

The
Complete
Reference



Chapter 1

Introducing Python

Before getting into the specifics of the Python language, let's take a little time to understand the principles behind the Python language and what it can and can't be used for. Let's also look at who uses Python and how Python differs from other programming languages.

What Is Python?

Python is an interpreted language that employs an object-oriented approach. It's a high-level programming language, which means that it separates the user from the underlying operating system as much as possible. However, unlike other languages, Python provides you with the ability to access the operating system at a lower level if you desire. Because of this ability, Python is often classified somewhere between such languages as Visual Basic or Perl and the system-level C language.

Although Python is considered an interpreted language like Perl, Tcl, and some others, it employs a compilation stage that translates the raw-text Python script into a series of *bytecodes*, which are then executed by the Python Virtual Machine. The use of the compilation and bytecode stages helps to improve performance and makes Python much faster than pure interpreters such as BASIC, but slower than the truly compiled languages such as C and Pascal. However, unlike many other languages, the bytecode versions of modules can be saved and executed without having to recompile them each time they are required, thereby improving performance by eliminating the compilation stage. Note that the bytecode that is created is completely platform and operating system independent, much like the bytecode produced by Java.

Python also has the added benefit of providing rapid application development on the MacOS, Windows (95/98/NT), and Unix platforms. Python is supplied with a module to the Tk interface libraries, and it's possible to write an application on one platform and use it on all three platforms without making any modifications. In addition to the core platforms, Python also runs on MS-DOS, Amiga, BeOS, OS/2, VMS, QNX, and many other operating systems. You can even run Python on your Psion organizer!

Before going any further I should probably explain the name. The name Python is taken from the comedy group Monty Python, which is best known for the talents of Eric Idle, John Cleese, Terry Jones, Terry Gilliam, Michael Palin, and Graham Chapman.

Python Is Free

Although this is not that unique—many of the most popular programming languages are available free—it does mean that you can write and deploy Python programs without having to purchase any software and without having to worry about licensing issues. You can even download the source code to the software if you want to take a closer look at how the Python language works.

Being free often means that there is little or no support. With Python this is not the case. There is a huge following of Python programmers, and the people involved in developing Python are always welcome to help new users get to know the language. There are also a number of commercial companies and individuals who provide custom programming and more in-depth support if you need it and are willing to pay for it.

Python Is Portable

Python is supported on a huge range of operating system platforms. It comes in ready-compiled format for Windows and MacOS and includes the Tk extensions so you can develop user interfaces. On Unix and all other platforms, Python is available as source code which you compile yourself. You can also visit one of the web sites offering precompiled binaries (see Appendix A). In all cases, the compatibility is invisible. You can write a script on a Unix platform, and 95% of the time you can execute it without modification on a Mac or a PC. Because of the Tk support, you can even run the same GUI-based application on all three platforms without any major modifications and have a consistent user interface in each case.

In addition to the native cross-platform capability, Python also supports some platform-native extensions to help ease the porting process or bridge gaps with other languages and environments. For example, the SunOS/Solaris implementation includes a driver for Sun audio devices, and the SGI version comes with tools for interfacing to the audio and video capabilities (including OpenGL) built into SGI workstations. The Windows Python interpreter comes with toolkits to interface to the Visual C++ libraries and the Windows audio drivers. You can even communicate with COM (Component Object Model) objects.

Python Is Powerful

There is very little you cannot do with Python. The core of the language is very small, but it provides enough of the basic building blocks to allow you to design most applications. Furthermore, because the language can be extended using C, C++, and even Java in certain circumstances, you should be able to develop any type of program. The Python interpreter actually comes with a huge library of additional modules that extend the capabilities of the language to allow network communication, text processing (including extensive XML support in Python 2.0), and regular expression matching.

Although Python's main objective is to hide much of the low-level complexity from the programmer, it also supports the necessary hooks, extensions, and functions to allow low-level access to certain areas of the operating system. By supporting both the high-level and low-level functionality, Python can be used at the same level as C or at the same level as Visual Basic, as well as at all the other levels in between. You can even use Python as a macro or application extension language by embedding it into

your applications, much like Visual Basic is the macro language used in Microsoft Office products.

Python Is Extensible

Because Python is written in C (some extensions are written in C++) and because you have access to the source code, you can also write extensions to the language. Many of the standard modules supplied with the language are supported by a C or C++ interface. This includes basic facilities such as networking and DBM database access, along with more advanced toolkits such as Tk.

In addition, Python can be embedded into C or C++ applications so you can provide a scripting interface to your application using the Python language. Because of the support for cross-language development, you can use Python to design and conceptualize an application and then port it, over time, to C. There is no need to rewrite the application in C before using it; Python and C can work together in tandem.

Finally, Python is a complete Python interpreter written entirely in Java. This means that you can write a Python program that interfaces to Java objects, or you can write a Java application that uses Python objects. Better still, because the interpreter is written entirely in Java, you can deploy a Python application on any platform that supports Java—even a web browser can directly execute a Python script.

Python Is Easy

Once you understand the basic principles of the Python language, learning the rest is easy. The core of the language is very small, and its semantics and style are very simple. Since all the other components and extensions use exactly the same syntax and structure, you should be up to speed programming in Python like an expert in no time at all.

That's not to say that there aren't complexities in the Python language that you'll need to learn. Many of the extensions and libraries require some careful thought to make effective use of the language. This is especially true when you start to integrate and interface to other languages such as Tk and SQL.

What Is Python Good For?

Not surprisingly, with such wide support and extensive features, Python is very effective for a large number of tasks. Here's a quick list of the more common uses of the Python language.

Mathematics

Python supports an extension called NumPy, which provides interfaces to many standard mathematics libraries. The Python language also supports unlimited

precision. If you want to add two 100-digit numbers together, you can do so with Python without requiring a third-party extension. If you need real speed in mathematical applications, the NumPy extension is written in C and as a result, operates faster than the native math supported by Python.

Text Processing

Python can split, separate, summarize, and report on any data. It comes with modules that separate out the elements of a log file line. You can then use the built-in data types to record and summarize the information before writing it all out again. In addition, Python comes with expression libraries that allow you to use the same expressions as Emacs, Perl, and many other utilities. This means that Python can do all of the things that other languages can do. For example, a number of programmers have produced a complex SGML-processing tool using Python.

Python actually comes with SGML-, HTML-, and XML-parsing modules for reading, writing, and translating the different formats. With Python's support for other text-processing engines (regular expressions and the natural splitting/combining of information) and flexible variable and object handling, Python becomes a very useful tool for the text-processing programmer.

Rapid Application Development

Because Python is so straightforward to develop applications with, you can use Python to develop applications very quickly. The extensive module library that comes with Python provides direct interfaces to many of the protocols, tools, and libraries that you would otherwise have to develop yourself.

Furthermore, because scripts are interpreted and don't have to go through the normal compilation and linking stages, you can make modifications very quickly. If there's something that you need to test quickly, you can easily check it with the interpreter shell before actually making changes to your program, and you can even interactively debug your script using the same shell and environment that you use for programming.

Also, because Python supports Tk, you can have not just an interface example, but a complete application in hours instead of days. I've written a cross-platform, Tk-based database query tool in less than day, something that would have taken me two or three days with Perl, and a week or more in C.

Cross-Platform Development

You now know that Python supports a wide range of platforms in a completely neutral format. If you are deploying an application across a network that uses a variety of different platforms, you can use Python to develop the application. You can also use Python where you want to be sure that future implementations of your system are going to work. Many companies start with a specific platform and then move to a

different platform as the performance starts to suffer. Using Python, you will never need to rewrite your software as you move between platforms.

Of course, you can also look at Python as an alternative when supplying software to end users. Instead of developing three separate applications, with operating system-specific compilation processes, testing systems, and interfaces, you only have to develop it once, saving significant time and money.

System Utilities

Although much of the ethos with Python is to hide from you the low-level parts of the operating system, the tools and extensions are there if you want to access the lowest levels. Because Python has access to the same set of functions as the operating system, you can use it to duplicate and extend the functionality of the operating system, while still retaining all of the compatibility and interface issues that you already know Python supports.

Internet Programming

Python comes with a standard set of modules that allow you to communicate over the network sockets, both at a basic level and at a protocol level. If you want to read e-mail from a POP server, for example, Python already comes with the library module that enables you to do that. In addition, Python also supports XML, HTML, and CGI libraries so you can parse user input and produce top-quality formatted output via a web server.

In fact, the combination of Python's high-level module support and RAD powers gives you an enormous, but speedy, development toolkit. Most Internet modules allow you to communicate directly with an Internet server using a simple object class and a number of different methods. One programmer even managed to write a newsreader entirely in Python within a couple of afternoons. On my home network, I have a Web-based e-mail interface to my IMAP server that took me about an hour to write in Python.

You can even compile a module for Apache, the Unix and Windows web server that embeds the Python interpreter. This means that when you want to execute a Python script, the interpreter does not need to be loaded separately each time, thus providing the maximum possible performance from your CGI scripts.

Database Programming

There are a myriad of extension modules that interface to all of the common database systems, from Oracle to Informix and free systems such as mSQL and MySQL. There is even a toolkit called Gadfly that provides a complete SQL environment within Python—no external modules or extensions are required. Because Python has strong text- and data-handling abilities, you can use Python to interface between databases

and to act as a better summary and report tool than many of the interfaces that come with the database systems themselves. Furthermore, because Python supports a number of different operating systems, instead of only one, you can use the same interface with any database. You can even use Tk to build the front end and then put it on any of the supported platforms—you'll get an instant cross-platform, database independent query tool!

Everything Else

Python can be used for anything—there are literally no limits to what this language can do. By supporting a small core set of functions, data types, and capabilities, Python provides an excellent base on which to build. Because you can extend the functionality with C and C++, you get the best of both worlds—unlimited and unfettered expansion to do whatever you want, but in a structured and manageable format.

What Isn't Python Good For?

It's very difficult to give a precise list of the problems that Python is unable to solve. Python provides most of its functionality in the extension modules that are supplied with the language, and this just demonstrates how easy it is to add functionality to the language. If you can't do what you want to within Python, then it's just as easy to write a C or C++ extension to do the job for you.

Some people criticize Python not because it's not capable of doing a particular task, but because they don't understand *how* to do a particular task. One of the most common complaints relates to Python's apparent lack of regular expression support, when in fact there are two modules (**re** and the older **regex**) that enable you to handle regular expressions; **regex** even supports the same syntax that is used in Perl. Regular expression handling may not be built into Python, and it probably isn't as fast as Perl's expression handling, which has been highly optimized over the years, but it's still possible.

The advantage of Python over a language such as Perl, Rebol, or Java is that the core of the language is very small. This improves the execution time—there's less code to be loaded each time the script is run—and helps make the rest of the language easier to learn and more flexible.

Once you are familiar with the minimalist style of programming that Python supports, you'll find that you still have all the power, but without the extra baggage. And you can read your code the next day, week, or year you look at it.

Who Uses Python?

Python is used by a large number of people for solving all sorts of tasks. Most of these are not well known, or at least not publicized, purely because the companies concerned do not normally divulge this sort of information. However, there are some

larger companies that use Python within a commercial environment and that are proud to announce and even celebrate the fact.

- Red Hat (www.redhat.com), who makes the popular Red Hat Linux distribution, uses Python in combination with Tk to provide a visual interface for configuring and managing the Linux operating system. The configuration system gives you complete control over all aspects of the Linux operating system and automatically updates the configuration files according to your selections.
- Infoseek (www.infoseek.com) uses Python within certain parts of its public search engines. Python is also used for customizing the Infoseek software that can be downloaded from this site for use on end users' machines.
- NASA (www.nasa.gov) uses Python in a number of different areas. The most significant use of Python is in the Mission Control Center, where Python is used in certain parts of the system for planning missions. Other uses take advantage of the strong numerical abilities of Python, which make it ideal for calculating the location of celestial objects and for plotting the paths taken by satellites.
- Industrial Light and Magic (www.ilm.com), famous for doing the special effects on films such as *Star Wars*, *The Abyss*, *Star Trek*, and *Indiana Jones*, uses Python to produce commercial-grade animation. In fact, if you visit their web site, you'll see they have a number of vacancies for Python programmers!

Python History

Python has a long history—noticeably longer than most people probably realize. In the year 2000, the Python development efforts underwent a great deal of reorganization as Guido van Rossum, the designer and primary developer of Python, and the rest of the Python team (including Tim Peters, Barry Warsaw, Jeremy Hylton, and Fred Drake) moved the development efforts first from CNRI (Centre for National Research Initiatives) to BeOpen, and ultimately, to Digital Creations.

Python Up to 1.5.2

Up until September 5, 2000, Python was developed and released through a public license supported by CNRI. Python 1.5.2 contained most of the functionality of Python that we are familiar with today. The final 1.5.2 release was released on April 13, 1999.

Python 1.6 (September 2000)

In September 2000, after Guido left CNRI, two versions of Python 1.6 were released. The first version came from CNRI and was their last official release of the Python language. Python 1.6b1, the beta version of the language, had been in development

and testing for some time, so it was not a complete surprise to see Python 1.6b1 released by Guido van Rossum and the BeOpen development team.

Version 1.6 included some minor improvements, including a change in the way list objects worked and some improvements in the socket and string-to-number conversion tools.

Python 2.0 (September 5, 2000)

Within 24 hours of CNRI's announcement, Guido and the rest of the development team released version 2.0b1 (a beta release) of Python to steal a coup from CNRI's 1.6 release. The entire Python team had moved from CNRI to the BeOpen initiative (www.beopen.com) on May 5, 2000 to continue the development of Python outside of CNRI and under an open-source agreement.

Version 2.0 included some important updates to Version 1.6, including new operators, new list syntax, and better module-importing methods. Version 2.0 also included one of the most significant updates to the standard Python library for more than a year, fixing numerous bugs and adding several new features and an entirely rewritten suite of XML tools. The final version of Python 2.0 was released on October 16, 2000.

Aside from the obvious language improvements, the move to BeOpen also enabled the development team to make other improvements for Python behind the scenes, such as moving the Python sources to SourceForge (www.sourceforge.net), and executing the final stages for moving JPython, the old Java-based Python interpreter, to the new Jython hybrid.

Python 2.0 (October 28, 2000)

On October 28, 2000, Guido announced on the newsgroups and the main Python web site that the Python team had moved again, this time to Digital Creations. Digital Creations are the people behind Zope (Z-Objects Publishing Environment), one of the best-known Python projects.

Python 3000

No further releases have come out since the Python 2.0 release from BeOpen, but work is planned for the new Python 3000 product, due to be released sometime in 2002. Hopefully, given Digital Creations' existing Python evangelism, there shouldn't be any further moves for the Python team.

Python 3000 itself is expected to be a major update of the language, in much the same way that Perl 6.0 is expected to be a major update to Perl 5.6. Python 3000 is unlikely to be the final name of the new version of Python; it's just a code name for the new version.

Similar Languages

The range of scripting languages that are available today is growing at a rapid rate. Although most people have heard of Perl, VBScript, and almost certainly BASIC, there are other useful scripting languages that can be used to solve a number of problems. But where does Python fit in, and how does it compare to these other languages?

The question is even more complex if you try to enforce usage environments onto the programming languages. Perl is great at text processing, so is Python, and so is Awk, but which should you choose? I tend to use Perl when it's simple text processing from a file, Python when I'm converting, translating, or otherwise encapsulating data into a new data-storage format, and Awk when I need to filter text from within a shell.

Perl

Perl and Python are the two most similar languages available at the moment. They are the two top interpreted scripting languages available, even though they have different syntax and were designed with different goals in mind.

Perl was developed by Larry Wall, originally as a tool for processing text suitable for summarizing and reporting; the word Perl stands for Practical Extraction and Report Language. Unlike Python, Perl was built using many of the principles and semantics from other languages and from the Unix environment. You can integrate the commands you use within the Unix shell with other advanced features such as regular expressions, and then bond it all together with some very flexible, but powerful variables.

However, Perl also has some difficulties. The object-orientation (OO) feature is easy to use, but it was bolted on after the language was originally developed. This doesn't cause any problems, but it makes object orientation an added rather than an integrated feature. The lack of a cohesive OO element gives you the flexibility to choose whether to write OO code or not. Unfortunately, this has the effect of making programming more difficult because you have a combination of object and nonobject entities to deal with, even during the development of a simple program.

When comparing most features, the features offered by Python and Perl are difficult to differentiate. The main difference is that the ethos is different. Perl was designed and is still largely used as a system for processing text and binary data. These tasks encompass most of the tasks for which Perl is most famous, including its use within the Internet for handling CGI (Common Gateway Interface) scripts. There are other strengths: Perl has very good system-level support, which makes it ideal for replicating and improving on many of the core system functions under Unix. But Perl's roots lie within the realm of text processing.

Python, on the other hand, was developed as an application programming language. With Python, you can develop full-blown, cross-platform applications for everything from utilities to complete integrated environments for application development.

When it comes to performance, there is probably little to choose between them. Both languages use the same system of compiling raw source into a bytecode, which is then

executed by a virtual machine. However, if you try to compare semantics and other language features, there are undoubtedly some differences. Python has incredibly flexible data handling, and because all data-storage containers are objects, the same methods are used to process most pieces of information, which improves performance.

Perl is somewhat limited when it comes to built-in data types, and objects are handled using references that can take time to access and process and are difficult to learn. However, Perl's text processing and regular expression parsing capabilities are the fastest available, surpassed only by Gawk.

Python has a smaller built-in function set than Perl, but it has a much larger library of supplied extensions that resolve most of the so-called "missing" elements (see the section "What Isn't Python Good For" earlier in this chapter). The object orientation is Python's biggest feature, and because it applies everywhere, including the extension modules, it allows for a greater level of flexibility and extensibility. This is particularly true if you want to use Python within a fully object-oriented environment and take advantage of the OO features, especially inheritance and polymorphism.

The final part of the comparison is less obvious immediately; Python is clean and easy to read. Perl can be confusing, with many elements of the language using a complex set of punctuation rather than textual characters. Although you can make certain aspects of Perl easier to read, you still have to contend with elements like variable prefixes and references that make heavy use of braces. As if that weren't enough, you have a number of magic variables, assumed and implied treatments of that information, and various standard and nonstandard shortcuts that can make debugging Perl a little more challenging. In short, a lot of Perl can look like line noise.

Java

Unlike Python and Perl, which are quasi-interpreted languages, Java is a true compiler-based language: Java source code must be compiled into bytecode before it is executed by a Java Virtual Machine (JVM) on a host computer. Although Python compiles programs into bytecode, the compilation stage is handled by the interpreter, so you can still execute a Python program directly from the raw-text source code.

Because the compiled Java scripts have a much tighter bytecode format, Java scripts generally run slightly faster than Python scripts. However, they take longer to develop because of the increased time required to compile the Java application into its bytecode format before it can be executed. With Python, you execute the script directly, but with Java, you need to compile the script before executing it.

Better still, you can use Python directly. You can enter Python statements interactively to the Python interpreter. You can test statements and sequences to ensure that they work correctly, and you can verify these as you go. Often, this means that Python programs work the first time because you checked each statement in the interpreter as you added it to the source file.

The true difference, though, is in the philosophy of the language. Java is very much a low-level language, at the same practical level as C and C++, from which it inherits

much of its design and methodologies. Python, on the other hand, is a high-level language. This helps hide the complexity of the underlying operating system from the programmer, while still providing all of the tools and features a programmer needs to complete the task at hand. Python is therefore far more suited to rapid development of an application within a structured framework.

Python is an excellent prototyping language for any object-oriented task, so it can be used for rough development before the design and implementation is properly developed in Java. Furthermore, Python and Java can be used together; there is even a Python interpreter (Jython) that has been written entirely in Java. Jython allows Java programs to use Python objects and Python programs to use Java objects. You can even use Python bytecode modules that were compiled with the C-based interpreter in Jython without conversion.

The issue of whether Python and a web or Tk interface is better than Java is a matter of personal opinion. I prefer Tk to Java because it's cleaner, clearer, and frequently easier to program, especially for small jobs. Tk's OO style fits well into the Python language. Tk is, in my opinion, easier to customize when you want to develop a new interface widget. Python has this advantage because it has true cross-platform compatibility, something that Java is still lacking at a production level. On the other hand, Java is currently supported on more platforms than Tk, which currently only works with Unix, Windows, and MacOS. In those situations, Java is obviously a better solution.

JavaScript

JavaScript and Python are similar in their design, but they have very different goals. Both are object-based but both also allow you to use the functions and capabilities of each language without requiring you to use the object features. However, JavaScript is not really a true scripting language, and by its very design, it isn't a competitor to Python.

The JavaScript language is an embedded language for controlling the interaction between the user and a web page. Because JavaScript is stored as part of the HTML text that is transferred when a page is accessed, there is no communication required between the user's browser and the web server as there is with CGI-based interfaces. However, JavaScript is not a stand-alone language, and outside of a web page, JavaScript is unusable. Furthermore, despite assurances to the contrary, JavaScript is not really cross-platform compliant or secure.

Programmers who already know how to program in JavaScript will feel comfortable with the Python environment because the terminology and many of the constructs are identical. The program layout and semantics are also very similar, and once you have learned the principles of JavaScript, Python is easy to migrate to. Python also provides JavaScript programmers with a much more open environment since it's possible to expand on the capabilities of the Python language—something not possible at all with JavaScript.

Tcl

Tcl was developed as a macro language to be used for extending existing applications, although it has also matured over the years into a stand-alone programming language. Most people are exposed to Tcl when they want to develop applications using the Tk interface builder. Tk is a system that provides a consistent and powerful interface for creating window-based applications. Tk is cross-platform, supporting the X Windows System on Unix and other operating systems that support X Windows, such as QNX, and the native environments under Windows 95/98/NT and MacOS. However, the Tk system is limited to those three main platforms, so while Tk is a cross-platform solution, you can't use it under lesser-used operating systems such as BeOS.

Tcl's disadvantage is that it is very weak on data structures. Languages like Perl provide much better features, and Python's strength over all the other languages discussed in this chapter is its data structuring and manipulation. Recent advances in Tcl, particularly with the version 8.1.1 release, have improved the support for data structures and provided a byte compiler to help improve the performance of Tcl applications. But in comparison to Python, Tcl still feels clumsy and slow. Since it has a good interface to Tk, Python provides an easier to use and faster solution for developing applications.

Rebol

Rebol is a very new language. Like Python, Rebol was developed with object orientation in mind. Rebol was designed as a message-style language, focusing on supporting the use of messages to transfer and communicate information. This makes Rebol ideal for Internet-based services, since Rebol can natively talk to e-mail, Usenet, web, and FTP servers, among many others. Because it instinctively knows how to access and handle many of these services, Rebol can be used to develop highly complex network-aware applications without much of the baggage normally required by languages such as Perl and Python.

Because Rebol is so highly focused, it's not entirely suited to the more general programming tasks normally applied with Perl or Python, but that functionality may come in time as the language matures.

Visual Basic

Visual Basic (VB) is the development environment offered by Microsoft as an extension to the original BASIC language that shipped with the early versions of DOS. Like JavaScript, VB has a specific target market and as such, is limited in a number of areas compared to proper programming languages such as Python.

Although it is often pitched as a general-purpose programming language, in fact, Visual Basic is targeted squarely at the development of interfaces to databases.

Although you can program complex environments for your databases within applications such as Microsoft Access or Visual FoxPro, VB is limited when it comes to developing a complete customizable interface to a database.

What VB provides is a completely customizable language that provides direct hooks into accessing data from ODBC (Open Database Connectivity) compliant databases. With VB, you have access to some of the same constructs so you can make decisions about what to display or what has been entered, but at the end of the day, you are still working with just a user-interface language.

Python has a number of extensions that provide you with access to a number of different database systems, including Oracle, Informix, mSQL/mySQL, and of course, ODBC. Because you can also program other elements with Python, you can do much more than just display the information to the user or get the information back—you can also communicate that data over the network or embed it in a web page. Perhaps more usefully, because Python supports so many different database systems, you can use Python to translate and transfer data from one system to another.

Awk/Gawk

Awk is a very old programming language; it was supplied with the early versions of the Unix operating system. Although it has its proponents (myself included), Awk and its GNU cousin Gawk are highly specialized languages for processing text. The features built into Awk and Gawk go far beyond those supported even by Perl. For summarizing and reporting on large volumes of data, Gawk far exceeds the performance of Python or Perl, and even in some instances, C.

The limitations of Gawk, however, are not restricted to text processing. Recent versions of Gawk include limited communication facilities, although it still doesn't have proper network socket facilities, and Gawk still lacks any serious file-management capabilities.

Gawk's main advantage over Python or Perl is that it is easy to write a quick script to process the output from a command within a Unix shell. In a line or two of Gawk code, you can achieve the same effect as a few lines of Python script or a large number of pipes through **cut**, **paste**, and **expr** within a Unix shell.

Awk still comes standard with most Unix distributions. Although many modern Linux distributions also tend to include Perl and Python, most of the commercial distributions do not. Python is compatible on all of them, but it still needs to be sourced, compiled, and installed.

Finally, Awk's regular expression syntax is much more advanced than either Perl's or Python's. You can use the regular expression syntax in both the search and replacement sections of substitution expressions. You can even do nested searches directly within a regular expression, something that would require a loop in most other languages.

C/C++

C is the source language for most operating systems, and it's also the language used to develop many new languages, including most of the other scripting languages mentioned in this chapter. The reason is very simple: C is a good low-level programming language that provides such fundamental access to most parts of the operating system that it can easily be designed to emulate and provide the functions and facilities required to develop other languages. Python itself is written in C, and this is one of the features that makes Python so cross-platform compatible—C is the system-level language for most operating systems.

However, Python offers much more than the facilities offered by C or C's object-oriented cousin, C++. First of all, Python is a much higher level language, hiding some of the low-level complexity of C within its own range of functions and structures. That doesn't mean you can't use Python for low-level programming; the facilities are there if you want them, but they are not required to write programs in Python.

Both C and C++ are complex languages that use a range of different terminology and constructs to support the functions and capabilities of the language. In particular, the object-oriented features of C++ are particularly complex, and they enforce a programming style that is very regimented. It's not possible in most instances to easily mix and match C++ and C functions into the same program; if you develop a C++ application, it needs to be completely C++ compliant. With Python, you can choose to use object orientation where you want to and ignore what you don't require. Furthermore, you can't take a C++ program from one platform and move it entirely to a new one; both platforms have to be using compatible compilers and run-time libraries that are simply not standard across platforms.

However, with all that in mind, it's worth remembering that Python is still written in C, and all of the functionality of the Python language relies on C source code somewhere. This reliance on C is not a limitation; in fact, you can develop extensions to the Python language using C or C++. You can also use the Python language within C programs, which enables you to embed Python statements. The embedding feature is particularly useful for programs where you want to support scripting abilities, but don't want to develop a new language for the purpose.

Unix/DOS Shells

Although they can't really be classified as programming languages, the Unix shells and the DOS command line do provide some basic scripting capabilities. Under Unix, the primary choices are the Bourne, Korn, and C shells, and variants such as the **bash** and **tcsh** shells. Many of the more advanced shells provide some data-handling and storage capabilities, but they are very limited. In all cases, when it comes to

processing or collating information within a shell, you need to use an external utility such as **sort** or **cut**; many programs actually use Awk or Gawk to achieve some of the more complex options.

Advanced programming within the shell requires heavy use of pipes, and it's often the case that you will have 10 or 15 processes, all joined together by pipes to perform a simple operation that would take just one or two lines within Python. Therefore, although shell-script programming is possible for most solutions, it's just not practical.

Under DOS, and therefore Windows, the only scripting ability you have is through the batch file. Like shells, batch files are only suitable for executing a simple series of commands; they're not really suitable for real programming. Data storage is limited to environment variables only, and you can only pipe or redirect with certain commands. Also, despite attempts with products for DOS, the limitation is still with the available commands, rather than with the environment itself. DOS does not come with even 1% of the tools that are supplied standard with Unix.