CHAPTER 0. INSTALLING PYTHON

❝ *Tempora mutantur nos et mutamur in illis. (Times change, and we change with them.)* ❞

— ancient Roman proverb

0.1. DIVING IN

Before you can start programming in Python 3, you need to install it. Or do you?

0.2. WHICH PYTHON IS RIGHT FOR YOU?

If you're using an account on a hosted server, your ISP may have already installed Python 3. If you're running Linux at home, you may already have Python 3, too. Most popular GNU/Linux distributions come with Python 2 in the default installation; a small but growing number of distributions also include Python 3. Mac OS X includes a command-line version of Python 2, but as of this writing it does not include Python 3. Microsoft Windows does not come with any version of Python. But don't despair! You can point-and-click your way through installing Python, regardless of what operating system you have.

The easiest way to check for Python 3 on your Linux or Mac OS X system is from the command line. Once you're at a command line prompt, just type `python3` (all lowercase, no spaces), press ENTER, and see what happens. On my home Linux system, Python 3.1 is already installed, and this command gets me into the *Python interactive shell*.

```
mark@atlantis:~$ python3
Python 3.1 (r31:73572, Jul 28 2009, 06:52:23)
[GCC 4.2.4 (Ubuntu 4.2.4-1ubuntu4)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

(Type `exit()` and press ENTER to exit the Python interactive shell.)

- In Files, you'll learn the difference between reading files in "binary" and "text" mode. Reading (and writing!) files in text mode requires an `encoding` parameter. Some text file methods count characters, but other methods count bytes. If your code assumes that one character == one byte, it *will* break on multi-byte characters.
- In HTTP Web Services, the `httplib2` module fetches headers and data over HTTP. HTTP headers are returned as strings, but the HTTP body is returned as bytes.
- In Serializing Python Objects, you'll learn why the `pickle` module in Python 3 defines a new data format that is backwardly incompatible with Python 2. (Hint: it's because of bytes and strings.) Also, Python 3 supports serializing objects to and from JSON, which doesn't even have a `bytes` type. I'll show you how to hack around that.
- In Case study: porting `chardet` to Python 3, it's just a bloody mess of bytes and strings everywhere.

Even if you don't care about Unicode (oh but you will), you'll want to read about string formatting in Python 3, which is completely different from Python 2.

Iterators are everywhere in Python 3, and I understand them a lot better than I did five years ago when I wrote "Dive Into Python". You need to understand them too, because lots of functions that used to return lists in Python 2 will now return iterators in Python 3. At a minimum, you should read the second half of the Iterators chapter and the second half of the Advanced Iterators chapter.

By popular request, I've added an appendix on Special Method Names, which is kind of like the Python docs "Data Model" chapter but with more snark.

When I was writing "Dive Into Python", all of the available XML libraries sucked. Then Fredrik Lundh wrote ElementTree, which doesn't suck at all. The Python gods wisely incorporated ElementTree into the standard library, and now it forms the basis for my new XML chapter. The old ways of parsing XML are still around, but you should avoid them, because they suck!

Also new in Python — not in the language but in the community — is the emergence of code repositories like The Python Package Index (PyPI). Python comes with utilities to package your code in standard formats and distribute those packages on PyPI. Read Packaging Python Libraries for details.

1.9. EVERYTHING IS CASE-SENSITIVE

All names in Python are case-sensitive: variable names, function names, class names, module names, exception names. If you can get it, set it, call it, construct it, import it, or raise it, it's case-sensitive.

```
>>> an_integer = 1
>>> an_integer
1
>>> AN_INTEGER
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'AN_INTEGER' is not defined
>>> An_Integer
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'An_Integer' is not defined
>>> an_inteGer
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'an_inteGer' is not defined
```

And so on.