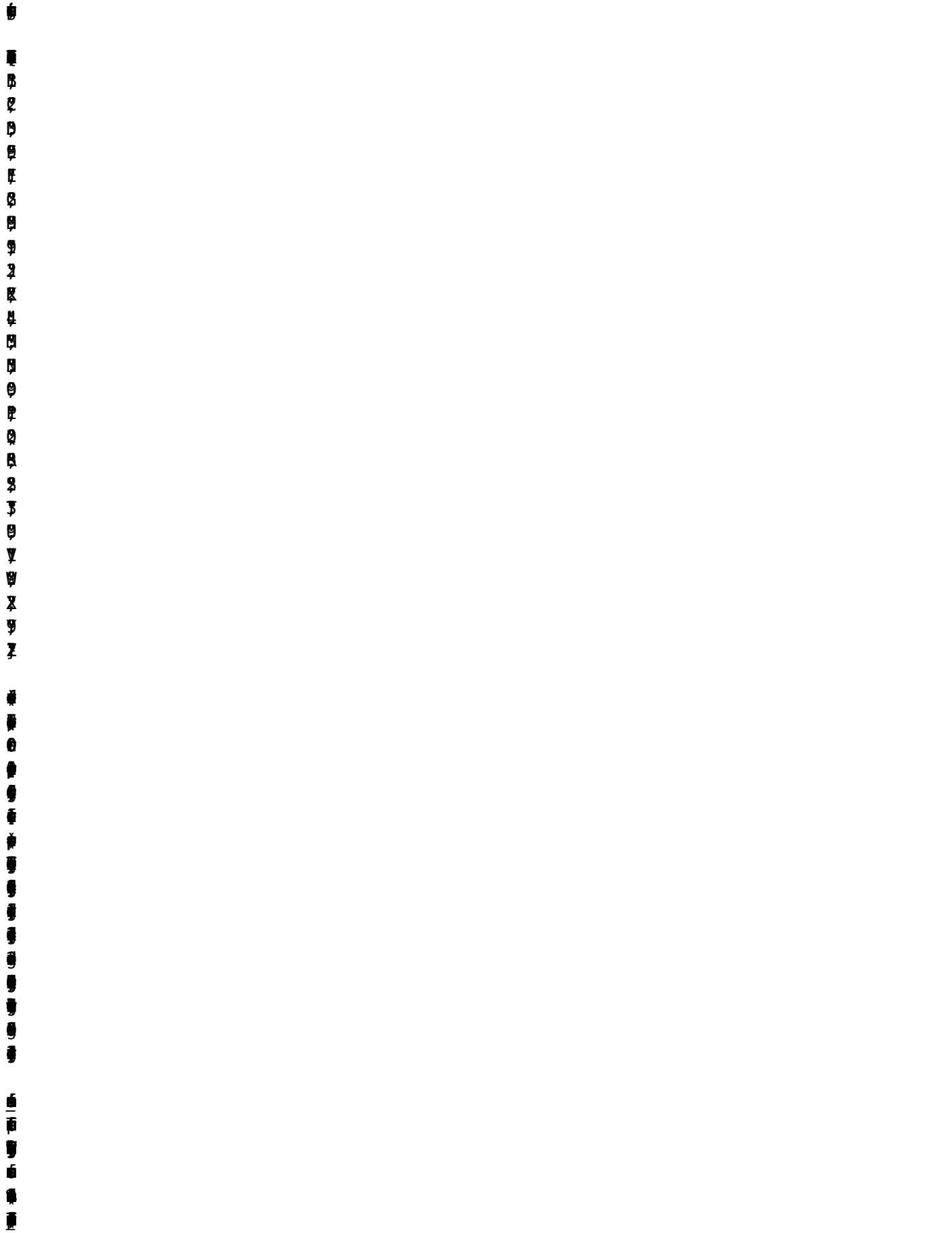
This is

```
0
```

How much did we gain by using this specific method in Quite a bit.

```
```

Example 18.3. Best Result So Far:



18.4. Optimizing Dictionary Lookups

The second step of the Soundex algorithm is to convert characters to digits in a specific pattern. What's the best way to do this?

The most obvious solution is to define a dictionary with individual characters as keys and their corresponding digits as values, and do diction

You timed already; this is how it performs:

This code is straightforward, but is it the best solution? Calling on each individual character seems inefficient; it would probably be better to

Then there's the matter of incrementally building the string. Incrementally building strings like this is horribly inefficient; internally, the Pyt

Python is good at lists, though. It can treat a string as a list of characters automatically. And lists are easy to combine into strings again, using

Here is which converts letters to digits by using and

```
#
```

Surprisingly, is not faster:

```
'
'
```

The overhead of the anonymous function kills any performance you gain by dealing with the string as a list of characters.

uses a list comprehension instead of and

Using a list comprehension in is faster than using and but still not faster than the original code (incrementally building a string in

```
'
'
```

It's time for a radically different approach. Dictionary lookups are a general purpose tool. Dictionary keys can be any length string (or many

This is

```
#
```

What the heck is going on here? creates a translation matrix between two strings: the first argument and the second argument. In this case, th

shows that is significantly faster than defining a dictionary and looping through the input and building the output incrementally:

```
'
'
```

You’re not going to get much better than that. Python has a specialized function that does exactly what you want to do; use it and move on.

Example 18.4. Best Result So Far:

18.5. Optimizing List Operations

The third step in the Soundex algorithm is eliminating consecutive duplicate digits. What’s the best way to do this?

Here’s the code we have so far, in

Here are the performance results for

The first thing to consider is whether it’s efficient to check each time through the loop. Are list indexes expensive? Would we be better off m

To answer this question, here is

```
```

does not run any faster than and may even be slightly slower (although it's not enough of a difference to say for sure):

```
```

Why isn't faster? It turns out that list indexes in Python are extremely efficient. Repeatedly accessing is no problem at all. On the other hand

Let's try something radically different. If it's possible to treat a string as a list of characters, it should be possible to use a list comprehension

However, it is possible to create a list of index numbers using the built-in function, and use those index numbers to progressively search thro

Here is

```
```

Is this faster? In a word, no.

```
```

It's possible that the techniques so far as have been "string-centric". Python can convert a string into a list of characters with a single comman

Here is which modifies a list in place to remove consecutive duplicate elements:

```
```

Is this faster than or No, in fact it's the slowest method yet:

```
```

We haven’t made any progress here at all, except to try and rule out several "clever" techniques. The fastest code we’ve seen so far was the o

Example 18.5. Best Result So Far:

18.6. Optimizing String Manipulation

The final step of the Soundex algorithm is padding short results with zeros, and truncating long results. What is the best way to do this?

This is what we have so far, taken from

These are the results for

```
```

The first thing to consider is replacing that regular expression with a loop. This code is from [illegible]

```
[illegible]
[illegible]
[illegible]
[illegible]
```

Is [illegible]aster? Yes it is:

```
[illegible]
[illegible]
[illegible]
[illegible]
```

But wait a minute. A loop to remove characters from a string? We can use a simple string method for that. Here's [illegible]

```
[illegible]
```

Is [illegible]aster? That's an interesting question. It depends on the input:

```
[illegible]
[illegible]
[illegible]
[illegible]
```

The string method in [illegible]s faster than the loop for most names, but it's actually slightly slower than [illegible]n the trivial case (of a very short name). Pe

Last but not least, let's examine the final two steps of the algorithm: padding short results with zeros, and truncating long results to four chara

```
[illegible]
[illegible]
```

Why do we need a [illegible]oop to pad out the result? We know in advance that we're going to truncate the result to four characters, and we know tha

How much speed do we gain in [illegible]y dropping the [illegible]oop? It's significant:

```
[illegible]
[illegible]
[illegible]
[illegible]
```

Finally, there is still one more thing you can do to these three lines of code to make them faster: you can combine them into one line. Take a l

```
[illegible]
```

Putting all this code on one line in [illegible]s barely faster than [illegible]



It is also significantly less readable, and for not much performance gain. Is that worth it? I hope you have good comments. Performance isn't

18.7. Summary

This chapter has illustrated several important aspects of performance tuning in Python, and performance tuning in general.

- If you need to choose between regular expressions and writing a loop, choose regular expressions. The regular expression engine is
- If you need to choose between regular expressions and string methods, choose string methods. Both are compiled in C, so choose th
- General-purpose dictionary lookups are fast, but specialtiy functions such as [illegible]nd string methods such as [illegible]re faster. If Python has a
- Don't be too clever. Sometimes the most obvious algorithm is also the fastest.·
- Don't sweat it too much. Performance isn't everything.·

I can't emphasize that last point strongly enough. Over the course of this chapter, you made this function three times faster and saved 20 seco

Appendix A. Further reading

Chapter 1. Installing Python

Chapter 2. Your First Python Program

2.3. Documenting Functions

PEP 257 (http://www.python.org/peps/pep-0257.html) defines docstring conventions.·

Python Style Guide (http://www.python.org/doc/essays/styleguide.html) discusses how to write a good docstring

Python Tutorial (http://www.python.org/doc/current/tut/tut.html) discusses conventions for spacing in docstrings (http://www.pytho

2.4.2. What's an Object?

Python Reference Manual (http://www.python.org/doc/current/ref/) explains exactly what it means to say that everything in

eff-bot (http://www.effbot.org/guides/) summarizes Python objects (http://www.effbot.org/guides/python-objects.htm).·

2.5. Indenting Code

Python Reference Manual (http://www.python.org/doc/current/ref/) discusses cross-platform indentation issues and shows

Python Style Guide (http://www.python.org/doc/essays/styleguide.html) discusses good indentation style.·

2.6. Testing Modules

Python Reference Manual (http://www.python.org/doc/current/ref/) discusses the low-level details of importing modules (h

Chapter 3. Native Datatypes

3.1.3. Deleting Items From Dictionaries

How to Think Like a Computer Scientist (http://www.ibiblio.org/obp/thinkCSpy/) teaches about dictionaries and shows how

Python Knowledge Base (http://www.faqts.com/knowledge-base/index.phtml/fid/199/) has a lot of example code using dict

Python Cookbook (http://www.activestate.com/ASPN/Python/Cookbook/) discusses how to sort the values of a dictionary

Python Library Reference (http://www.python.org/doc/current/lib/) summarizes all the dictionary methods (http://www.pyt

3.2.5. Using List Operators

How to Think Like a Computer Scientist (http://www.ibiblio.org/obp/thinkCSpy/) teaches about lists and makes an importa

Python Tutorial (http://www.python.org/doc/current/tut/tut.html) shows how to use lists as stacks and queues (http://www.p

Python Knowledge Base (http://www.faqts.com/knowledge-base/index.phtml/fid/199/) answers common questions about li

Python Library Reference (http://www.python.org/doc/current/lib/) summarizes all the list methods (http://www.python.org

3.3. Introducing Tuples

How to Think Like a Computer Scientist (http://www.ibiblio.org/obp/thinkCSpy/) teaches about tuples and shows how to co

Python Knowledge Base (http://www.faqts.com/knowledge-base/index.phtml/fid/199/) shows how to sort a tuple (http://ww

Python Tutorial (http://www.python.org/doc/current/tut/tut.html) shows how to define a tuple with one element (http://www

3.4.2. Assigning Multiple Values at Once

Python Reference Manual (http://www.python.org/doc/current/ref/) shows examples of when you can skip the line continua

How to Think Like a Computer Scientist (http://www.ibiblio.org/obp/thinkCSpy/) shows how to use multi-variable assignm

3.5. Formatting Strings

Python Library Reference (http://www.python.org/doc/current/lib/) summarizes all the string formatting format characters

Effective AWK Programming (http://www-gnats.gnu.org:8080/cgi-bin/info2www?(gawk)Top) discusses all the format char

3.6. Mapping Lists

Python Tutorial (http://www.python.org/doc/current/tut/tut.html) discusses another way to map lists using the built-in functi

Python Tutorial (http://www.python.org/doc/current/tut/tut.html) shows how to do nested list comprehensions (http://www

3.7. Joining Lists and Splitting Strings

Python Knowledge Base (http://www.faqts.com/knowledge-base/index.phtml/fid/199/) answers common questions about s

Python Library Reference (http://www.python.org/doc/current/lib/) summarizes all the string methods (http://www.python.

Python Library Reference (http://www.python.org/doc/current/lib/) documents the string module (http://www.python.org/doc/cu

The Whole Python FAQ (http://www.python.org/doc/FAQ.html) explains why join is a string method (http://www.python.org/d

Chapter 4. The Power Of Introspection

4.2. Using Optional and Named Arguments

Python Tutorial (http://www.python.org/doc/current/tut/tut.html) discusses exactly when and how default arguments are ev

4.3.3. Built-In Functions

Python Library Reference (http://www.python.org/doc/current/lib/) documents all the built-in functions (http://www.python

4.5. Filtering Lists

Python Tutorial (http://www.python.org/doc/current/tut/tut.html) discusses another way to filter lists using the built-in fun

4.6.1. Using the and-or Trick

Python Cookbook (http://www.activestate.com/ASPN/Python/Cookbook/) discusses alternatives to the and-or trick (http://www.a

4.7.1. Real-World lambda Functions

Python Knowledge Base (http://www.faqts.com/knowledge-base/index.phtml/fid/199/) discusses using apply to call functions in

Python Tutorial (http://www.python.org/doc/current/tut/tut.html) shows how to access outside variables from inside a funct

The Whole Python FAQ (http://www.python.org/doc/FAQ.html) has examples of obfuscated one-liners using apply (http://www.

Chapter 5. Objects and Object-Orientation

5.2. Importing Modules Using from module import

eff-bot (http://www.effbot.org/guides/) has more to say on from module import vs. import module (http://www.effbot.org/guides/import-confusion.htm).

Python Tutorial (http://www.python.org/doc/current/tut/tut.html) discusses advanced import techniques, including __import__ (http://w

5.3.2. Knowing When to Use self and __init__

Learning to Program (http://www.freenetpages.co.uk/hp/alan.gauld/) has a gentler introduction to classes (http://www.free

How to Think Like a Computer Scientist (http://www.ibiblio.org/obp/thinkCSpy/) shows how to use classes to model comp

Python Tutorial (http://www.python.org/doc/current/tut/tut.html) has an in-depth look at classes, namespaces, and inheritan

Python Knowledge Base (http://www.faqts.com/knowledge-base/index.phtml/fid/199/) answers common questions about c

5.4.1. Garbage Collection

Python Library Reference (http://www.python.org/doc/current/lib/) summarizes built-in attributes like __dict__ (http://www.python

Python Library Reference (http://www.python.org/doc/current/lib/) documents the gc module (http://www.python.org/doc/cu

5.5. Exploring UserDict: A Wrapper Class

Python Library Reference (http://www.python.org/doc/current/lib/) documents the UserDict module (http://www.python.org/doc/cu

5.7. Advanced Special Class Methods

Python Reference Manual (http://www.python.org/doc/current/ref/) documents all the special class methods (http://www.py

5.9. Private Functions

Python Tutorial (http://www.python.org/doc/current/tut/tut.html) discusses the inner workings of private variables (http://w

Chapter 6. Exceptions and File Handling

6.1.1. Using Exceptions For Other Purposes

Python Tutorial (http://www.python.org/doc/current/tut/tut.html) discusses defining and raising your own exceptions, and h

Python Library Reference (http://www.python.org/doc/current/lib/) summarizes all the built-in exceptions (http://www.pyth

Python Library Reference (http://www.python.org/doc/current/lib/) documents the getpass (http://www.python.org/doc/cur

Python Library Reference (http://www.python.org/doc/current/lib/) documents the traceback module (http://www.python.org/doc/cu

Python Reference Manual (http://www.python.org/doc/current/ref/) discusses the inner workings of the try block (http://www

6.2.4. Writing to Files

Python Tutorial (http://www.python.org/doc/current/tut/tut.html) discusses reading and writing files, including how to read

eff-bot (http://www.effbot.org/guides/) discusses efficiency and performance of various ways of reading a file (http://www.

Python Knowledge Base (http://www.faqts.com/knowledge-base/index.phtml/fid/199/) answers common questions about f

Python Library Reference (http://www.python.org/doc/current/lib/) summarizes all the file object methods (http://www.pyt

6.4. Using sys.modules

Python Tutorial (http://www.python.org/doc/current/tut/tut.html) discusses exactly when and how default arguments are ev

Python Library Reference (http://www.python.org/doc/current/lib/) documents the sys (http://www.python.org/doc/current/lib/

6.5. Working with Directories

Python Knowledge Base (http://www.faqts.com/knowledge-base/index.phtml/fid/199/) answers questions about the os modul

Python Library Reference (http://www.python.org/doc/current/lib/) documents the os (http://www.python.org/doc/current/lib/

Chapter 7. Regular Expressions

7.6. Case study: Parsing Phone Numbers

Regular Expression HOWTO (http://py-howto.sourceforge.net/regex/regex.html) teaches about regular expressions and ho

Python Library Reference (http://www.python.org/doc/current/lib/) summarizes the re module (http://www.python.org/doc/c

Chapter 8. HTML Processing

8.4. Introducing BaseHTMLProcessor.py

W3C (http://www.w3.org/) discusses character and entity references (http://www.w3.org/TR/REC-html40/charset.html#en

Python Library Reference (http://www.python.org/doc/current/lib/) confirms your suspicions that the sgmllib module (http://www.

8.9. Putting it all together

You thought I was kidding about the server-side scripting idea. So did I, until I found this web-based dialectizer (http://rink

Chapter 9. XML Processing

9.4. Unicode

Unicode.org (http://www.unicode.org/) is the home page of the unicode standard, including a brief technical introduction (

Unicode Tutorial (http://www.reportlab.com/i18n/python_unicode_tutorial.html) has some more examples of how to use P

PEP 263 (http://www.python.org/peps/pep-0263.html) goes into more detail about how and when to define a character enc

Chapter 10. Scripts and Streams

Chapter 11. HTTP Web Services

11.1. Diving in

Paul Prescod believes that pure HTTP web services are the future of the Internet (http://webservices.xml.com/pub/a/ws/200

Chapter 12. SOAP Web Services

12.1. Diving In

http://www.xmethods.net/ is a repository of public access SOAP web services.

The SOAP specification (http://www.w3.org/TR/soap/) is surprisingly readable, if you like that sort of thing.

12.8. Troubleshooting SOAP Web Services

New developments for SOAPpy (http://www-106.ibm.com/developerworks/webservices/library/ws-pyth17.html) steps thr

Chapter 13. Unit Testing

13.1. Introduction to Roman numerals

This site (http://www.wilkiecollins.demon.co.uk/roman/front.htm) has more on Roman numerals, including a fascinating h

13.3. Introducing romantest.py

The PyUnit home page (http://pyunit.sourceforge.net/) has an in-depth discussion of using the framework (http://pyunit.so

The PyUnit FAQ (http://pyunit.sourceforge.net/pyunit.html) explains why test cases are stored separately (http://pyunit.sou

Python Library Reference (http://www.python.org/doc/current/lib/) summarizes the unittest (http://www.python.org/doc/current/lib

ExtremeProgramming.org (http://www.extremeprogramming.org/) discusses why you should write unit tests (http://www.e

The Portland Pattern Repository (http://www.c2.com/cgi/wiki) has an ongoing discussion of unit tests (http://www.c2.com/

Chapter 14. Test-First Programming

Chapter 15. Refactoring

15.5. Summary

XProgramming.com (http://www.xprogramming.com/) has links to download unit testing frameworks (http://www.xprogra

Chapter 16. Functional Programming

Chapter 17. Dynamic functions

17.7. plural.py, stage 6

PEP 255 (http://www.python.org/peps/pep-0255.html) defines generators.·

Python Cookbook (http://www.activestate.com/ASPN/Python/Cookbook/) has many more examples of generators (http://w

Chapter 18. Performance Tuning

18.1. Diving in

Soundexing and Genealogy (http://www.avotaynu.com/soundex.html) gives a chronology of the evolution of the Soundex a

Appendix B. A 5-minute review

Chapter 1. Installing Python

1.1. Which Python is right for you?

The first thing you need to do with Python is install it. Or do you?

1.2. Python on Windows

On Windows, you have a couple choices for installing Python.

1.3. Python on Mac OS X

On Mac OS X, you have two choices for installing Python: install it, or don't install it. You probably want to install it.

1.4. Python on Mac OS 9

Mac OS 9 does not come with any version of Python, but installation is very simple, and there is only one choice.

1.5. Python on RedHat Linux

Download the latest Python RPM by going to http://www.python.org/ftp/python/ and selecting the highest version number

1.6. Python on Debian GNU/Linux

If you are lucky enough to be running Debian GNU/Linux, you install Python through the **apt** command.

1.7. Python Installation from Source

If you prefer to build from source, you can download the Python source code from http://www.python.org/ftp/python/. Sele

1.8. The Interactive Shell

Now that you have Python installed, what's this interactive shell thing you're running?

1.9. Summary

You should now have a version of Python installed that works for you.

Chapter 2. Your First Python Program

2.1. Diving in

Here is a complete, working Python program.

2.2. Declaring Functions

Python has functions like most other languages, but it does not have separate header files like C++ or [illegible]ections like Pascal.

2.3. Documenting Functions

You can document a Python function by giving it a [illegible]

2.4. Everything Is an Object

A function, like everything else in Python, is an object.

2.5. Indenting Code

Python functions have no explicit b or e and no curly braces to mark where the function code starts and stops. The only delin

2.6. Testing Modules

Python modules are objects and have several useful attributes. You can use this to easily test your modules as you write the

Chapter 3. Native Datatypes

3.1. Introducing Dictionaries

One of Python's built-in datatypes is the dictionary, which defines one-to-one relationships between keys and values.

3.2. Introducing Lists

Lists are Python's workhorse datatype. If your only experience with lists is arrays in Visual Basic or (God forbid) the datas

3.3. Introducing Tuples

A tuple is an immutable list. A tuple can not be changed in any way once it is created.

3.4. Declaring variables

Python has local and global variables like most other languages, but it has no explicit variable declarations. Variables sprin

3.5. Formatting Strings

Python supports formatting values into strings. Although this can include very complicated expressions, the most basic usa

3.6. Mapping Lists

One of the most powerful features of Python is the list comprehension, which provides a compact way of mapping a list int

3.7. Joining Lists and Splitting Strings

You have a list of key-value pairs in the form k and you want to join them into a single string. To join any list of strings into

3.8. Summary

The p program and its output should now make perfect sense.

Chapter 4. The Power Of Introspection

4.1. Diving In

Here is a complete, working Python program. You should understand a good deal about it just by looking at it. The number

4.2. Using Optional and Named Arguments

Python allows function arguments to have default values; if the function is called without the argument, the argument gets

4.3. Using type, str, dir, and Other Built-In Functions

Python has a small set of extremely useful built-in functions. All other functions are partitioned off into modules. This was

4.4. Getting Object References With getattr

You already know that Python functions are objects. What you don’t know is that you can get a reference to a function witl

4.5. Filtering Lists

As you know, Python has powerful capabilities for mapping lists into other lists, via list comprehensions (Section 3.6, “Ma

4.6. The Peculiar Nature of and and or

In Python, and perform boolean logic as you would expect, but they do not return boolean values; instead, they return one

4.7. Using lambda Functions

Python supports an interesting syntax that lets you define one-line mini-functions on the fly. Borrowed from Lisp, these so-

4.8. Putting It All Together

The last line of code, the only one you haven’t deconstructed yet, is the one that does all the work. But by now the work is

4.9. Summary

The program and its output should now make perfect sense.

Chapter 5. Objects and Object-Orientation

5.1. Diving In

Here is a complete, working Python program. Read the of the module, the classes, and the functions to get an overview of

5.2. Importing Modules Using from module import

Python has two ways of importing modules. Both are useful, and you should know when to use each. One way, you’ve alr

5.3. Defining Classes

Python is fully object-oriented: you can define your own classes, inherit from your own or built-in classes, and instantiate t

5.4. Instantiating Classes

Instantiating classes in Python is straightforward. To instantiate a class, simply call the class as if it were a function, passin

5.5. Exploring UserDict: A Wrapper Class

As you’ve seen, is a class that acts like a dictionary. To explore this further, let’s look at the class in the module, which is

5.6. Special Class Methods

In addition to normal class methods, there are a number of special methods that Python classes can define. Instead of being

5.7. Advanced Special Class Methods

Python has more special methods than just and Some of them let you emulate functionality that you may not even know a

5.8. Introducing Class Attributes

You already know about data attributes, which are variables owned by a specific instance of a class. Python also supports c

5.9. Private Functions

Unlike in most languages, whether a Python function, method, or attribute is private or public is determined entirely by its

5.10. Summary

That's it for the hard-core object trickery. You'll see a real-world application of special class methods in Chapter 12, which

Chapter 6. Exceptions and File Handling

6.1. Handling Exceptions

Like many other programming languages, Python has exception handling via blocks.

6.2. Working with File Objects

Python has a built-in function, for opening a file on disk. returns a file object, which has methods and attributes for gettin

6.3. Iterating with for Loops

Like most other languages, Python has loops. The only reason you haven't seen them until now is that Python is good at so

6.4. Using sys.modules

Modules, like everything else in Python, are objects. Once imported, you can always get a reference to a module through th

6.5. Working with Directories

The module has several functions for manipulating files and directories. Here, we're looking at handling pathnames and lis

6.6. Putting It All Together

Once again, all the dominoes are in place. You've seen how each line of code works. Now let's step back and see how it al

6.7. Summary

The program introduced in Chapter 5 should now make perfect sense.

Chapter 7. Regular Expressions

7.1. Diving In

If what you're trying to do can be accomplished with string functions, you should use them. They're fast and simple and ea

7.2. Case Study: Street Addresses

This series of examples was inspired by a real-life problem I had in my day job several years ago, when I needed to scrub a

7.3. Case Study: Roman Numerals

You've most likely seen Roman numerals, even if you didn't recognize them. You may have seen them in copyrights of ol

7.4. Using the {n,m} Syntax

In the previous section, you were dealing with a pattern where the same character could be repeated up to three times. Ther

7.5. Verbose Regular Expressions

So far you've just been dealing with what I'll call "compact" regular expressions. As you've seen, they are difficult to read

7.6. Case study: Parsing Phone Numbers

So far you've concentrated on matching whole patterns. Either the pattern matches, or it doesn't. But regular expressions a

7.7. Summary

This is just the tiniest tip of the iceberg of what regular expressions can do. In other words, even though you're completely

Chapter 8. HTML Processing

8.1. Diving in

I often see questions on comp.lang.python (http://groups.google.com/groups?group=comp.lang.python) like "How can I lis

8.2. Introducing sgmllib.py

HTML processing is broken into three steps: breaking down the HTML into its constituent pieces, fiddling with the pieces,

8.3. Extracting data from HTML documents

To extract data from HTML documents, subclass the class and define methods for each tag or entity you want to capture.

8.4. Introducing BaseHTMLProcessor.py

doesn't produce anything by itself. It parses and parses and parses, and it calls a method for each interesting thing it finds,

8.5. locals and globals

Let's digress from HTML processing for a minute and talk about how Python handles variables. Python has two built-in fu

8.6. Dictionary-based string formatting

There is an alternative form of string formatting that uses dictionaries instead of tuples of values.

8.7. Quoting attribute values

(This is generally precipitated by a project (http://groups.google.com/groups?group=comp.lang.python) in "I have a project

8.8. Introducing dialect.py

is a simple (and silly) descendant of It runs blocks of text through a series of substitutions, but it makes sure that anything

8.9. Putting it all together

It’s time to put everything you’ve learned so far to good use. I hope you were paying attention.

8.10. Summary

Python provides you with a powerful tool, to manipulate HTML by turning its structure into an object model. You can use

Chapter 9. XML Processing

9.1. Diving in

There are two basic ways to work with XML. One is called SAX ("Simple API for XML"), and it works by reading the XM

9.2. Packages

Actually parsing an XML document is very simple: one line of code. However, before you get to that line of code, you nee

9.3. Parsing XML

As I was saying, actually parsing an XML document is very simple: one line of code. Where you go from there is up to you

9.4. Unicode

Unicode is a system to represent characters from all the world’s different languages. When Python parses an XML docume

9.5. Searching for elements

Traversing XML documents by stepping through each node can be tedious. If you’re looking for something in particular, b

9.6. Accessing element attributes

XML elements can have one or more attributes, and it is incredibly simple to access them once you have parsed an XML d

9.7. Segue

OK, that’s it for the hard-core XML stuff. The next chapter will continue to use these same example programs, but focus on

Chapter 10. Scripts and Streams

10.1. Abstracting input sources

One of Python’s greatest strengths is its dynamic binding, and one powerful use of dynamic binding is the *file-like object*.

10.2. Standard input, output, and error

UNIX users are already familiar with the concept of standard input, standard output, and standard error. This section is for

10.3. Caching node lookups

employs several tricks which may or may not be useful to you in your XML processing. The first one takes advantage of th

10.4. Finding direct children of a node

Another useful techique when parsing XML documents is finding all the direct child elements of a particular element. For i

10.5. Creating separate handlers by node type

The third useful XML processing tip involves separating your code into logical functions, based on node types and element

10.6. Handling command-line arguments

Python fully supports creating programs that can be run on the command line, complete with command-line arguments and

10.7. Putting it all together

You've covered a lot of ground. Let's step back and see how all the pieces fit together.

10.8. Summary

Python comes with powerful libraries for parsing and manipulating XML documents. The takes an XML file and parses it

Chapter 11. HTTP Web Services

11.1. Diving in

You've learned about HTML processing and XML processing, and along the way you saw how to download a web page ar

11.2. How not to fetch data over HTTP

Let's say you want to download a resource over HTTP, such as a syndicated Atom feed. But you don't just want to downlo

11.3. Features of HTTP

There are five important features of HTTP which you should support.

11.4. Debugging HTTP web services

First, let's turn on the debugging features of Python's HTTP library and see what's being sent over the wire. This will be us

11.5. Setting the User-Agent

The first step to improving your HTTP web services client is to identify yourself properly with a To do that, you need to

11.6. Handling Last-Modified and ETag

Now that you know how to add custom HTTP headers to your web service requests, let's look at adding support for and

11.7. Handling redirects

You can support permanent and temporary redirects using a different kind of custom URL handler.

11.8. Handling compressed data

The last important HTTP feature you want to support is compression. Many web services have the ability to send data com

11.9. Putting it all together

You've seen all the pieces for building an intelligent HTTP web services client. Now let's see how they all fit together.

11.10. Summary

The and its functions should now make perfect sense.

Chapter 12. SOAP Web Services

12.1. Diving In

You use Google, right? It’s a popular search engine. Have you ever wished you could programmatically access Google sea

12.2. Installing the SOAP Libraries

Unlike the other code in this book, this chapter relies on libraries that do not come pre-installed with Python.

12.3. First Steps with SOAP

The heart of SOAP is the ability to call remote functions. There are a number of public access SOAP servers that provide s

12.4. Debugging SOAP Web Services

The SOAP libraries provide an easy way to see what’s going on behind the scenes.

12.5. Introducing WSDL

The class proxies local method calls and transparently turns then into invocations of remote SOAP methods. As you’ve see

12.6. Introspecting SOAP Web Services with WSDL

Like many things in the web services arena, WSDL has a long and checkered history, full of political strife and intrigue. I v

12.7. Searching Google

Let’s finally turn to the sample code that you saw that the beginning of this chapter, which does something more useful and

12.8. Troubleshooting SOAP Web Services

Of course, the world of SOAP web services is not all happiness and light. Sometimes things go wrong.

12.9. Summary

SOAP web services are very complicated. The specification is very ambitious and tries to cover many different use cases fo

Chapter 13. Unit Testing

13.1. Introduction to Roman numerals

In previous chapters, you "dived in" by immediately looking at code and trying to understand it as quickly as possible. Nov

13.2. Diving in

Now that you’ve completely defined the behavior you expect from your conversion functions, you’re going to do somethin

13.3. Introducing romantest.py

This is the complete test suite for your Roman numeral conversion functions, which are yet to be written but will eventuall

13.4. Testing for success

The most fundamental part of unit testing is constructing individual test cases. A test case answers a single question about t

13.5. Testing for failure

It is not enough to test that functions succeed when given good input; you must also test that they fail when given bad inpu

13.6. Testing for sanity

Often, you will find that a unit of code contains a set of reciprocal functions, usually in the form of conversion functions wh

Chapter 14. Test-First Programming

14.1. roman.py, stage 1

Now that the unit tests are complete, it’s time to start writing the code that the test cases are attempting to test. You’re goin

14.2. roman.py, stage 2

Now that you have the framework of the module laid out, it’s time to start writing code and passing test cases.

14.3. roman.py, stage 3

Now that behaves correctly with good input (integers from 1 to 9), it’s time to make it behave correctly with bad input (every

14.4. roman.py, stage 4

Now that is done, it’s time to start coding . Thanks to the rich data structure that maps individual Roman numerals to integ

14.5. roman.py, stage 5

Now that works properly with good input, it’s time to fit in the last piece of the puzzle: making it work properly with bad i

Chapter 15. Refactoring

15.1. Handling bugs

Despite your best efforts to write comprehensive unit tests, bugs happen. What do I mean by "bug"? A bug is a test case yo

15.2. Handling changing requirements

Despite your best efforts to pin your customers to the ground and extract exact requirements from them on pain of horrible

15.3. Refactoring

The best thing about comprehensive unit testing is not the feeling you get when all your test cases finally pass, or even the

15.4. Postscript

A clever reader read the previous section and took it to the next level. The biggest headache (and performance drain) in the

15.5. Summary

Unit testing is a powerful concept which, if properly implemented, can both reduce maintenance costs and increase flexibil

Chapter 16. Functional Programming

16.1. Diving in

In Chapter 13, *Unit Testing*, you learned about the philosophy of unit testing. In Chapter 14, *Test-First Programming*, you

16.2. Finding the path

When running Python scripts from the command line, it is sometimes useful to know where the currently running script is l

16.3. Filtering lists revisited

You're already familiar with using list comprehensions to filter lists. There is another way to accomplish this same thing, w

16.4. Mapping lists revisited

You're already familiar with using list comprehensions to map one list into another. There is another way to accomplish th

16.5. Data-centric programming

By now you're probably scratching your head wondering why this is better than using loops and straight function calls. An

16.6. Dynamically importing modules

OK, enough philosophizing. Let's talk about dynamically importing modules.

16.7. Putting it all together

You've learned enough now to deconstruct the first seven lines of this chapter's code sample: reading a directory and impo

16.8. Summary

The program and its output should now make perfect sense.

Chapter 17. Dynamic functions

17.1. Diving in

I want to talk about plural nouns. Also, functions that return other functions, advanced regular expressions, and generators.

17.2. plural.py, stage 1

So you're looking at words, which at least in English are strings of characters. And you have rules that say you need to find

17.3. plural.py, stage 2

Now you're going to add a level of abstraction. You started by defining a list of rules: if this, then do that, otherwise go to t

17.4. plural.py, stage 3

Defining separate named functions for each match and apply rule isn’t really necessary. You never call them directly; you c

17.5. plural.py, stage 4

Let’s factor out the duplication in the code so that defining new rules can be easier.

17.6. plural.py, stage 5

You’ve factored out all the duplicate code and added enough abstractions so that the pluralization rules are defined in a list

17.7. plural.py, stage 6

Now you’re ready to talk about generators.

17.8. Summary

You talked about several different advanced techniques in this chapter. Not all of them are appropriate for every situation.

Chapter 18. Performance Tuning

18.1. Diving in

There are so many pitfalls involved in optimizing your code, it’s hard to know where to start.

18.2. Using the timeit Module

The most important thing you need to know about optimizing Python code is that you shouldn’t write your own timing func

18.3. Optimizing Regular Expressions

The first thing the Soundex function checks is whether the input is a non-empty string of letters. What’s the best way to do

18.4. Optimizing Dictionary Lookups

The second step of the Soundex algorithm is to convert characters to digits in a specific pattern. What’s the best way to do

18.5. Optimizing List Operations

The third step in the Soundex algorithm is eliminating consecutive duplicate digits. What’s the best way to do this?

18.6. Optimizing String Manipulation

The final step of the Soundex algorithm is padding short results with zeros, and truncating long results. What is the best wa

18.7. Summary

This chapter has illustrated several important aspects of performance tuning in Python, and performance tuning in general.

Appendix C. Tips and tricks

Chapter 1. Installing Python

Chapter 2. Your First Python Program

2.1. Diving in



In the ActivePython IDE on Windows, you can run the Python program you're editing by choosing File->Run... (**Ctrl-R**). Output is



In the Python IDE on Mac OS, you can run a Python program with Python->Run window... (**Cmd-R**), but there is an important opti



On UNIX-compatible systems (including Mac OS X), you can run a Python program from the command line:

2.2. Declaring Functions



In Visual Basic, functions (that return a value) start with and subroutines (that do not return a value) start with There are no subro



In Java, C++, and other statically-typed languages, you must specify the datatype of the function return value and each function arg

2.3. Documenting Functions



Triple quotes are also an easy way to define a string with both single and double quotes, like in Perl.



Many Python IDEs use the to provide context-sensitive documentation, so that when you type a function name, its appears as a too

2.4. Everything Is an Object



In Python is like in Perl. Once you a Python module, you access its functions with once you a Perl module, you access its functi

2.5. Indenting Code



Python uses carriage returns to separate statements and a colon and indentation to separate code blocks. C++ and Java use semicolo

2.6. Testing Modules



Like C, Python uses for comparison and for assignment. Unlike C, Python does not support in-line assignment, so there's no chanc



On MacPython, there is an additional step to make the trick work. Pop up the module's options menu by clicking the black triangle

Chapter 3. Native Datatypes

3.1. Introducing Dictionaries



A dictionary in Python is like a hash in Perl. In Perl, variables that store hashes always start with a character. In Python, variables c

A dictionary in Python is like an instance of the Hashtable class in Java.

A dictionary in Python is like an instance of the Scripting.Dictionary object in Visual Basic.

3.1.2. Modifying Dictionaries

Dictionaries have no concept of order among elements. It is incorrect to say that the elements are "out of order"; they are simply unc

3.2. Introducing Lists

A list in Python is like an array in Perl. In Perl, variables that store arrays always start with the @ character; in Python, variables can b

A list in Python is much more than an array in Java (although it can be used as one if that's really all you want out of life). A better

3.2.3. Searching Lists

Before version 2.2.1, Python had no separate boolean datatype. To compensate for this, Python accepted almost anything in a boole

- 0 is false; all other numbers are true.·
- An empty string ('' or "") is false, all other strings are true.·
- An empty list ([]) is false; all other lists are true.·
- An empty tuple (()) is false; all other tuples are true.·
- An empty dictionary ({}) is false; all other dictionaries are true.·

These rules still apply in Python 2.2.1 and beyond, but now you can also use an actual boolean, which has a value of True or False. Note the c

3.3. Introducing Tuples

Tuples can be converted into lists, and vice-versa. The built-in tuple function takes a list and returns a tuple with the same elements, and

3.4. Declaring variables

When a command is split among several lines with the line-continuation marker ("\"), the continued lines can be indented in any mar

3.5. Formatting Strings

String formatting in Python uses the same syntax as the sprintf function in C.

3.7. Joining Lists and Splitting Strings

join works only on lists of strings; it does not do any type coercion. Joining a list that has one or more non-string elements will raise an

split is a useful technique when you want to search a string for a substring and then work with everything before the substring (which en

Chapter 4. The Power Of Introspection

4.2. Using Optional and Named Arguments

The only thing you need to do to call a function is specify a value (somehow) for each required argument; the manner and order in v

4.3.3. Built-In Functions

Python comes with excellent reference manuals, which you should peruse thoroughly to learn all the modules Python has to offer. E

4.7. Using lambda Functions

lunctions are a matter of style. Using them is never required; anywhere you could use them, you could define a separate normal fun

4.8. Putting It All Together

In SQL, you must use nstead of o compare a null value. In Python, you can use either or but s faster.

Chapter 5. Objects and Object-Orientation

5.2. Importing Modules Using from module import

n Python is like n Perl; n Python is like n Perl.

n Python is like n Java; n Python is like n Java.

Use paringly, because it makes it difficult to determine where a particular function or attribute came from, and that makes debuggi

5.3. Defining Classes

The statement in Python is like an empty set of braces (in Java or C.

In Python, the ancestor of a class is simply listed in parentheses immediately after the class name. There is no special keyword like

5.3.1. Initializing and Coding Classes

By convention, the first argument of any Python class method (the reference to the current instance) is called This argument fills th

5.3.2. Knowing When to Use self and __init__

methods are optional, but when you define one, you must remember to explicitly call the ancestor's method (if it defines one). This

5.4. Instantiating Classes

In Python, simply call a class as if it were a function to create a new instance of the class. There is no explicit operator like C++ or

5.5. Exploring UserDict: A Wrapper Class

In the ActivePython IDE on Windows, you can quickly open any module in your library path by selecting File->Locate... (**Ctrl-L**).

Java and Powerbuilder support function overloading by argument list, *i.e.* one class can have multiple methods with the same name

Guido, the original author of Python, explains method overriding this way: "Derived classes may override methods of their base cla

Always assign an initial value to all of an instance's data attributes in the method. It will save you hours of debugging later, trackin

In versions of Python prior to 2.2, you could not directly subclass built-in datatypes like strings, lists, and dictionaries. To compensa

5.6.1. Getting and Setting Items

When accessing data attributes within a class, you need to qualify the attribute name: When calling other methods within a class, y

5.7. Advanced Special Class Methods

In Java, you determine whether two string variables reference the same physical memory location by using This is called *object id*

While other object-oriented languages only let you define the physical model of an object ("this object has a method"), Python's sp

5.8. Introducing Class Attributes

In Java, both static variables (called class attributes in Python) and instance variables (called data attributes in Python) are defined i

There are no constants in Python. Everything can be changed if you try hard enough. This fits with one of the core principles of Pyt

5.9. Private Functions

In Python, all special methods (like and built-in attributes (like follow a standard naming convention: they both start with and en

Chapter 6. Exceptions and File Handling

6.1. Handling Exceptions

Python uses to handle exceptions and to generate them. Java and C++ use to handle exceptions, and to generate them.

6.5. Working with Directories

Whenever possible, you should use the functions in and for file, directory, and path manipulations. These modules are wrappers fo

Chapter 7. Regular Expressions

7.4. Using the {n,m} Syntax

There is no way to programmatically determine that two regular expressions are equivalent. The best you can do is write a lot of tes

Chapter 8. HTML Processing

8.2. Introducing sgmllib.py

Python 2.0 had a bug where would not recognize declarations at all (would never be called), which meant that were silently ignor

In the ActivePython IDE on Windows, you can specify command line arguments in the "Run script" dialog. Separate multiple argu

8.4. Introducing BaseHTMLProcessor.py

The HTML specification requires that all non-HTML (like client-side JavaScript) must be enclosed in HTML comments, but not all

8.5. locals and globals

Python 2.2 introduced a subtle but important change that affects the namespace search order: nested scopes. In versions of Python p

Using the and functions, you can get the value of arbitrary variables dynamically, providing the variable name as a string. This mi

8.6. Dictionary-based string formatting

Using dictionary-based string formatting with is a convenient way of making complex string formatting expressions more readable

Chapter 9. XML Processing

9.2. Packages

A package is a directory with the special file in it. The file defines the attributes and methods of the package. It doesn't need to def

9.6. Accessing element attributes

This section may be a little confusing, because of some overlapping terminology. Elements in an XML document have attributes, ar

Like a dictionary, attributes of an XML element have no ordering. Attributes may *happen to be* listed in a certain order in the origin

Chapter 10. Scripts and Streams

Chapter 11. HTTP Web Services

11.6. Handling Last-Modified and ETag

In these examples, the HTTP server has supported both and headers, but not all servers do. As a web services client, you should be

Chapter 12. SOAP Web Services

Chapter 13. Unit Testing

13.2. Diving in

is included with Python 2.1 and later. Python 2.0 users can download it from (http://pyunit.sourceforge.net/).

Chapter 14. Test-First Programming

14.3. roman.py, stage 3

The most important thing that comprehensive unit testing can tell you is when to stop coding. When all the unit tests for a function p

14.5. roman.py, stage 5

When all of your tests pass, stop coding.

Chapter 15. Refactoring

15.3. Refactoring

Whenever you are going to use a regular expression more than once, you should compile it to get a pattern object, then call the meth

Chapter 16. Functional Programming

16.2. Finding the path

The pathnames and filenames you pass to do not need to exist.

not only constructs full path names, it also normalizes them. That means that if you are in the directory, will return It normalizes

Like the other functions in the and modules, is cross-platform. Your results will look slightly different than my examples if you're

Chapter 17. Dynamic functions

Chapter 18. Performance Tuning

18.2. Using the timeit Module

You can use the module on the command line to test an existing Python program, without modifying the code. See http://docs.pyth

The module only works if you already know what piece of code you need to optimize. If you have a larger Python program and don

Appendix D. List of examples

Chapter 1. Installing Python

1.3. Python on Mac OS X
Example 1.1. Two versions of Python
1.5. Python on RedHat Linux
Example 1.2. Installing on RedHat Linux 9
1.6. Python on Debian GNU/Linux
Example 1.3. Installing on Debian GNU/Linux
1.7. Python Installation from Source
Example 1.4. Installing from source
1.8. The Interactive Shell
Example 1.5. First Steps in the Interactive Shell

Chapter 2. Your First Python Program

2.1. Diving in
Example 2.1. odbchelper.py
2.3. Documenting Functions
Example 2.2. Defining the buildConnectionString Function’s doc string
2.4. Everything Is an Object
Example 2.3. Accessing the buildConnectionString Function’s doc string
2.4.1. The Import Search Path
Example 2.4. Import Search Path
2.5. Indenting Code
Example 2.5. Indenting the buildConnectionString Function
Example 2.6. if Statements

Chapter 3. Native Datatypes

3.1.1. Defining Dictionaries
Example 3.1. Defining a Dictionary
3.1.2. Modifying Dictionaries
Example 3.2. Modifying a Dictionary
Example 3.3. Dictionary Keys Are Case-Sensitive
Example 3.4. Mixing Datatypes in a Dictionary
3.1.3. Deleting Items From Dictionaries
Example 3.5. Deleting Items from a Dictionary
3.2.1. Defining Lists
Example 3.6. Defining a List
Example 3.7. Negative List Indices
Example 3.8. Slicing a List
Example 3.9. Slicing Shorthand
3.2.2. Adding Elements to Lists
Example 3.10. Adding Elements to a List
Example 3.11. The Difference between extend and append
3.2.3. Searching Lists
Example 3.12. Searching a List
3.2.4. Deleting List Elements
Example 3.13. Removing Elements from a List
3.2.5. Using List Operators
Example 3.14. List Operators

3.3. Introducing Tuples
Example 3.15. Defining a tuple
Example 3.16. Tuples Have No Methods
3.4. Declaring variables
Example 3.17. Defining the myParams Variable
3.4.1. Referencing Variables
Example 3.18. Referencing an Unbound Variable
3.4.2. Assigning Multiple Values at Once
Example 3.19. Assigning multiple values at once
Example 3.20. Assigning Consecutive Values
3.5. Formatting Strings
Example 3.21. Introducing String Formatting
Example 3.22. String Formatting vs. Concatenating
Example 3.23. Formatting Numbers
3.6. Mapping Lists
Example 3.24. Introducing List Comprehensions
Example 3.25. The keys, values, and items Functions
Example 3.26. List Comprehensions in buildConnectionString, Step by Step
3.7. Joining Lists and Splitting Strings
Example 3.27. Output of odbchelper.py
Example 3.28. Splitting a String

Chapter 4. The Power Of Introspection

4.1. Diving In
Example 4.1. apihelper.py
Example 4.2. Sample Usage of apihelper.py
Example 4.3. Advanced Usage of apihelper.py
4.2. Using Optional and Named Arguments
Example 4.4. Valid Calls of info
4.3.1. The type Function
Example 4.5. Introducing type
4.3.2. The str Function
Example 4.6. Introducing str
Example 4.7. Introducing dir
Example 4.8. Introducing callable
4.3.3. Built-In Functions
Example 4.9. Built-in Attributes and Functions
4.4. Getting Object References With getattr
Example 4.10. Introducing getattr
4.4.1. getattr with Modules
Example 4.11. The getattr Function in apihelper.py
4.4.2. getattr As a Dispatcher
Example 4.12. Creating a Dispatcher with getattr
Example 4.13. getattr Default Values
4.5. Filtering Lists
Example 4.14. Introducing List Filtering
4.6. The Peculiar Nature of and and or
Example 4.15. Introducing and
Example 4.16. Introducing or
4.6.1. Using the and-or Trick
Example 4.17. Introducing the and-or Trick
Example 4.18. When the and-or Trick Fails

Example 4.19. Using the and-or Trick Safely
4.7. Using lambda Functions
Example 4.20. Introducing lambda Functions
4.7.1. Real-World lambda Functions
Example 4.21. split With No Arguments
4.8. Putting It All Together
Example 4.22. Getting a doc string Dynamically
Example 4.23. Why Use str on a doc string?
Example 4.24. Introducing ljust
Example 4.25. Printing a List

Chapter 5. Objects and Object-Orientation

5.1. Diving In
Example 5.1. fileinfo.py
5.2. Importing Modules Using from module import
Example 5.2. import module vs. from module import
5.3. Defining Classes
Example 5.3. The Simplest Python Class
Example 5.4. Defining the FileInfo Class
5.3.1. Initializing and Coding Classes
Example 5.5. Initializing the FileInfo Class
Example 5.6. Coding the FileInfo Class
5.4. Instantiating Classes
Example 5.7. Creating a FileInfo Instance
5.4.1. Garbage Collection
Example 5.8. Trying to Implement a Memory Leak
5.5. Exploring UserDict: A Wrapper Class
Example 5.9. Defining the UserDict Class
Example 5.10. UserDict Normal Methods
Example 5.11. Inheriting Directly from Built-In Datatype dict
5.6.1. Getting and Setting Items
Example 5.12. The __getitem__ Special Method
Example 5.13. The __setitem__ Special Method
Example 5.14. Overriding __setitem__ in MP3FileInfo
Example 5.15. Setting an MP3FileInfo's name
5.7. Advanced Special Class Methods
Example 5.16. More Special Methods in UserDict
5.8. Introducing Class Attributes
Example 5.17. Introducing Class Attributes
Example 5.18. Modifying Class Attributes
5.9. Private Functions
Example 5.19. Trying to Call a Private Method

Chapter 6. Exceptions and File Handling

6.1. Handling Exceptions
Example 6.1. Opening a Non-Existent File
6.1.1. Using Exceptions For Other Purposes
Example 6.2. Supporting Platform-Specific Functionality
6.2. Working with File Objects
Example 6.3. Opening a File
6.2.1. Reading Files

Example 6.4. Reading a File
6.2.2. Closing Files
Example 6.5. Closing a File
6.2.3. Handling I/O Errors
Example 6.6. File Objects in MP3FileInfo
6.2.4. Writing to Files
Example 6.7. Writing to Files
6.3. Iterating with for Loops
Example 6.8. Introducing the for Loop
Example 6.9. Simple Counters
Example 6.10. Iterating Through a Dictionary
Example 6.11. for Loop in MP3FileInfo
6.4. Using sys.modules
Example 6.12. Introducing sys.modules
Example 6.13. Using sys.modules
Example 6.14. The __module__ Class Attribute
Example 6.15. sys.modules in fileinfo.py
6.5. Working with Directories
Example 6.16. Constructing Pathnames
Example 6.17. Splitting Pathnames
Example 6.18. Listing Directories
Example 6.19. Listing Directories in fileinfo.py
Example 6.20. Listing Directories with glob
6.6. Putting It All Together
Example 6.21. listDirectory

Chapter 7. Regular Expressions

7.2. Case Study: Street Addresses
Example 7.1. Matching at the End of a String
Example 7.2. Matching Whole Words
7.3.1. Checking for Thousands
Example 7.3. Checking for Thousands
7.3.2. Checking for Hundreds
Example 7.4. Checking for Hundreds
7.4. Using the {n,m} Syntax
Example 7.5. The Old Way: Every Character Optional
Example 7.6. The New Way: From n o m
7.4.1. Checking for Tens and Ones
Example 7.7. Checking for Tens
Example 7.8. Validating Roman Numerals with {n,m}
7.5. Verbose Regular Expressions
Example 7.9. Regular Expressions with Inline Comments
7.6. Case study: Parsing Phone Numbers
Example 7.10. Finding Numbers
Example 7.11. Finding the Extension
Example 7.12. Handling Different Separators
Example 7.13. Handling Numbers Without Separators
Example 7.14. Handling Leading Characters
Example 7.15. Phone Number, Wherever I May Find Ye
Example 7.16. Parsing Phone Numbers (Final Version)

Chapter 8. HTML Processing

8.1. Diving in
Example 8.1. BaseHTMLProcessor.py
Example 8.2. dialect.py
Example 8.3. Output of dialect.py
8.2. Introducing sgmllib.py
Example 8.4. Sample test of sgmllib.py
8.3. Extracting data from HTML documents
Example 8.5. Introducing urllib
Example 8.6. Introducing urllister.py
Example 8.7. Using urllister.py
8.4. Introducing BaseHTMLProcessor.py
Example 8.8. Introducing BaseHTMLProcessor
Example 8.9. BaseHTMLProcessor output
8.5. locals and globals
Example 8.10. Introducing locals
Example 8.11. Introducing globals
Example 8.12. locals is read-only, globals is not
8.6. Dictionary-based string formatting
Example 8.13. Introducing dictionary-based string formatting
Example 8.14. Dictionary-based string formatting in BaseHTMLProcessor.py
Example 8.15. More dictionary-based string formatting
8.7. Quoting attribute values
Example 8.16. Quoting attribute values
8.8. Introducing dialect.py
Example 8.17. Handling specific tags
Example 8.18. SGMLParser
Example 8.19. Overriding the handle_data method
8.9. Putting it all together
Example 8.20. The translate function, part 1
Example 8.21. The translate function, part 2: curiouser and curiouser
Example 8.22. The translate function, part 3

Chapter 9. XML Processing

9.1. Diving in
Example 9.1. kgp.py
Example 9.2. toolbox.py
Example 9.3. Sample output of kgp.py
Example 9.4. Simpler output from kgp.py
9.2. Packages
Example 9.5. Loading an XML document (a sneak peek)
Example 9.6. File layout of a package
Example 9.7. Packages are modules, too
9.3. Parsing XML
Example 9.8. Loading an XML document (for real this time)
Example 9.9. Getting child nodes
Example 9.10. toxml works on any node
Example 9.11. Child nodes can be text
Example 9.12. Drilling down all the way to text
9.4. Unicode
Example 9.13. Introducing unicode
Example 9.14. Storing non-ASCII characters
Example 9.15. sitecustomize.py

Example 9.16. Effects of setting the default encoding
Example 9.17. Specifying encoding in .py files
Example 9.18. russiansample.xml
Example 9.19. Parsing russiansample.xml
9.5. Searching for elements
Example 9.20. binary.xml
Example 9.21. Introducing getElementsByTagName
Example 9.22. Every element is searchable
Example 9.23. Searching is actually recursive
9.6. Accessing element attributes
Example 9.24. Accessing element attributes
Example 9.25. Accessing individual attributes

Chapter 10. Scripts and Streams

10.1. Abstracting input sources
Example 10.1. Parsing XML from a file
Example 10.2. Parsing XML from a URL
Example 10.3. Parsing XML from a string (the easy but inflexible way)
Example 10.4. Introducing StringIO
Example 10.5. Parsing XML from a string (the file-like object way)
Example 10.6. openAnything
Example 10.7. Using openAnything in kgp.py
10.2. Standard input, output, and error
Example 10.8. Introducing stdout and stderr
Example 10.9. Redirecting output
Example 10.10. Redirecting error information
Example 10.11. Printing to stderr
Example 10.12. Chaining commands
Example 10.13. Reading from standard input in kgp.py
10.3. Caching node lookups
Example 10.14. loadGrammar
Example 10.15. Using the ref element cache
10.4. Finding direct children of a node
Example 10.16. Finding direct child elements
10.5. Creating separate handlers by node type
Example 10.17. Class names of parsed XML objects
Example 10.18. parse, a generic XML node dispatcher
Example 10.19. Functions called by the parse dispatcher
10.6. Handling command-line arguments
Example 10.20. Introducing sys.argv
Example 10.21. The contents of sys.argv
Example 10.22. Introducing getopt
Example 10.23. Handling command-line arguments in kgp.py

Chapter 11. HTTP Web Services

11.1. Diving in
Example 11.1. openanything.py
11.2. How not to fetch data over HTTP
Example 11.2. Downloading a feed the quick-and-dirty way
11.4. Debugging HTTP web services
Example 11.3. Debugging HTTP

11.5. Setting the User-Agent
Example 11.4. Introducing urllib2
Example 11.5. Adding headers with the Request
11.6. Handling Last-Modified and ETag
Example 11.6. Testing Last-Modified
Example 11.7. Defining URL handlers
Example 11.8. Using custom URL handlers
Example 11.9. Supporting ETag/If-None-Match
11.7. Handling redirects
Example 11.10. Accessing web services without a redirect handler
Example 11.11. Defining the redirect handler
Example 11.12. Using the redirect handler to detect permanent redirects
Example 11.13. Using the redirect handler to detect temporary redirects
11.8. Handling compressed data
Example 11.14. Telling the server you would like compressed data
Example 11.15. Decompressing the data
Example 11.16. Decompressing the data directly from the server
11.9. Putting it all together
Example 11.17. The openanything function
Example 11.18. The fetch function
Example 11.19. Using openanything.py

Chapter 12. SOAP Web Services

12.1. Diving In
Example 12.1. search.py
Example 12.2. Sample Usage of search.py
12.2.1. Installing PyXML
Example 12.3. Verifying PyXML Installation
12.2.2. Installing fpconst
Example 12.4. Verifying fpconst Installation
12.2.3. Installing SOAPpy
Example 12.5. Verifying SOAPpy Installation
12.3. First Steps with SOAP
Example 12.6. Getting the Current Temperature
12.4. Debugging SOAP Web Services
Example 12.7. Debugging SOAP Web Services
12.6. Introspecting SOAP Web Services with WSDL
Example 12.8. Discovering The Available Methods
Example 12.9. Discovering A Method’s Arguments
Example 12.10. Discovering A Method’s Return Values
Example 12.11. Calling A Web Service Through A WSDL Proxy
12.7. Searching Google
Example 12.12. Introspecting Google Web Services
Example 12.13. Searching Google
Example 12.14. Accessing Secondary Information From Google
12.8. Troubleshooting SOAP Web Services
Example 12.15. Calling a Method With an Incorrectly Configured Proxy
Example 12.16. Calling a Method With the Wrong Arguments
Example 12.17. Calling a Method and Expecting the Wrong Number of Return Values
Example 12.18. Calling a Method With An Application-Specific Error

Chapter 13. Unit Testing

13.3. Introducing romantest.py
Example 13.1. romantest.py
13.4. Testing for success
Example 13.2. testToRomanKnownValues
13.5. Testing for failure
Example 13.3. Testing bad input to toRoman
Example 13.4. Testing bad input to fromRoman
13.6. Testing for sanity
Example 13.5. Testing toRoman against fromRoman
Example 13.6. Testing for case

Chapter 14. Test-First Programming

14.1. roman.py, stage 1
Example 14.1. roman1.py
Example 14.2. Output of romantest1.py against roman1.py
14.2. roman.py, stage 2
Example 14.3. roman2.py
Example 14.4. How toRoman works
Example 14.5. Output of romantest2.py against roman2.py
14.3. roman.py, stage 3
Example 14.6. roman3.py
Example 14.7. Watching toRoman handle bad input
Example 14.8. Output of romantest3.py against roman3.py
14.4. roman.py, stage 4
Example 14.9. roman4.py
Example 14.10. How fromRoman works
Example 14.11. Output of romantest4.py against roman4.py
14.5. roman.py, stage 5
Example 14.12. roman5.py
Example 14.13. Output of romantest5.py against roman5.py

Chapter 15. Refactoring

15.1. Handling bugs
Example 15.1. The bug
Example 15.2. Testing for the bug (romantest61.py)
Example 15.3. Output of romantest61.py against roman61.py
Example 15.4. Fixing the bug (roman62.py)
Example 15.5. Output of romantest62.py against roman62.py
15.2. Handling changing requirements
Example 15.6. Modifying test cases for new requirements (romantest71.py)
Example 15.7. Output of romantest71.py against roman71.py
Example 15.8. Coding the new requirements (roman72.py)
Example 15.9. Output of romantest72.py against roman72.py
15.3. Refactoring
Example 15.10. Compiling regular expressions
Example 15.11. Compiled regular expressions in roman81.py
Example 15.12. Output of romantest81.py against roman81.py
Example 15.13. roman82.py
Example 15.14. Output of romantest82.py against roman82.py
Example 15.15. roman83.py
Example 15.16. Output of romantest83.py against roman83.py

15.4. Postscript
Example 15.17. roman9.py
Example 15.18. Output of romantest9.py against roman9.py

Chapter 16. Functional Programming

16.1. Diving in
Example 16.1. regression.py
Example 16.2. Sample output of regression.py
16.2. Finding the path
Example 16.3. fullpath.py
Example 16.4. Further explanation of os.path.abspath
Example 16.5. Sample output from fullpath.py
Example 16.6. Running scripts in the current directory
16.3. Filtering lists revisited
Example 16.7. Introducing filter
Example 16.8. filter in regression.py
Example 16.9. Filtering using list comprehensions instead
16.4. Mapping lists revisited
Example 16.10. Introducing map
Example 16.11. map with lists of mixed datatypes
Example 16.12. map in regression.py
16.6. Dynamically importing modules
Example 16.13. Importing multiple modules at once
Example 16.14. Importing modules dynamically
Example 16.15. Importing a list of modules dynamically
16.7. Putting it all together
Example 16.16. The regressionTest function
Example 16.17. Step 1: Get all the files
Example 16.18. Step 2: Filter to find the files you care about
Example 16.19. Step 3: Map filenames to module names
Example 16.20. Step 4: Mapping module names to modules
Example 16.21. Step 5: Loading the modules into a test suite
Example 16.22. Step 6: Telling unittest to use your test suite

Chapter 17. Dynamic functions

17.2. plural.py, stage 1
Example 17.1. plural1.py
Example 17.2. Introducing re.sub
Example 17.3. Back to plural1.py
Example 17.4. More on negation regular expressions
Example 17.5. More on re.sub
17.3. plural.py, stage 2
Example 17.6. plural2.py
Example 17.7. Unrolling the plural function
17.4. plural.py, stage 3
Example 17.8. plural3.py
17.5. plural.py, stage 4
Example 17.9. plural4.py
Example 17.10. plural4.py continued
Example 17.11. Unrolling the rules definition
Example 17.12. plural4.py, finishing up

Example 17.13. Another look at buildMatchAndApplyFunctions
Example 17.14. Expanding tuples when calling functions
17.6. plural.py, stage 5
Example 17.15. rules.en
Example 17.16. plural5.py
17.7. plural.py, stage 6
Example 17.17. plural6.py
Example 17.18. Introducing generators
Example 17.19. Using generators instead of recursion
Example 17.20. Generators in for loops
Example 17.21. Generators that generate dynamic functions

Chapter 18. Performance Tuning

18.1. Diving in
Example 18.1. soundex/stage1/soundex1a.py
18.2. Using the timeit Module
Example 18.2. Introducing timeit
18.3. Optimizing Regular Expressions
Example 18.3. Best Result So Far: soundex/stage1/soundex1e.py
18.4. Optimizing Dictionary Lookups
Example 18.4. Best Result So Far: soundex/stage2/soundex2c.py
18.5. Optimizing List Operations
Example 18.5. Best Result So Far: soundex/stage2/soundex2c.py

Appendix E. Revision history

Revision History	
Revision 5.4	2004-05-20
Added Section 12.1, “Diving In”. Added Section 12.2, “Installing the SOAP Libraries”. Added Section 12.3, “First Steps with SOAP”. Added Section 12.4, “Debugging SOAP Web Services”. Added Section 12.5, “Introducing WSDL”. Added Section 12.6, “Introspecting SOAP Web Services with WSDL”. Added Section 12.7, “Searching Google”. Added Section 12.8, “Troubleshooting SOAP Web Services”. Added Section 12.9, “Summary”. Incorporated technical reviewer revisions in Chapter 16, *Functional Programming* and Chapter 18, *Performance Tuning*.	
Revision 5.3	2004-05-12
Added example to Section 18.3, “Optimizing Regular Expressions”. Thanks, Paul. Incorporated copyediting revisions into Chapter 5, *Objects and Object-Orientation* and Chapter 6, *Exceptions and File Handling*. Fixed URL of Section 9.7, “Segue”.	
Revision 5.2	2004-05-09
Fixed URL of Section 14.1, “roman.py, stage 1”. Added Section 18.1, “Diving in”. Added Section 18.2, “Using the timeit Module”. Added Section 18.3, “Optimizing Regular Expressions”. Added Section 18.4, “Optimizing Dictionary Lookups”. Added Section 18.5, “Optimizing List Operations”. Added Section 18.6, “Optimizing String Manipulation”. Added Section 18.7, “Summary”.	
Revision 5.1	2004-05-05
Clarified Example 7.7, “Checking for Tens” and Example 7.8, “Validating Roman Numerals with {n,m}”. Clarified Example 7.10, “Finding Numbers”. Fixed typo in Example 11.6, “Testing Last-Modified”. Thanks, Jesir. Fixed typo in Example 3.11, “The Difference between extend and append”. Thanks, Daniel. Incorporated technical reviewer revisions. Incorporated copy editor revisions in Chapter 1, *Installing Python*, Chapter 2, *Your First Python Program*, Chapter 3, *Native Datat*	
Revision 5.0	2004-04-16
Added Section 11.1, “Diving in”. Added Section 11.2, “How not to fetch data over HTTP”. Added Section 11.3, “Features of HTTP”. Added Section 11.4, “Debugging HTTP web services”. Added Section 11.5, “Setting the User-Agent”. Added Section 11.6, “Handling Last-Modified and ETag”. Added Section 11.7, “Handling redirects”. Added Section 11.8, “Handling compressed data”. Added Section 11.9, “Putting it all together”. Added Section 11.10, “Summary”. Added Example 3.11, “The Difference between extend and append”. Changed descriptions of how to download Python throughout Chapter 1, *Installing Python* to be more generic and less version-spec Changed references of "module" to "program" in Section 2.1, “Diving in” and Section 2.4, “Everything Is an Object” since we have Added explicit instructions in Section 2.4, “Everything Is an Object” for the reader to open their Python IDE and follow along with	

Changed all examples and descriptions that referred to truth values 1 and 0 to refer to True and False Updated Example 3.22, “String Formatting vs. Concatenating” to show new Python 2.3 binary message. Fixed typo in Example 17.19, “Using generators instead of recursion”. Fixed typo in Section 7.7, “Summary”. Fixed typo in Example 17.9, “plural4.py”.	
Revision 4.9	2004-03-25
Finished Section 16.7, “Putting it all together”. Added Section 16.8, “Summary”. Split unit testing introduction into two chapters, Chapter 13, *Unit Testing* and Chapter 14, *Test-First Programming*. Fixed typo in Example 17.12, “plural4.py, finishing up”. Fixed typo in Example 17.18, “Introducing generators”.	
Revision 4.8	2004-03-25
Finished Section 17.7, “plural.py, stage 6”. Finished Section 17.8, “Summary”. Fixed broken links in Appendix A, *Further reading*, Appendix B, *A 5-minute review*, Appendix C, *Tips and tricks*, Appendix D, *Lis*	
Revision 4.7	2004-03-21
Added Section 17.1, “Diving in”. Added Section 17.2, “plural.py, stage 1”. Added Section 17.3, “plural.py, stage 2”. Added Section 17.4, “plural.py, stage 3”. Added Section 17.5, “plural.py, stage 4”. Added Section 17.6, “plural.py, stage 5”. Added Section 17.7, “plural.py, stage 6” (unfinished). Added Section 17.8, “Summary” (unfinished).	
Revision 4.6	2004-03-14
Finished Section 7.4, “Using the {n,m} Syntax”. Finished Section 7.5, “Verbose Regular Expressions”. Finished Section 7.6, “Case study: Parsing Phone Numbers”. Expanded Section 7.7, “Summary”.	
Revision 4.5	2004-03-07
Added Section 7.1, “Diving In”. Added Section 7.4, “Using the {n,m} Syntax” (incomplete). Added Section 7.5, “Verbose Regular Expressions” (incomplete). Added Section 7.6, “Case study: Parsing Phone Numbers” (incomplete). Added Section 7.7, “Summary”. Moved Section 7.2, “Case Study: Street Addresses” and Section 7.3, “Case Study: Roman Numerals” to regular expressions chapter Added Example 6.20, “Listing Directories with glob”. Added Example 6.7, “Writing to Files”. Added Example 5.11, “Inheriting Directly from Built-In Datatype dict”. Added Example 10.11, “Printing to stderr”. Added Example 4.12, “Creating a Dispatcher with getattr” and Example 4.13, “getattr Default Values”. Added Example 2.6, “if Statements”. Added Example 3.23, “Formatting Numbers”. Split Chapter 5, *Objects and Object-Orientation* into 2 chapters: Chapter 5, *Objects and Object-Orientation* and Chapter 6, *Exceptio* Split Chapter 9, *XML Processing* into 2 chapters: Chapter 9, *XML Processing* and Chapter 10, *Scripts and Streams*. Split Chapter 13, *Unit Testing* into 2 chapters: Chapter 13, *Unit Testing* and Chapter 15, *Refactoring*. Renamed apihelper.py to info.py in Chapter 4, *The Power Of Introspection*. Fixed incorrect back-reference in Section 8.5, “locals and globals”. Fixed broken example links in Section 8.1, “Diving in”. Fixed missing line in example in Section 9.1, “Diving in”.	

Fixed typo in Section 8.2, “Introducing sgmllib.py”.	
Revision 4.4	2003-10-08
Added Section 1.1, “Which Python is right for you?”. Added Section 1.2, “Python on Windows”. Added Section 1.3, “Python on Mac OS X”. Added Section 1.4, “Python on Mac OS 9”. Added Section 1.5, “Python on RedHat Linux”. Added Section 1.6, “Python on Debian GNU/Linux”. Added Section 1.7, “Python Installation from Source”. Added Section 1.9, “Summary”. Removed preface.· Fixed typo in Example 3.27, “Output of odbchelper.py”. Added link to PEP 257 in Section 2.3, “Documenting Functions”. Fixed link to *How to Think Like a Computer Scientist* (http://www.ibiblio.org/obp/thinkCSpy/) in Section 3.4.2, “Assigning Multipl Added note about implied assert in Section 3.3, “Introducing Tuples”.	
Revision 4.3	2003-09-28
Added Section 16.6, “Dynamically importing modules”. Added Section 16.7, “Putting it all together” (incomplete). Fixed links in Appendix F, *About the book.*	
Revision 4.2.1	2003-09-17
Fixed links on index page.· Fixed syntax highlighting.·	
Revision 4.2	2003-09-12
Fixed typos in Section 16.4, “Mapping lists revisited”, Section 16.3, “Filtering lists revisited”, Section 7.2, “Case Study: Street Add Fixed external link in Section 5.3, “Defining Classes”. Thanks, Harry. Changed wording at the end of Section 4.5, “Filtering Lists”. Thanks, Paul. Added sentence in Section 13.5, “Testing for failure” to make it clearer that we’re passing a function to [illegible]not a function name as a st Fixed typo in Section 8.8, “Introducing dialect.py”. Thanks, Wellie. Fixed links to dialectized examples.· Fixed external link to the history of Roman numerals. Thanks to many concerned Roman numeral fans around the world.·	
Revision 4.1	2002-07-28
Added Section 10.3, “Caching node lookups”. Added Section 10.4, “Finding direct children of a node”. Added Section 10.5, “Creating separate handlers by node type”. Added Section 10.6, “Handling command-line arguments”. Added Section 10.7, “Putting it all together”. Added Section 10.8, “Summary”. Fixed typo in Section 6.5, “Working with Directories”. It’s [illegible]not [illegible]Thanks, Abhishek. Fixed typo in Section 3.7, “Joining Lists and Splitting Strings”. When evaluated (instead of printed), the Python IDE will display sin Changed [illegible]example in Section 4.8, “Putting It All Together” to use a user-defined function, since Python 2.2 obsoleted the old examp Fixed typo in Section 9.4, “Unicode”, "anyway" to "anywhere". Thanks Frank. Fixed typo in Section 13.6, “Testing for sanity”, doubled word "accept". Thanks Ralph. Fixed typo in Section 15.3, “Refactoring”, [illegible]matches 0 to 3 [illegible]characters, not 4. Thanks Ralph. Clarified and expanded explanation of implied slice indices in Example 3.9, “Slicing Shorthand”. Thanks Petr. Added historical note in Section 5.5, “Exploring UserDict: A Wrapper Class” now that Python 2.2 supports subclassing built-in data Added explanation of [illegible]dictionary method in Example 5.9, “Defining the UserDict Class”. Thanks Petr. Clarified Python’s lack of overloading in Section 5.5, “Exploring UserDict: A Wrapper Class”. Thanks Petr. Fixed typo in Example 8.8, “Introducing BaseHTMLProcessor”. HTML comments end with two dashes and a bracket, not one. Tha Changed tense of note about nested scopes in Section 8.5, “locals and globals” now that Python 2.2 is out. Thanks Petr. Fixed typo in Example 8.14, “Dictionary-based string formatting in BaseHTMLProcessor.py”; a space should have been a non-brea	

Added title to note on derived classes in Section 5.5, “Exploring UserDict: A Wrapper Class”. Thanks Petr.
Added title to note on downloading in Section 15.3, “Refactoring”. Thanks Petr.
Fixed typesetting problem in Example 16.6, “Running scripts in the current directory”; tabs should have been spaces, and the line n
Fixed capitalization typo in the tip on truth values in Section 3.2, “Introducing Lists”. It’s and not and Thanks to everyone who
Changed section titles of Section 3.1, “Introducing Dictionaries”, Section 3.2, “Introducing Lists”, and Section 3.3, “Introducing Tu
Upgraded to version 1.52 of the DocBook XSL stylesheets.·
Upgraded to version 6.52 of the SAXON XSLT processor from Michael Kay.·
Various accessibility-related stylesheet tweaks.·
Somewhere between this revision and the last one, she said yes. The wedding will be next spring.·

Revision 4.0-2	2002-04-26

Fixed typo in Example 4.15, “Introducing and”.
Fixed typo in Example 2.4, “Import Search Path”.
Fixed Windows help file (missing table of contents due to base stylesheet changes).·

Revision 4.0	2002-04-19

Expanded Section 2.4, “Everything Is an Object” to include more about import search paths.
Fixed typo in Example 3.7, “Negative List Indices”. Thanks to Brian for the correction.
Rewrote the tip on truth values in Section 3.2, “Introducing Lists”, now that Python has a separate boolean datatype.
Fixed typo in Section 5.2, “Importing Modules Using from module import” when comparing syntax to Java. Thanks to Rick for the
Added note in Section 5.5, “Exploring UserDict: A Wrapper Class” about derived classes always overriding ancestor classes.
Fixed typo in Example 5.18, “Modifying Class Attributes”. Thanks to Kevin for the correction.
Added note in Section 6.1, “Handling Exceptions” that you can define and raise your own exceptions. Thanks to Rony for the sugge
Fixed typo in Example 8.17, “Handling specific tags”. Thanks for Rick for the correction.
Added note in Example 8.18, “SGMLParser” about what the return codes mean. Thanks to Howard for the suggestion.
Added function when creating instance in Example 10.6, “openAnything”. Thanks to Ganesan for the idea.
Added link in Section 13.3, “Introducing romantest.py” to explanation of why test cases belong in a separate file.
Changed Section 16.2, “Finding the path” to use instead of Thanks to Marc for the idea.
Added code samples (and for the upcoming regular expressions chapter.·
Updated and expanded list of Python distributions on home page.·

Revision 3.9	2002-01-01

Added Section 9.4, “Unicode”.
Added Section 9.5, “Searching for elements”.
Added Section 9.6, “Accessing element attributes”.
Added Section 10.1, “Abstracting input sources”.
Added Section 10.2, “Standard input, output, and error”.
Added simple counter loop examples (good usage and bad usage) in Section 6.3, “Iterating with for Loops”. Thanks to Kevin for th
Fixed typo in Example 3.25, “The keys, values, and items Functions” (two elements of were reversed).
Fixed mistake in Section 4.3, “Using type, str, dir, and Other Built-In Functions” with regards to the name of the module. Thanks t
Added additional example in Section 16.2, “Finding the path” to show how to run unit tests in the current working directory, instead
Modified explanation of how to derive a negative list index from a positive list index in Example 3.7, “Negative List Indices”. Than
Updated links on home page for downloading latest version of Python.·
Added link on home page to Bruce Eckel’s preliminary draft of *Thinking in Python* (http://www.mindview.net/Books/TIPython), a

Revision 3.8	2001-11-18

Added Section 16.2, “Finding the path”.
Added Section 16.3, “Filtering lists revisited”.
Added Section 16.4, “Mapping lists revisited”.
Added Section 16.5, “Data-centric programming”.
Expanded sample output in Section 16.1, “Diving in”.
Finished Section 9.3, “Parsing XML”.

Revision 3.7	2001-09-30

Added Section 9.2, “Packages”.

Added Section 9.3, “Parsing XML”.
Cleaned up introductory paragraph in Section 9.1, “Diving in”. Thanks to Matt for this suggestion.
Added Java tip in Section 5.2, “Importing Modules Using from module import”. Thanks to Ori for this suggestion.
Fixed mistake in Section 4.8, “Putting It All Together” where I implied that you could not use to compare to a null value in Python
Clarified in Section 3.2, “Introducing Lists” where I said that was equivalent to The result is the same, but is faster because it doe
Fixed mistake in Section 3.2, “Introducing Lists” where I said that was equivalent to In fact, it’s equivalent to since it doesn’t cre
Fixed typographical laziness in Chapter 2, *Your First Python Program*; when I was writing it, I had not yet standardized on putting
Fixed mistake in Section 2.2, “Declaring Functions” where I said that statically typed languages always use explicit variable + datat
Added link to Spanish translation (http://es.diveintopython.org/).·

Revision 3.6.4	2001-09-06

Added code in to handle non-HTML entity references, and added a note about it in Section 8.4, “Introducing BaseHTMLProcessor
Modified Example 8.11, “Introducing globals” to include in the output.

Revision 3.6.3	2001-09-04

Fixed typo in Section 9.1, “Diving in”.
Added link to Korean translation (http://kr.diveintopython.org/html/index.htm).·

Revision 3.6.2	2001-08-31

Fixed typo in Section 13.6, “Testing for sanity” (the last requirement was listed twice).

Revision 3.6	2001-08-31

Finished Chapter 8, *HTML Processing* with Section 8.9, “Putting it all together” and Section 8.10, “Summary”.
Added Section 15.4, “Postscript”.
Started Chapter 9, *XML Processing* with Section 9.1, “Diving in”.
Started Chapter 16, *Functional Programming* with Section 16.1, “Diving in”.
Fixed long-standing bug in colorizing script that improperly colorized the examples in Chapter 8, *HTML Processing*.
Added link to French translation (http://fr.diveintopython.org/toc.html). They did the right thing and translated the source XML, so
Upgraded to version 1.43 of the DocBook XSL stylesheets.·
Upgraded to version 6.43 of the SAXON XSLT processor from Michael Kay.·
Massive stylesheet changes, moving away from a table-based layout and towards more appropriate use of cascading style sheets. Ur
Moved to Ant (http://jakarta.apache.org/ant/) to have better control over the build process, which is especially important now that I’
Consolidated the available downloadable archives; previously, I had different files for each platform, because the .zip files that Pyth
Now hosting the complete XML source, XSL stylesheets, and associated scripts and libraries on SourceForge. There’s also CVS ac
Re-licensed the example code under the new-and-improved GPL-compatible Python 2.1.1 license (http://www.python.org/2.1.1/lice

Revision 3.5	2001-06-26

Added explanation of strong/weak/static/dynamic datatypes in Section 2.2, “Declaring Functions”.
Added case-sensitivity example in Section 3.1, “Introducing Dictionaries”.
Use in Chapter 5, *Objects and Object-Orientation* to compensate for inferior operating systems whose files aren’t case-sensitive.
Fixed indentation problems in code samples in PDF version.·

Revision 3.4	2001-05-31

Added Section 14.5, “roman.py, stage 5”.
Added Section 15.1, “Handling bugs”.
Added Section 15.2, “Handling changing requirements”.
Added Section 15.3, “Refactoring”.
Added Section 15.5, “Summary”.
Fixed yet another stylesheet bug that was dropping nested tags.·

Revision 3.3	2001-05-24

Added Section 13.2, “Diving in”.
Added Section 13.3, “Introducing romantest.py”.
Added Section 13.4, “Testing for success”.
Added Section 13.5, “Testing for failure”.
Added Section 13.6, “Testing for sanity”.

Added Section 14.1, “roman.py, stage 1”. Added Section 14.2, “roman.py, stage 2”. Added Section 14.3, “roman.py, stage 3”. Added Section 14.4, “roman.py, stage 4”. Tweaked stylesheets in an endless quest for complete Netscape/Mozilla compatibility.·	
Revision 3.2	2001-05-03
Added Section 8.8, “Introducing dialect.py”. Added Section 7.2, “Case Study: Street Addresses”. Fixed bug in method that would produce incorrect declarations (adding a space where it couldn’t be).· Fixed bug in CSS (introduced in 2.9) where body background color was missing.·	
Revision 3.1	2001-04-18
Added code in to handle declarations, now that Python 2.1 supports them.· Added note about nested scopes in Section 8.5, “locals and globals”. Fixed obscure bug in Example 8.1, “BaseHTMLProcessor.py” where attribute values with character entities would not be properly Now recommending (but not requiring) Python 2.1, due to its support of declarations in Updated download links on the home page (http://diveintopython.org/) to point to Python 2.1, where available.· Moved to versioned filenames, to help people who redistribute the book.·	
Revision 3.0	2001-04-16
Fixed minor bug in code listing in Chapter 8, *HTML Processing*. Added link to Chinese translation on home page (http://diveintopython.org/).·	
Revision 2.9	2001-04-13
Added Section 8.5, “locals and globals”. Added Section 8.6, “Dictionary-based string formatting”. Tightened code in Chapter 8, *HTML Processing*, specifically to use fewer and simpler regular expressions. Fixed a stylesheet bug that was inserting blank pages between chapters in the PDF version.· Fixed a script bug that was stripping the from the home page (http://diveintopython.org/).· Added link to Python Cookbook (http://www.activestate.com/ASPN/Python/Cookbook/), and added a few links to individual recipe Switched to Google (http://www.google.com/services/free.html) for searching on Upgraded to version 1.36 of the DocBook XSL stylesheets, which was much more difficult than it sounds. There may still be linger	
Revision 2.8	2001-03-26
Added Section 8.3, “Extracting data from HTML documents”. Added Section 8.4, “Introducing BaseHTMLProcessor.py”. Added Section 8.7, “Quoting attribute values”. Tightened up code in Chapter 4, *The Power Of Introspection*, using the built-in function instead of manually checking types. Moved Section 5.2, “Importing Modules Using from module import” from Chapter 4, *The Power Of Introspection* to Chapter 5, *Ob* Fixed typo in code example in Section 5.1, “Diving In” (added colon). Added several additional downloadable example scripts.· Added Windows Help output format.·	
Revision 2.7	2001-03-16
Added Section 8.2, “Introducing sgmllib.py”. Tightened up code in Chapter 8, *HTML Processing*. Changed code in Chapter 2, *Your First Python Program* to use method instead of Moved Section 3.4.2, “Assigning Multiple Values at Once” section to Chapter 2, *Your First Python Program*. Edited note about string method, and provided a link to the new entry in *The Whole Python FAQ* (http://www.python.org/doc/FAQ Rewrote Section 4.6, “The Peculiar Nature of and and or” to emphasize the fundamental nature of and and de-emphasize the trick Reorganized language comparisons into ·	
Revision 2.6	2001-02-28
The PDF and Word versions now have colorized examples, an improved table of contents, and properly indented and · The Word version is now in native Word format, compatible with Word 97.·	

	The PDF and text versions now have fewer problems with improperly converted special characters (like trademark symbols and cur Added link to download Word version for UNIX, in case some twisted soul wants to import it into StarOffice or something.· Fixed several chapters which were missing titles.· Fixed stylesheets to work around bug in Internet Explorer 5 for Mac OS which caused colorized words in the examples to be displa Fixed archive corruption in Mac OS downloads.· In first section of each chapter, added link to download examples. (My access logs show that people skim or skip the two pages whe Tightened the home page (http://diveintopython.org/) and preface even more, in the hopes that someday someone will read them.· Soon I hope to get back to actually writing this book instead of debugging it.·
Revision 2.5	2001-02-23
	Added Section 6.4, “Using sys.modules”. Added Section 6.5, “Working with Directories”. Moved Example 6.17, “Splitting Pathnames” from Section 3.4.2, “Assigning Multiple Values at Once” to Section 6.5, “Working wi Added Section 6.6, “Putting It All Together”. Added Section 5.10, “Summary”. Added Section 8.1, “Diving in”. Fixed program listing in Example 6.10, “Iterating Through a Dictionary” which was missing a colon.
Revision 2.4.1	2001-02-12
	Changed newsgroup links to use "news:" protocol, now that Deja is defunct.· Added file sizes to download links.·
Revision 2.4	2001-02-12
	Added "further reading" links in most sections, and collated them in Appendix A, *Further reading*. Added URLs in parentheses next to external links in text version.·
Revision 2.3	2001-02-09
	Rewrote some of the code in Chapter 5, *Objects and Object-Orientation* to use class attributes and a better example of multi-variabl Reorganized Chapter 5, *Objects and Object-Orientation* to put the class sections first. Added Section 5.8, “Introducing Class Attributes”. Added Section 6.1, “Handling Exceptions”. Added Section 6.2, “Working with File Objects”. Merged the "review" section in Chapter 5, *Objects and Object-Orientation* into Section 5.1, “Diving In”. Colorized all program listings and examples.· Fixed important error in Section 2.2, “Declaring Functions”: functions that do not explicitly return a value return None so you *can* assign Added minor clarifications to Section 2.3, “Documenting Functions”, Section 2.4, “Everything Is an Object”, and Section 3.4, “Dec
Revision 2.2	2001-02-02
	Edited Section 4.4, “Getting Object References With getattr”. Added titles to <a> tags, so they can have their cute little tooltips too.· Changed the look of the revision history page.· Fixed problem I introduced yesterday in my HTML post-processing script that was causing invalid HTML character references and Upgraded to version 1.29 of the DocBook XSL stylesheets.·
Revision 2.1	2001-02-01
	Rewrote the example code of Chapter 4, *The Power Of Introspection* to use getattr instead of hasattr and rewrote explanatory text to match. Added example of list operators in Section 3.2, “Introducing Lists”. Added links to relevant sections in the summary lists at the end of each chapter (Section 3.8, “Summary” and Section 4.9, “Summar
Revision 2.0	2001-01-31
	Split Section 5.6, “Special Class Methods” into three sections, Section 5.5, “Exploring UserDict: A Wrapper Class”, Section 5.6, “S Changed notes on garbage collection to point out that Python 2.0 and later can handle circular references without additional coding. Fixed UNIX downloads to include all relevant files.·
Revision 1.9	2001-01-15
	Removed introduction to Chapter 2, *Your First Python Program*. Removed introduction to Chapter 4, *The Power Of Introspection*.

Removed introduction to Chapter 5, *Objects and Object-Orientation*. Edited text ruthlessly. I tend to ramble.·	
Revision 1.8	2001-01-12
Added more examples to Section 3.4.2, “Assigning Multiple Values at Once”. Added Section 5.3, “Defining Classes”. Added Section 5.4, “Instantiating Classes”. Added Section 5.6, “Special Class Methods”. More minor stylesheet tweaks, including adding titles to [illegible]ags, which, if your browser is cool enough, will display a description of th	
Revision 1.71	2001-01-03
Made several modifications to stylesheets to improve browser compatibility.·	
Revision 1.7	2001-01-02
Added introduction to Chapter 2, *Your First Python Program*. Added introduction to Chapter 4, *The Power Of Introspection*. Added review section to Chapter 5, *Objects and Object-Orientation* [later removed] Added Section 5.9, “Private Functions”. Added Section 6.3, “Iterating with for Loops”. Added Section 3.4.2, “Assigning Multiple Values at Once”. Wrote scripts to convert book to new output formats: one single HTML file, PDF, Microsoft Word 97, and plain text.· Registered the [illegible]omain and moved the book there, along with links to download the book in all available output formats for offline Modified the XSL stylesheets to change the header and footer navigation that displays on each page. The top of each page is brande	
Revision 1.6	2000-12-11
Added Section 4.8, “Putting It All Together”. Finished Chapter 4, *The Power Of Introspection* with Section 4.9, “Summary”. Started Chapter 5, *Objects and Object-Orientation* with Section 5.1, “Diving In”.	
Revision 1.5	2000-11-22
Added Section 4.6, “The Peculiar Nature of and and or”. Added Section 4.7, “Using lambda Functions”. Added appendix that lists section abstracts.· Added appendix that lists tips.· Added appendix that lists examples.· Added appendix that lists revision history.· Expanded example of mapping lists in Section 3.6, “Mapping Lists”. Encapsulated several more common phrases into entities.· Upgraded to version 1.25 of the DocBook XSL stylesheets.·	
Revision 1.4	2000-11-14
Added Section 4.5, “Filtering Lists”. Added [illegible]ocumentation to Section 4.3, “Using type, str, dir, and Other Built-In Functions”. Added [illegible]xample in Section 3.3, “Introducing Tuples”. Added additional note about [illegible]rick under MacPython.· Switched to the SAXON XSLT processor from Michael Kay.· Upgraded to version 1.24 of the DocBook XSL stylesheets.· Added db-html processing instructions with explicit filenames of each chapter and section, to allow deep links to content even if I a Made several common phrases into entities for easier reuse.· Changed several [illegible]ags to [illegible]	
Revision 1.3	2000-11-09
Added section on dynamic code execution.· Added links to relevant section/example wherever I refer to previously covered concepts.· Expanded introduction of chapter 2 to explain what the function actually does.· Explicitly placed example code under the GNU General Public License and added appendix to display license. [Note 8/16/2001: co	

Changed links to licenses to use flags, now that I know how to use them.·	
Revision 1.2	2000-11-06
Added first four sections of chapter 2.· Tightened up preface even more, and added link to Mac OS version of Python.· Filled out examples in "Mapping lists" and "Joining strings" to show logical progression.· Added output in chapter 1 summary.·	
Revision 1.1	2000-10-31
Finished chapter 1 with sections on mapping and joining, and a chapter summary.· Toned down the preface, added links to introductions for non-programmers.· Fixed several typos.·	
Revision 1.0	2000-10-30
Initial publication·	

Appendix F. About the book

This book was written in DocBook XML (http://www.oasis-open.org/docbook/) using Emacs (http://www.gnu.org/software/emacs/), and com

If you’re interested in learning more about DocBook for technical writing, you can download the XML source (http://diveintopython.org/dow

Appendix G. GNU Free Documentation License

Version 1.1, March 2000

Copyright (C) 2000 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA Everyone is permitt

G.0. Preamble

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the e

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It compl

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free progra

G.1. Applicability and definitions

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the t

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifica

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publ

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that sa

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the gener

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this Lic

G.2. Verbatim copying

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyrig

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

G.3. Copying in quantity

If you publish printed copies of the Document numbering more than 100, and the Document’s license notice requires Cover Texts, you must

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the ac

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transpare

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give t

G.4. Modifications

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release t

Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (whi
List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Ve

State on the Title page the name of the publisher of the Modified Version, as the publisher.3.
Preserve all the copyright notices of the Document.4.
Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.5.
Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the
Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.7.
Include an unaltered copy of this License.8.
Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of
Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise
In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substan
Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are
Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.13.
Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.14.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied f

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties-

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the li

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imp

G.5. Combining documents

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "Histor

G.6. Collections of documents

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies o

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this

G.7. Aggregation with independent works

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the

G.8. Translation

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing

G.9. Termination

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to

G.10. Future revisions of this license

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new vers

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this Li

G.11. How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and licens

> Copyright (c) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the C

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover T

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of fre

Appendix H. Python license

H.A. History of the software

Python was created in the early 1990s by Guido van Rossum at Stichting Mathematisch Centrum (CWI) in the Netherlands as a successor of

Following the release of Python 1.6, and after Guido van Rossum left CNRI to work with commercial software developers, it became clear th

After Python 2.0 was released by BeOpen.com, Guido van Rossum and the other PythonLabs developers joined Digital Creations. All intelle

Thanks to the many outside volunteers who have worked under Guido's direction to make these releases possible.

H.B. Terms and conditions for accessing or otherwise using Python

H.B.1. PSF license agreement

This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"), and the Individual or Organization ("Licensee"
Subject to the terms and conditions of this License Agreement, PSF hereby grants Licensee a nonexclusive, royalty-free, world-wide
In the event Licensee prepares a derivative work that is based on or incorporates Python 2.1.1 or any part thereof, and wants to make
PSF is making Python 2.1.1 available to Licensee on an "AS IS" basis. PSF MAKES NO REPRESENTATIONS OR WARRANTIE
PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 2.1.1 FOR ANY INCIDENTAL, SPECIA
This License Agreement will automatically terminate upon a material breach of its terms and conditions.6.
Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between PSF a
By copying, installing or otherwise using Python 2.1.1, Licensee agrees to be bound by the terms and conditions of this License Agr

H.B.2. BeOpen Python open source license agreement version 1

This LICENSE AGREEMENT is between BeOpen.com ("BeOpen"), having an office at 160 Saratoga Avenue, Santa Clara, CA 950
Subject to the terms and conditions of this BeOpen Python License Agreement, BeOpen hereby grants Licensee a non-exclusive, roy
BeOpen is making the Software available to Licensee on an "AS IS" basis. BEOPEN MAKES NO REPRESENTATIONS OR WAR
BEOPEN SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF THE SOFTWARE FOR ANY INCIDENTAL
This License Agreement will automatically terminate upon a material breach of its terms and conditions.5.
This License Agreement shall be governed by and interpreted in all respects by the law of the State of California, excluding conflict
By copying, installing or otherwise using the software, Licensee agrees to be bound by the terms and conditions of this License Agre

H.B.3. CNRI open source GPL-compatible license agreement

This LICENSE AGREEMENT is between the Corporation for National Research Initiatives, having an office at 1895 Preston White
Subject to the terms and conditions of this License Agreement, CNRI hereby grants Licensee a nonexclusive, royalty-free, world-wi
In the event Licensee prepares a derivative work that is based on or incorporates Python 1.6.1 or any part thereof, and wants to make
CNRI is making Python 1.6.1 available to Licensee on an "AS IS" basis. CNRI MAKES NO REPRESENTATIONS OR WARRAN
CNRI SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 1.6.1 FOR ANY INCIDENTAL, SPEC
This License Agreement will automatically terminate upon a material breach of its terms and conditions.6.
This License Agreement shall be governed by the federal intellectual property law of the United States, including without limitation
By clicking on the "ACCEPT" button where indicated, or by copying, installing or otherwise using Python 1.6.1, Licensee agrees to

H.B.4. CWI permissions statement and disclaimer

Copyright (c) 1991 - 1995, Stichting Mathematisch Centrum Amsterdam, The Netherlands. All rights reserved.

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provid

STICHTING MATHEMATISCH CENTRUM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING