WELCOME TO PYTHON!

Chapter Topics

• What is Python, Its History and Features
• Where to Obtain Python
• How to Install and Run Python
• Python Documentation
• Comparing Python
Our introductory chapter provides some background on what Python is, where it came from, and what some of its “bullet points” are. Once we have stimulated your interest and enthusiasm, we describe how you can obtain Python and get it up and running on your system. Finally, the exercises at the end of the chapter will make you comfortable with using Python, both in the interactive interpreter and also in creating scripts and executing them.

1.1 What Is Python?

Python is an uncomplicated and robust programming language that delivers both the power and complexity of traditional compiled languages along with the ease-of-use (and then some) of simpler scripting and interpreted languages. You’ll be amazed at how quickly you’ll pick up the language as well as what kind of things you can do with Python, not to mention the things that have already been done. Your imagination will be the only limit.
1.2 History of Python

Work on Python began in late 1989 by Guido van Rossum, then at CWI in the Netherlands, and eventually released for public distribution in early 1991. How did it all begin? Innovative languages are usually born from one of two motivations: a large well-funded research project or general frustration due to the lack of tools that were needed at the time to accomplish mundane and/or time-consuming tasks, many of which could be automated.

At the time, van Rossum was a researcher with considerable language design experience with the interpreted language ABC, also developed at CWI, but he was unsatisfied with its ability to be developed into something more. Some of the tools he envisioned were for performing general system administration tasks, so he also wanted access to the power of system calls that were available through the Amoeba distributed operating system. Although an Amoeba-specific language was given some thought, a generalized language made more sense, and late in 1989, the seeds of Python were sown.

1.3 Features of Python

Although practically a decade in age, Python is still somewhat relatively new to the general software development industry. We should, however, use caution with our use of the word “relatively,” as a few years seem like decades when developing on “Internet time.”

When people ask, “What is Python?” it is difficult to say any one thing. The tendency is to want to blurt out all the things that you feel Python is in one breath. Python is ______ (fill-in-the-blanks here) ______. Just what are some of those blanks? For your sanity, we will elucidate on each here... one at a time.

1.3.1 High-level

It seems that with every generation of languages, we move to a higher level. Assembly was a godsend for those who struggled with machine code, then came FORTRAN, C, and Pascal, all of which took computing to another plane and created the software development industry. These languages then evolved into the current compiled systems languages C++ and Java. And further still we climb, with powerful, system-accessible, interpreted scripting languages like Tcl, Perl, and Python. Each of these languages has higher-level data structures that reduce the “framework” development time which was once required. Useful types like Python’s lists (resizeable arrays) and dictio-
naries (hash tables) are built into the language. Providing these crucial building blocks encourages their use and minimizes development time as well as code size, resulting in more readable code. Implementing them in C is complicated and often frustrating due to the necessities of using structures and pointers, not to mention repetitious if some forms of the same data structures require implementation for every large project. This initial setup is mitigated somewhat with C++ and its use of templates, but still involves work that may not be directly related to the application that needs to be developed.

1.3.2 Object-oriented

Object-oriented programming (OOP) adds another dimension to structured and procedural languages where data and logic are discrete elements of programming. OOP allows for associating specific behaviors, characteristics, and/or capabilities with the data that they execute on or are representative of. The object-oriented nature of Python was part of its design from the very beginning. Other OO scripting languages include SmallTalk, the original Xerox PARC language that started it all, and Netscape’s JavaScript.

1.3.3 Scalable

Python is often compared to batch or Unix shell scripting languages. Simple shell scripts handle simple tasks. They grow (indeinitely) in length, but not truly in depth. There is little code-reusability and you are confined to small projects with shell scripts. In fact, even small projects may lead to large and unwieldy scripts. Not so with Python, where you can grow your code from project to project, add other new or existing Python elements, and reuse code at your whim. Python encourages clean code design, high-level structure, and “packaging” of multiple components, all of which deliver the flexibility, consistency, and faster development time required as projects expand in breadth and scope.

The term “scalable” is most often applied to measuring hardware throughput and usually refers to additional performance when new hardware is added to a system. We would like to differentiate this comparison with ours here, which tries to inflect the notion that Python provides basic building blocks on which you can build an application, and as those needs expand and grow, Python’s pluggable and modular architecture allows your project to flourish as well as maintain manageability.
Chapter 1 Welcome to Python!

1.3.4 Extensible

As the amount of Python code increases in your project, you may still be able to organize it logically due to its dual structured and object-oriented programming environments. Or, better yet, you can separate your code into multiple files, or “modules” and be able to access one module’s code and attributes from another. And what is even better is that Python’s syntax for accessing modules is the same for all modules, whether you access one from the Python standard library or one you created just a minute ago. Using this feature, you feel like you have just “extended” the language for your own needs, and you actually have.

The most critical portions of code, perhaps those hotspots that always show up in profile analysis or areas where performance is absolutely required, are candidates for extensions as well. By “wrapping” lower-level code with Python interfaces, you can create a “compiled” module. But again, the interface is exactly the same as for pure Python modules. Access to code and objects occurs in exactly the same way without any code modification whatsoever. The only thing different about the code now is that you should notice an improvement in performance. Naturally, it all depends on your application and how resource-intensive it is. There are times where it is absolutely advantageous to convert application bottlenecks to compiled code because it will decidedly improve overall performance.

This type of extensibility in a language provides engineers with the flexibility to add-on or customize their tools to be more productive, and to develop in a shorter period of time. Although this feature is self-evident in mainstream third-generation languages (3GLs) such as C, C++, and even Java, it is rare among scripting languages. Other than Python, true extensibility in a current scripting language is readily available only in the Tool Command Language (TCL). Python extensions can be written in C and C++ for CPython and in Java for JPython.

1.3.5 Portable

Python is available on a wide variety of platforms (see Section 1.4), which contributes to its surprisingly rapid growth in today’s computing domain. Because Python is written in C, and because of C’s portability, Python is available on practically every type of system with a C compiler and general operating system interfaces.
Although there are some platform-specific modules, any general Python application written on one system will run with little or no modification on another. Portability applies across multiple architectures as well as operating systems.

### 1.3.6 Easy-to-learn

Python has relatively few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language in a relatively short period of time. There is no extra effort wasted in learning completely foreign concepts or unfamiliar keywords and syntax. What may perhaps be new to beginners is the object-oriented nature of Python. Those who are not fully-versed in the ways of object-oriented programming (OOP) may be apprehensive about jumping straight into Python, but OOP is neither necessary nor mandatory. Getting started is easy, and you can pick up OOP and use when you are ready to.

### 1.3.7 Easy-to-read

Conspicuously absent from the Python syntax are the usual symbols found in other languages for accessing variables, code block definition, and pattern-matching. These include: dollar signs ($), semicolons (;), tildes (~), etc. Without all these distractions, Python code is much more clearly defined and visible to the eyes. In addition, much to many programmers’ dismay (and relief), Python does not give as much flexibility to write obfuscated code as compared to other languages, making it easier for others to understand your code faster and vice versa. Being easy-to-read usually leads to a language’s being easy-to-learn, as we described above. We would even venture to claim that Python code is fairly understandable, even to a reader who has never seen a single line of Python before. Take a look at the examples in the next chapter, Getting Started, and let us know how well you fare.

### 1.3.8 Easy-to-maintain

Maintaining source code is part of the software development lifecycle. Your software is permanent until it is replaced or obsoleted, and in the meantime, it is more likely that your code will outlive you in your current position. Much of Python’s success is that source code is fairly easy-to-maintain, dependent, of course, on size and complexity. However, this conclusion is not difficult to draw given that Python is easy-to-learn and easy-to-read. Another motivating advantage of Python is that upon reviewing a script you wrote six months ago,
1.3.9 Robust

Nothing is more powerful than allowing a programmer to recognize error conditions and provide a software handler when such errors occur. Python provides “safe and sane” exits on errors, allowing the programmer to be in the driver’s seat. When Python exits due to fatal errors, a complete stack trace is available, providing an indication of where and how the error occurred. Python errors generate “exceptions,” and the stack trace will indicate the name and type of exception that took place. Python also provides the programmer with the ability to recognize exceptions and take appropriate action, if necessary. These “exception handlers” can be written to take specific courses of action when exceptions arise, either defusing the problem, redirecting program flow, or taking clean-up or other maintenance measures before shutting down the application gracefully. In either case, the debugging part of the development cycle is reduced considerably due to Python’s ability to help pinpoint the problem faster rather than just being on the hunt alone. Python’s robustness is beneficial for both the software designer as well as for the user. There is also some accountability when certain errors occur which are not handled properly. The stack trace which is generated as a result of an error reveals not only the type and location of the error, but also in which module the erroneous code resides.

1.3.10 Effective as a Rapid Prototyping Tool

We’ve mentioned before how Python is easy-to-learn and easy-to-read. But, you say, so is a language like BASIC. What more can Python do? Unlike self-contained and less flexible languages, Python has so many different interfaces to other systems that it is powerful enough in features and robust enough that entire systems can be prototyped completely in Python. Obviously, the same systems can be completed in traditional compiled languages, but Python’s simplicity of engineering allows us to do the same thing and still be home in time for supper. Also, numerous external libraries have already been developed for Python, so whatever your application is, someone may have traveled down that road before. All you need to do is plug-‘n’-play (some assembly required, as usual). Some of these libraries include: networking, Internet/Web/CGI, graphics and graphical user interface (GUI) development (Tkinter), imaging (PIL), numerical computation and analysis (NumPy),
1.3.11 A Memory Manager

The biggest pitfall with programming in C or C++ is that the responsibility of memory management is in the hands of the developer. Even if the application has very little to do with memory access, memory modification, and memory management, the programmer must still perform those duties, in addition to the original task at hand. This places an unnecessary burden and responsibility upon the developer and often provides an extended distraction.

Because memory management is performed by the Python interpreter, the application developer is able to steer clear of memory issues and focus on the immediate goal of just creating the application that was planned in the first place. This lead to fewer bugs, a more robust application, and shorter overall development time.

1.3.12 Interpreted and (Byte-) Compiled

Python is classified as an interpreted language, meaning that compile-time is no longer a factor during development. Traditionally purely interpreted languages are almost always slower than compiled languages because execution does not take place in a system’s native binary language. However, like Java, Python is actually byte-compiled, resulting in an intermediate form closer to machine language. This improves Python’s performance, yet allows it to retain all the advantages of interpreted languages.

CORE NOTE: File Extensions

Python source files typically end with the .py extension. The source is byte-compiled upon being loaded by the interpreter or by being byte-compiled explicitly. Depending on how you invoke the interpreter, it may leave behind byte-compiled files with a .pyc or .pyo extension. You can find out more about file extensions in Chapter 12, Modules.

1.4 Obtaining Python

As we alluded to earlier in Section 1.3.5, Python is available on a wide variety of platforms:
Chapter 1 Welcome to Python!

- Unix (Solaris, Linux, FreeBSD, AIX, HP/UX, SunOS, IRIX, et al.)
- Win 9x/NT/2000 (Windows 32-bit systems)
- Macintosh (PPC, 68K)
- OS/2
- DOS (multiple versions)
- Windows 3.x
- PalmOS
- Windows CE
- Acorn/RISC OS
- BeOS
- Amiga
- VMS/OpenVMS
- QNX
- VxWorks
- Psion

There are currently three contemporary versions of Python today. 1.5.2 is the most stable version, having been released over a year and a half ago. Python 1.6, recently made available to the public in early September 2000 introduces several major new features and improvements over the 1.5 series. However, 1.6 is seen as more of a transition to the new Python 2.0, which was released in mid-October 2000. Which version should you use? The answer is based on your needs and expectations.

If you don’t need all the fancy new features, but do desire rock solid stability, code which is backwards-compatible with the older releases (and cohabiting with existing Python installations), and is available on the greatest number of platforms, 1.5.2 is the obvious choice.

For all new projects, those without backwards dependence on older versions or Python, and those either wanting or needing to take advantage of the most crucial new features such as Unicode support, not to mention wanting to have access to the latest and greatest, cutting edge Python technology, you should start with 2.0.
1.6.0 is an alternative for those migrating from 1.5.2 to 2.0 who need a migration path, but is otherwise not recommended since it was only the most current version of Python by slightly over a month's time.

1.5 Obtaining Python

For the most up-to-date and current source code, binaries, documentation, news, etc., check either the main Python language site or the PythonLabs Web site:

http://www.python.org (community home page)

http://www.pythonlabs.com (commercial home page)

If you do not have access to the Internet readily available, all three versions (source code and binaries) are available on the CD-ROM in the back of the book. The CD-ROM also features the complete online documentation sets viewable via offline browsing or as archive files which can be installed on hard disk. All of the code samples in the book are there as well as the Online Resources appendix section (featured as the Python “hotlist”).

1.6 Installing Python

Platforms with ready-to-install binaries require only the file download and initiation of the installation application. If a binary distribution is not available for your platform, you need to obtain and compile the source code manually. This is not as bad an option as it may seem at first. Manually building your own binaries offers the most flexibility.

You can choose what features to put into your interpreter and which to leave out. The smaller your executable, the faster it will load and run. For example, on Unix systems, you may wish to install the GNU readline module. This allows you to scroll back through Python commands and use Emacs- or vi-like key bindings to scroll through, access, and perhaps edit previous commands. Other popular options include incorporating Tkinter so that you can build GUI applications or the threading library to create multi-threaded applications. All of the options we described can be added by editing the Modules/Setup file found in your source distribution.
Chapter 1 Welcome to Python!

In general, these are the steps when building your own Python interpreter:

- download and extract files, customizing build files (if applicable)
- run "./configure script"
- make
- make install

Python is usually installed in a standard location so that you can find it rather easily. On Unix machines, the executable is usually installed in "/usr/local/bin" while the libraries are in "/usr/local/lib/python1.x" where the 1.x is the version of Python you are using.

On DOS and Windows, you will usually find Python installed in C:\Python or C:\Program Files\Python. Since DOS does not support long names like “Program Files,” it is usually aliased as “Progra-1,” so if you are in a DOS window in a Windows system, you will have to use the short name to get to Python. The standard library files are typically installed in C:\Program Files\Python\Lib.

1.7 Running Python

There are three different ways to start Python. The simplest way is by starting the interpreter interactively, entering one line of Python at a time for execution. Another way to start Python is by running a script written in Python. This is accomplished by invoking the interpreter on your script application. Finally, you can run from a graphical user interface (GUI) from within an integrated development environment (IDE). IDEs typically feature additional tools such as debuggers and text editors.

1.7.1 Interactive Interpreter from the Command-line

You can enter Python and start coding right away in the interactive interpreter by starting it from the command line. You can do this from Unix, DOS, or any other system which provides you a command-line interpreter or shell window. One of the best ways to start learning Python is to run the interpreter interactively. Interactive mode is also very useful later on when you want to experiment with specific features of Python.
Unix

To access Python, you will need to type in the full pathname to its location unless you have added the directory where Python resides to your search path. Common places where Python is installed include `/usr/bin` and `/usr/local/bin`.

We recommend that you add Python (i.e., the executable file `python`, or `jpython` if you wish to use the Java version of the interpreter) to your search path because you do not want to have to type in the full pathname every time you wish to run interactively. Once this is accomplished, you can start the interpreter with just its name.

To add Python to your search path, simply check your login start-up scripts and look for a set of directories given to the `set path` or `PATH=` directive. Adding the full path to where your Python interpreter is located is all you have to do, followed by refreshing your shell’s path variable. Now at the Unix prompt (`%` or `$`, depending on your shell), you can start the interpreter just by invoking the name `python` (or `jpython`), as in the following:

```
% python
```

Once Python has started, you’ll see the interpreter startup message indicating version and platform and be given the interpreter prompt “```” to enter Python commands. Figure 1–1 is a screen shot of what Python looks like when you start it in a Unix environment:

![Figure 1–1 Starting Python in a Unix (Solaris) Window](image-url)
Chapter 1  Welcome to Python!

DOS

To add Python to your search path, you need to edit the C:\autoexec.bat file and add the full path to where your interpreter is installed. It is usually either C:\Python or C:\Program Files \Python (or its short DOS name equivalent C:\Progra~1\Python).

From a DOS window (either really running in DOS or started from Windows), the command to start Python is the same as Unix, python. The only difference is the prompt, which is C:\>.

C:\> python

Command-line Options

When starting Python from the command-line, additional options may be provided to the interpreter. Here are some of the options to choose from:

- `-d` provide debug output
- `-O` generate optimized bytecode (resulting in .pyo files)
- `-S` do not run import site to look for Python paths on startup
- `-v` verbose output (detailed trace on import statements)
- `-X` disable class-based built-in exceptions (just use strings); obsolete starting with version 1.6
- `-c cmd` run Python script sent in as cmd string
- `file` run Python script from given file (see below)
1.7.2 As a Script from the Command-line

From Unix, DOS, or any other version with a command-line interface, a Python script can be executed by invoking the interpreter on your application, as in the following:

C:\> python script.py
unix% python script.py

Most Python scripts end with a file extension of .py, as indicated above.

It is also possible in Unix to automatically launch the Python interpreter without explicitly invoking it from the command-line. If you are using any Unix-flavored system, you can use the shell-launching ("sh-bang") first line of your program:

```
#!/usr/local/bin/python
```

The "file path," i.e., the part that follows the "#!", is the full path location of the Python interpreter. As we mentioned before, it is usually installed in /usr/local/bin or /usr/bin. If not, be sure to get the exact pathname correct so that you can run your Python scripts. Pathnames that are not correct will result in the familiar "Command not found" error message.

As a preferred alternative, many Unix systems have a command named env, either installed in /bin or /usr/bin, that will look for the Python interpreter in your path. If you have env, your startup line can be changed to something like this:

```
#!/usr/bin/env python
```

env is useful when you either do not know exactly where the Python executable is located, or if it changes location often, yet still remains available via your directory path. Once you add the proper startup directive to the beginning of your script, it becomes directly executable, and when invoked, loads the Python interpreter first, then runs your script. As we mentioned before, Python no longer has to be invoked explicitly from the command. You only need the script name:

```
unix% script.py
```

Be sure the file permission mode allows execution first. There should be an "rwx" flag for the user in the long listing of your file. Check with your system administrator if you require help in finding where Python is installed or if you need help with file permissions or the chmod (CHange MODe) command.
Chapter 1 Welcome to Python!

DOS does not support the auto-launching mechanism; however, Windows does provide a “file type” interface. This interface allows Windows to recognize file types based on extension names and to invoke a program to “handle” files of predetermined types. For example, if you install Python with PythonWin (see below), double-clicking on a Python script with the .py extension will invoke Python or PythonWin IDE (if you have it installed) to run your script.

1.7.3 In an Integrated Development Environment

You can run Python from a graphical user interface (GUI) environment as well. All you need is a GUI application on your system that supports Python. If you have found one, chances are that it is also an IDE (integrated development environment). IDEs are more than just graphical interfaces. They typically have source code editors and trace and debugging facilities.

Unix

IDLE is the very first Unix IDE for Python. It was also developed by Guido and made its debut in Python 1.5.2. IDLE either stands for IDE with a raised “L,” as in Integrated DeveLopment Environment. Suspiciously, IDLE also happens to be the name of a Monty Python troupe member. Hmmm.... IDLE is Tkinter-based, thus requiring you to have Tcl/Tk installed on your system. Current versions of Python include a distributed minimal subset of the Tcl/Tk library so that a full install is no longer required.

You will find the idle executable in the Tools subdirectory with the source distribution. The Tk toolkit also exists on Windows, so IDLE is also available on that platform and on the Macintosh as well. A screen shot of IDLE in Unix appears in Figure 1–3.

Windows

PythonWin is the first Windows interface for Python and is an IDE with a GUI. Included with the PythonWin distribution are a Windows API, COM (Component Object Model, a.k.a. OLE [Object Linking and Embedding] and ActiveX) extensions. PythonWin itself was written to the MFC
Running Python

PythonWin is usually installed in the same directory as Python, in its own subdirectory, C:\Program Files\Python\Pythonwin as the executable pythonwin.exe. PythonWin features a color editor, a new and improved debugger, interactive shell window, COM extensions, and more. A screen snapshot of the PythonWin IDE running on a Windows machine appears in Figure 1–4.

More documentation from the installed software can be found by firing up your web browser and pointing it to the following location (or wherever your PythonWin is installed):

file://C:/Program Files/Python/Pythonwin/readme.html

As we mentioned before, IDLE is also available on the Windows platform, due to the portability of Tcl/Tk and Python/Tkinter. It looks similar to its Unix counterpart (Figure 1–5).

From Windows, IDLE can be found in the Tools\idle subdirectory of where your Python interpreter is found, usually C:\Program Files \Python\Tools\idle. To start IDLE from a DOS window, invoke idle.py. You can also invoke idle.py from a Windows environment, but that starts an unnecessary DOS window. Instead, double-click on idle.pyw.
Chapter 1  Welcome to Python!

Figure 1–4 PythonWin Environment in Windows

Figure 1–5 Starting IDLE in Windows
Macintosh

The Macintosh effort of Python is called MacPython and also available from the main website, downloadable as either MacBinary or BinHex'd files. Python source code is available as a Stuff-It archive. This distribution contains all the software you need to run Python on either the PowerPC or Motorola 68K architectures. MacPython includes an IDE, the numerical Python (NumPy) module, and various graphics modules, and the Tk windowing tool-kit comes with the package, so IDLE will work on the Mac as well. Figure 1–6 shows what the MacPython environment looks like. Presented in the figure below are a text window open to edit a Python script as well as a Python "shell" running the interpreter:

Figure 1–6 Running the IDE in MacPython
1.8 Python Documentation

Most of the documentation that you need with Python can be found on the CD-ROM or the main website. Documentation is available for download in printable format or as hypertext HTML files for online (or offline) viewing.

If you download the Windows version of Python, the HTML documentation comes with the distribution as an install option. Be sure to leave the “Help Files” box checked if you would like to install the HTML files in your Python directory. Once the installation is complete, you may then access the Python documentation through your web browser by pointing to the link below or wherever your interpreter is installed:

file://C:/Program Files/Python/Doc/index.html

Also see the Appendix for an exhaustive list of both printed and online documentation for Python.

1.9 Comparing Python

Python has been compared with many languages. One reason is that it provides many features found in other languages. Another reason is that Python itself is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell and other scripting languages, to name a few. Python is a virtual “greatest hits” where van Rossum combined the features he admired most in the other languages he had studied and brought them together for our programming sanity.

However, more often than not, since Python is an interpreted language, you will find that most of the comparisons are with Perl, Java, Tel, and JavaScript. Perl is another scripting language which goes well beyond the realm of the standard shell scripts. Like Python, Perl gives you the power of a full programming language as well as system call access.

Perl’s greatest strength is in its string pattern matching ability, providing an extremely powerful regular expression matching engine. This has pushed Perl to become the de facto language for string text stream filtering, recognition, and extraction, and it is still the most popular language for developing Internet applications through web servers’ Common Gateway Interface (CGI). However, Perl’s obscure and overly-symbolic syntax is much more difficult to decipher, resulting in a steep learning curve that inhibits the beginner, frustrating those for whom grasping concepts is impeded by semantics. This, coupled with Perl’s “feature” of providing many ways of accomplishing
Comparing Python

the same task, introduces inconsistency and factionization of developers. Finally, all too often the reference book is required reading to decipher a Perl script which was written just a mere quarter ago.

Python is often compared to Java because of their similar object-oriented nature and syntax. Java’s syntax, although much simpler than C++’s, can still be fairly cumbersome, especially if you want to perform just a small task. Python’s simplicity offers a much more rapid development environment that using just pure Java. One major evolution in Python’s relationship with Java is the development of JPython, a Python interpreter written completely in Java. It is now possible to run Python programs with only the presence of a Java VM (virtual machine). We will mention more of JPython’s advantages briefly in the following section, but for now we can tell you that in the JPython scripting environment, you can manipulate Java objects, Java can interact with Python objects, and you have access to your normal Java class libraries as if Java has always been part of the Python environment.

Tcl is another scripting language that bears some similarities to Python. Tcl is one of the first truly easy-to-use scripting languages providing the programmer extensibility as well as system call access. Tcl is still popular today and perhaps somewhat more restrictive (due to its limited types) than Python, but it shares Python’s ability to extend past its original design. More importantly, Tcl is often used with its graphical toolkit partner, Tk, in developing graphical user interface (GUI) applications. Due to its popularity, Tk has been ported to Perl (Perl/Tk) and Python (Tkinter).

Python has some light functional programming (FP) constructs which likens it to languages such as Lisp or Scheme. However, it should be noted that Python is not considered an FP language; therefore, it does provide much more than what you see.

Of all the languages most often compared to Python, JavaScript bears the most resemblance. It is the most similar syntax-wise as well as also being object-oriented. Any proficient JavaScript programmer will find that picking up Python requires little or no effort. Python provides execution outside the web browser environment as well as the ability to interact with system calls and perform general system tasks commonly handled by shell scripts.

You can access a number of comparisons between Python and other languages at:

http://www.python.org/doc/Comparisons.html
1.10 JPython and Some Nomenclature

As we mentioned in the previous section, a Python interpreter completely (re)written in Java called JPython is currently available. Although there are still minor differences between both interpreters, they are very similar and provide a comparable startup environment.

What are the advantages of JPython? JPython...

- Can run (almost) anywhere a Java Virtual Machine (JVM) can be found
- Provides access to Java packages and class libraries
- Furnishes a scripting environment for Java development
- Enables ease-of-testing for Java class libraries
- Matches object-oriented programming environments
- Delivers JavaBeans property and introspection ability
- Encourages Python-to-Java development (and vice versa)
- Gives GUI developers access to Java AWT/Swing libraries
- Utilizes Java's native garbage collector (so CPython's was not implemented)

A full treatment of JPython is beyond the scope of this text, but there is a good amount of information online. JPython is still an ongoing development project, so keep an eye out for new features.

1.11 Exercises

1–1. **Installing Python.** Download the Python software or load it from the CD-ROM, and install it on your system.

1–2. **Executing Python.** How many different ways are there to run Python?

1–3. **Python Standard Library.**
   (a) Find where the Python executables and standard library modules are installed on your system.
   (b) Take a look at some of the standard library files, for example, `string.py`. It will help you get acclimated to looking at Python scripts.

1–4. **Interactive Execution.** Start the Python interactive interpreter. You can invoke it by typing in its full pathname or just
its name (python or python.exe) if you have
installed its location in your search path. (You can also
use the Python interpreter compiled in Java [jpython or
jpython.exe] if you wish.) The startup screen should look
like the ones depicted in this chapter. When you see the
“>>>”, that means the interpreter is ready to accept your
Python commands.
Try entering the command for the famous Hello World! pro-
gram by typing print “Hello World!”, then exit the inter-
preter. On Unix systems, Ctrl-D will send the EOF signal to
terminate the Python interpreter, and on DOS systems, the
keypress is Ctrl-Z. Exiting from windows in graphical user
environments like the Macintosh, PythonWin or IDLE on
Windows, or IDLE on Unix can be accomplished by simply
closing their respective windows.
1–5. Scripting. As a follow-up to Exercise 1–4, create “Hello
World!” as a Python script that does exactly the same thing as
the interactive exercise above. If you are using the Unix sys-
tem, try setting up the automatic startup line so that you can
run the program without invoking the Python interpreter.
1–6. Scripting. Create a script that displays your name, age, favor-
ite color, and a bit about you (background, interests, hobbies,
etc.) to the screen using the print statement.