Learning\_Python\_3E\_Mark\_C02

22

Chapter 2CHAPTER 2

How Python Runs Programs 2

This chapter and the next give a quick look at program execution—how you launch code, and how Python runs it. In this chapter, we’ll study the Python interpreter. Chapter 3 will then show you how to get your own programs up and running. Startup details are inherently platform-specific, and some of the material in this chapter may not apply to the platform you work on, so you should feel free to skip parts not relevant to your intended use. Likewise, more advanced readers who have used similar tools in the past and prefer to get to the meat of the language quickly may want to file some of this chapter away as “for future reference.” For the rest of you, let’s learn how to run some code.

Este capítulo y el siguiente brindan una mirada rápida a la ejecución del programa: cómo inicia el código y cómo lo ejecuta Python. En este capítulo, estudiaremos el intérprete de Python. El Capítulo 3 le mostrará cómo poner en marcha sus propios programas. Los detalles de inicio son intrínsecamente específicos de la plataforma, y es posible que parte del material de este capítulo no se aplique a la plataforma en la que trabaja, por lo que puede omitir las partes que no sean relevantes para el uso previsto. Del mismo modo, los lectores más avanzados que hayan usado herramientas similares en el pasado y prefieran llegar al meollo del idioma rápidamente pueden querer archivar parte de este capítulo como "para referencia futura". Para el resto de ustedes, aprendamos a ejecutar código.

**Introducing the Python Interpreter**

So far, I’ve mostly been talking about Python as a programming language. But, as currently implemented, it’s also a software package called an interpreter. An interpreter is a kind of program that executes other programs. When you write a Python program, the Python interpreter reads your program and carries out the instructions it contains. In effect, the interpreter is a layer of software logic between your code and the computer hardware on your machine.

Hasta ahora, he estado hablando principalmente de Python como lenguaje de programación. Pero, como está implementado actualmente, también es un paquete de software llamado intérprete. Un intérprete es un tipo de programa que ejecuta otros programas. Cuando escribe un programa de Python, el intérprete de Python lee su programa y lleva a cabo las instrucciones que contiene. En efecto, el intérprete es una capa de lógica de software entre su código y el hardware de su máquina.

When the Python package is installed on your machine, it generates a number of components—minimally, an interpreter and a support library. Depending on how you use it, the Python interpreter may take the form of an executable program, or a set of libraries linked into another program. Depending on which flavor of Python you run, the interpreter itself may be implemented as a C program, a set of Java classes, or something else. Whatever form it takes, the Python code you write must always be run by this interpreter. And, to enable that, you must install a Python interpreter on your computer.

Cuando el paquete de Python está instalado en su máquina, genera una cantidad de componentes, como mínimo, un intérprete y una biblioteca de soporte. Dependiendo de cómo lo use, el intérprete de Python puede tomar la forma de un programa ejecutable o un conjunto de bibliotecas vinculadas a otro programa. Según el tipo de Python que ejecute, el intérprete en sí puede implementarse como un programa C, un conjunto de clases Java o algo más. Sea cual sea la forma que adopte, el código de Python que escriba siempre debe ser ejecutado por este intérprete. Y, para habilitar eso, debe instalar un intérprete de Python en su computadora.

Python installation details vary by platform, and are covered in more depth in Appendix A. In short:

Los detalles de instalación de Python varían según la plataforma y se cubren con más profundidad en el Apéndice A. En resumen:

• Windows users fetch and run a self-installing executable file that puts Python on their machines. Simply double-click and say Yes or Next at all prompts.

• On Windows Vista, you may need to take extra steps to use the Python 2.5 MSI installer file; see Appendix A for more details.

• Linux and Mac OS X users probably already have a usable Python preinstalled on their computers—it’s a standard component on these platforms today.

• Some Linux users (and most Unix users) typically either install Python from RPM files, or compile it from its full source code distribution package.

• Other platforms have installation techniques relevant to those platforms. For instance, Python is also available on cell phones, game consoles, and iPods, but the installation details vary too widely to cover here.

Python itself may be fetched from the downloads page at Python’s web site. It may also be found through various other distribution channels. Keep in mind that you should always check to see whether Python is already present before installing it. If you’re working on Windows, you’ll usually find Python in the Start menu, as captured in Figure 2-1 (these menu options are discussed in the next chapter). On Unix and Linux, Python probably lives in your /usr directory tree.

Python mismo se puede obtener de la página de descargas en el sitio web de Python. También se puede encontrar a través de varios otros canales de distribución. Tenga en cuenta que siempre debe verificar si Python ya está presente antes de instalarlo. Si está trabajando en Windows, normalmente encontrará Python en el menú Inicio, como se muestra en la Figura 2-1 (estas opciones de menú se analizan en el próximo capítulo). En Unix y Linux, Python probablemente vive en su árbol de directorios /usr.

Because installation details are so platform-specific, we’ll finesse the rest of this story here. For more details on the installation process, consult Appendix A. For the purposes of this chapter and the next, I’ll assume that you’ve got Python ready to go.

Debido a que los detalles de instalación son tan específicos de la plataforma, refinaremos el resto de esta historia aquí. Para obtener más detalles sobre el proceso de instalación, consulte el Apéndice A. Para los propósitos de este capítulo y el siguiente, asumiré que tiene Python listo para funcionar.

**Program Execution**

What it means to write and run a Python script depends on whether you look at these tasks as a programmer, or as a Python interpreter. Both views offer important perspectives on Python programming.

Lo que significa escribir y ejecutar un script de Python depende de si considera estas tareas como programador o como intérprete de Python. Ambas vistas ofrecen perspectivas importantes sobre la programación de Python.

**The Programmer’s View**

In its simplest form, a Python program is just a text file containing Python statements. For example, the following file, named script1.py, is one of the simplest Python scripts we could dream up, but it passes for an official Python program:

En su forma más simple, un programa de Python es solo un archivo de texto que contiene declaraciones de Python. Por ejemplo, el siguiente archivo, llamado script1.py, es uno de los scripts de Python más simples que podríamos imaginar, pero pasa por un programa oficial de Python:

print 'hello world'

print 2 \*\* 100

This file contains two Python print statements, which simply print a string (the text in quotes) and a numeric expression result (2 to the power 100) to the output stream. Don’t worry about the syntax of this code yet—for this chapter, we’re interested only in getting it to run. I’ll explain the print statement, and why you can raise 2 to the power 100 in Python without overflowing, in later parts of this book. You can create such a file of statements with any text editor you like. By convention, Python program files are given names that end in .py; technically, this naming scheme is required only for files that are “imported,” as shown later in this book, but most Python files have .py names for consistency.

Este archivo contiene dos declaraciones de impresión de Python, que simplemente imprimen una cadena (el texto entre comillas) y un resultado de expresión numérica (2 elevado a 100) en el flujo de salida. No se preocupe todavía por la sintaxis de este código; en este capítulo, solo nos interesa que se ejecute. Explicaré la declaración de impresión y por qué puede elevar 2 a la potencia 100 en Python sin desbordarse, en partes posteriores de este libro. Puede crear dicho archivo de declaraciones con cualquier editor de texto que desee. Por convención, los archivos de programa de Python reciben nombres que terminan en .py; técnicamente, este esquema de nomenclatura solo se requiere para los archivos que se "importan", como se muestra más adelante en este libro, pero la mayoría de los archivos de Python tienen nombres .py para mantener la coherencia.

After you’ve typed these statements into a text file, you must tell Python to execute the file—which simply means to run all the statements in the file from top to bottom, one after another. As you’ll see in the next chapter, you can launch Python program files by command lines, by clicking their icons, and with other standard techniques. If all goes well, when you execute the file, you’ll see the results of the two print statements show up somewhere on your computer—by default, usually in the same window you were in when you ran the program:

Después de escribir estas declaraciones en un archivo de texto, debe indicarle a Python que ejecute el archivo, lo que simplemente significa ejecutar todas las declaraciones en el archivo de arriba a abajo, una tras otra. Como verá en el próximo capítulo, puede iniciar archivos de programa de Python mediante líneas de comando, haciendo clic en sus iconos y con otras técnicas estándar. Si todo va bien, cuando ejecute el archivo, verá que los resultados de las dos declaraciones de impresión aparecen en algún lugar de su computadora; de manera predeterminada, generalmente en la misma ventana en la que estaba cuando ejecutó el programa:

hello world

1267650600228229401496703205376km

For example, here’s what happened when I ran this script from a DOS command line on a Windows laptop (typically called a Command Prompt window, found in the Accessories program menu), to make sure it didn’t have any silly typos:

Por ejemplo, esto es lo que sucedió cuando ejecuté este script desde una línea de comandos de DOS en una computadora portátil con Windows (generalmente llamada ventana del símbolo del sistema, que se encuentra en el menú del programa Accesorios), para asegurarme de que no tuviera errores tipográficos tontos:

D:\temp> python script1.py

hello world

1267650600228229401496703205376

We’ve just run a Python script that prints a string and a number. We probably won’t win any programming awards with this code, but it’s enough to capture the basics of program execution.

Acabamos de ejecutar un script de Python que imprime una cadena y un número. Probablemente no ganemos ningún premio de programación con este código, pero es suficiente para capturar los conceptos básicos de la ejecución del programa.

**Python’s View**

The brief description in the prior section is fairly standard for scripting languages, and it’s usually all that most Python programmers need to know. You type code into text files, and you run those files through the interpreter. Under the hood, though, a bit more happens when you tell Python to “go.” Although knowledge of Python internals is not strictly required for Python programming, a basic understanding of the runtime structure of Python can help you grasp the bigger picture of program execution. When you instruct Python to run your script, there are a few steps that Python carries out before your code actually starts crunching away. Specifically, it’s first compiled to something called “byte code” and then routed to something called a “virtual machine.”

La breve descripción de la sección anterior es bastante estándar para los lenguajes de secuencias de comandos y, por lo general, es todo lo que la mayoría de los programadores de Python necesitan saber. Escribe código en archivos de texto y ejecuta esos archivos a través del intérprete. Sin embargo, debajo del capó, sucede un poco más cuando le dices a Python que "vaya". Aunque no se requiere estrictamente el conocimiento de los aspectos internos de Python para la programación de Python, una comprensión básica de la estructura de tiempo de ejecución de Python puede ayudarlo a comprender el panorama general de la ejecución del programa. Cuando le indica a Python que ejecute su secuencia de comandos, hay algunos pasos que Python lleva a cabo antes de que su código realmente comience a procesarse. Específicamente, primero se compila en algo llamado "código de bytes" y luego se enruta a algo llamado "máquina virtual".

**Byte code compilation**

Internally, and almost completely hidden from you, when you execute a program, Python first compiles your source code (the statements in your file) into a format known as byte code. Compilation is simply a translation step, and byte code is a lower-level, platform-independent representation of your source code. Roughly, Python translates each of your source statements into a group of byte code instructions by decomposing them into individual steps. This byte code translation is performed to speed execution—byte code can be run much more quickly than the original source code statements in your text file.

Internamente, y casi completamente oculto para usted, cuando ejecuta un programa, Python primero compila su código fuente (las declaraciones en su archivo) en un formato conocido como código de bytes. La compilación es simplemente un paso de traducción, y el código de bytes es una representación de nivel inferior e independiente de la plataforma de su código fuente. Aproximadamente, Python traduce cada una de sus declaraciones fuente en un grupo de instrucciones de código de bytes al descomponerlas en pasos individuales. Esta traducción del código de bytes se realiza para acelerar la ejecución: el código de bytes se puede ejecutar mucho más rápido que las declaraciones del código fuente original en su archivo de texto.

You’ll notice that the prior paragraph said that this is almost completely hidden from you. If the Python process has write access on your machine, it will store the byte code of your program in files that end with a .pyc extension (“.pyc” means compiled “.py” source). You will see these files show up on your computer after you’ve run a few programs alongside the corresponding source code files (that is, in the same directories).

Notarás que el párrafo anterior decía que esto está casi completamente oculto para ti. Si el proceso de Python tiene acceso de escritura en su máquina, almacenará el código de bytes de su programa en archivos que terminan con una extensión .pyc (“.pyc” significa fuente compilada “.py”). Verá que estos archivos aparecen en su computadora después de ejecutar algunos programas junto con los archivos de código fuente correspondientes (es decir, en los mismos directorios).

Python saves byte code like this as a startup speed optimization. The next time you run your program, Python will load the .pyc and skip the compilation step, as long as you haven’t changed your source code since the byte code was last saved. Python automatically checks the timestamps of source and byte code files to know when it must recompile—if you resave your source code, byte code is automatically re-created the next time your program is run.

Python guarda un código de bytes como este como una optimización de la velocidad de inicio. La próxima vez que ejecute su programa, Python cargará el .pyc y omitirá el paso de compilación, siempre que no haya cambiado su código fuente desde la última vez que guardó el código de bytes. Python verifica automáticamente las marcas de tiempo de los archivos de código fuente y de bytes para saber cuándo debe volver a compilarse; si vuelve a guardar su código fuente, el código de bytes se vuelve a crear automáticamente la próxima vez que se ejecuta su programa.

If Python cannot write the byte code files to your machine, your program still works—the byte code is generated in memory and simply discarded on program exit.\* However, because .pyc files speed startup time, you’ll want to make sure they are written for larger programs. Byte code files are also one way to ship Python programs—Python is happy to run a program if all it can find are .pyc files, even if the original .py source files are absent. (See “Frozen Binaries” later in this chapter for another shipping option.)

Si Python no puede escribir los archivos de código de bytes en su máquina, su programa aún funciona: el código de bytes se genera en la memoria y simplemente se descarta al salir del programa.\* Sin embargo, debido a que los archivos .pyc aceleran el tiempo de inicio, querrá asegurarse de que están escritos para programas más grandes. Los archivos de código de bytes también son una forma de enviar programas de Python: Python está feliz de ejecutar un programa si todo lo que puede encontrar son archivos .pyc, incluso si los archivos fuente originales .py están ausentes. (Consulte "Binarios congelados" más adelante en este capítulo para conocer otra opción de envío).

**The Python Virtual Machine (PVM)**

Once your program has been compiled to byte code (or the byte code has been loaded from existing .pyc files), it is shipped off for execution to something generally known as the Python Virtual Machine (PVM, for the more acronym-inclined among you). The PVM sounds more impressive than it is; really, it’s not a separate program, and it need not be installed by itself. In fact, the PVM is just a big loop that iterates through your byte code instructions, one by one, to carry out their operations. The PVM is the runtime engine of Python; it’s always present as part of the Python system, and it’s the component that truly runs your scripts. Technically, it’s just the last step of what is called the “Python interpreter.” Figure 2-2 illustrates the runtime structure described here. Keep in mind that all of this complexity is deliberately hidden from Python programmers. Byte code compilation is automatic, and the PVM is just part of the Python system that you have installed on your machine. Again, programmers simply code and run files of statements.

Una vez que su programa se ha compilado en código de bytes (o el código de bytes se ha cargado desde archivos .pyc existentes), se envía para su ejecución a algo generalmente conocido como Python Virtual Machine (PVM, por sus siglas en inglés). ). El PVM suena más impresionante de lo que es; realmente, no es un programa separado, y no necesita ser instalado por sí mismo. De hecho, el PVM es solo un gran bucle que itera a través de las instrucciones de código de bytes, una por una, para llevar a cabo sus operaciones. El PVM es el motor de tiempo de ejecución de Python; siempre está presente como parte del sistema Python y es el componente que realmente ejecuta sus scripts. Técnicamente, es solo el último paso de lo que se llama el "intérprete de Python". La Figura 2-2 ilustra la estructura del tiempo de ejecución descrita aquí. Tenga en cuenta que toda esta complejidad se oculta deliberadamente a los programadores de Python. La compilación del código de bytes es automática y el PVM es solo una parte del sistema Python que ha instalado en su máquina. Nuevamente, los programadores simplemente codifican y ejecutan archivos de declaraciones.

**Performance implications**

Readers with a background in fully compiled languages such as C and C++ might notice a few differences in the Python model. For one thing, there is usually no build or “make” step in Python work: code runs immediately after it is written. For another, Python byte code is not binary machine code (e.g., instructions for an Intel chip). Byte code is a Python-specific representation.

Los lectores con experiencia en lenguajes completamente compilados, como C y C++, pueden notar algunas diferencias en el modelo de Python. Por un lado, normalmente no hay un paso de compilación o “hacer” en el trabajo de Python: el código se ejecuta inmediatamente después de escribirlo. Por otro lado, el código de bytes de Python no es un código de máquina binario (por ejemplo, instrucciones para un chip Intel). El código de bytes es una representación específica de Python.

\* And, strictly speaking, byte code is saved only for files that are imported, not for the top-level file of a program. We’ll explore imports in Chapter 3, and again in Part V. Byte code is also never saved for code typed at the interactive prompt, which is described in Chapter 3

Y, estrictamente hablando, el código de bytes se guarda solo para los archivos que se importan, no para el archivo de nivel superior de un programa. Exploraremos las importaciones en el Capítulo 3 y nuevamente en la Parte V. El código de bytes tampoco se guarda nunca para el código escrito en el mensaje interactivo, que se describe en el Capítulo 3.

This is why some Python code may not run as fast as C or C++ code, as described in Chapter 1—the PVM loop, not the CPU chip, still must interpret the byte code, and byte code instructions require more work than CPU instructions. On the other hand, unlike in classic interpreters, there is still an internal compile step—Python does not need to reanalyze and reparse each source statement repeatedly. The net effect is that pure Python code runs at speeds somewhere between those of a traditional compiled language and a traditional interpreted language. See Chapter 1 for more on Python performance implications.

Esta es la razón por la que algunos códigos Python pueden no ejecutarse tan rápido como el código C o C++, como se describe en el Capítulo 1: el bucle PVM, no el chip de la CPU, aún debe interpretar el código de bytes, y las instrucciones del código de bytes requieren más trabajo que las instrucciones de la CPU. Por otro lado, a diferencia de los intérpretes clásicos, todavía hay un paso de compilación interno: Python no necesita volver a analizar y analizar cada declaración fuente repetidamente. El efecto neto es que el código Python puro se ejecuta a velocidades entre las de un lenguaje compilado tradicional y un lenguaje interpretado tradicional. Consulte el Capítulo 1 para obtener más información sobre las implicaciones de rendimiento de Python.

**Development implications**

Another ramification of Python’s execution model is that there is really no distinction between the development and execution environments. That is, the systems that compile and execute your source code are really one and the same. This similarity may have a bit more significance to readers with a background in traditional compiled languages, but in Python, the compiler is always present at runtime, and is part of the system that runs programs.

Otra ramificación del modelo de ejecución de Python es que realmente no hay distinción entre los entornos de desarrollo y ejecución. Es decir, los sistemas que compilan y ejecutan su código fuente son realmente uno y el mismo. Esta similitud puede tener un poco más de importancia para los lectores con experiencia en lenguajes compilados tradicionales, pero en Python, el compilador siempre está presente en el tiempo de ejecución y es parte del sistema que ejecuta los programas.

This makes for a much more rapid development cycle. There is no need to precompile and link before execution may begin; simply type and run the code. This also adds a much more dynamic flavor to the language—it is possible, and often very convenient, for Python programs to construct and execute other Python programs at runtime. The eval and exec built-ins, for instance, accept and run strings containing Python program code. This structure is also why Python lends itself to product customization—because Python code can be changed on the fly, users can modify the Python parts of a system onsite without needing to have or compile the entire system’s code.

Esto hace que el ciclo de desarrollo sea mucho más rápido. No hay necesidad de precompilar y vincular antes de que pueda comenzar la ejecución; simplemente escriba y ejecute el código. Esto también agrega un sabor mucho más dinámico al lenguaje: es posible, y a menudo muy conveniente, que los programas de Python construyan y ejecuten otros programas de Python en tiempo de ejecución. Los integrados eval y exec, por ejemplo, aceptan y ejecutan cadenas que contienen código de programa Python. Esta estructura también es la razón por la cual Python se presta a la personalización del producto, ya que el código de Python se puede cambiar sobre la marcha, los usuarios pueden modificar las partes de Python de un sistema en el sitio sin necesidad de tener o compilar todo el código del sistema.

At a more fundamental level, keep in mind that all we really have in Python is runtime—there is no initial compile-time phase at all, and everything happens as the program is running. This even includes operations such as the creation of functions and classes and the linkage of modules. Such events occur before execution in more static languages, but happen as programs execute in Python. As we’ll see, the net effect makes for a much more dynamic programming experience than that to which some readers may be accustomed.

En un nivel más fundamental, tenga en cuenta que todo lo que realmente tenemos en Python es el tiempo de ejecución: no hay ninguna fase inicial de tiempo de compilación y todo sucede mientras se ejecuta el programa. Esto incluso incluye operaciones como la creación de funciones y clases y la vinculación de módulos. Dichos eventos ocurren antes de la ejecución en lenguajes más estáticos, pero ocurren cuando los programas se ejecutan en Python. Como veremos, el efecto neto genera una experiencia de programación mucho más dinámica que aquella a la que algunos lectores pueden estar acostumbrados.

**Execution Model Variations**

Before moving on, I should point out that the internal execution flow described in the prior section reflects the standard implementation of Python today, and is not really a requirement of the Python language itself. Because of that, the execution model is prone to changing with time. In fact, there are already a few systems that modify the picture in Figure 2-2 somewhat. Let’s take a few moments to explore the most prominent of these variations.

Antes de continuar, debo señalar que el flujo de ejecución interno descrito en la sección anterior refleja la implementación estándar de Python en la actualidad y no es realmente un requisito del lenguaje Python en sí. Por eso, el modelo de ejecución es propenso a cambiar con el tiempo. De hecho, ya hay algunos sistemas que modifican un poco la imagen de la Figura 2-2. Tomemos unos momentos para explorar la más destacada de estas variaciones.

**Python Implementation Alternatives**

Really, as this book is being written, there are three primary implementations of the Python language—CPython, Jython, and IronPython—along with a handful of secondary implementations such as Stackless Python. In brief, CPython is the standard implementation; all the others have very specific purposes and roles. All implement the same Python language, but execute programs in different ways.

En realidad, mientras se escribe este libro, hay tres implementaciones principales del lenguaje Python: CPython, Jython y IronPython, junto con un puñado de implementaciones secundarias, como Stackless Python. En resumen, CPython es la implementación estándar; todos los demás tienen propósitos y roles muy específicos. Todos implementan el mismo lenguaje Python, pero ejecutan programas de diferentes maneras.

**CPython**

The original, and standard, implementation of Python is usually called CPython, when you want to contrast it with the other two. Its name comes from the fact that it is coded in portable ANSI C language code. This is the Python that you fetch from http://www.python.org, get with the ActivePython distribution, and have automatically on most Linux and Mac OS X machines. If you’ve found a preinstalled version of Python on your machine, it’s probably CPython, unless your company is using Python in very specialized ways. Unless you want to script Java or .NET applications with Python, you probably want to use the standard CPython system. Because it is the reference implementation of the language, it tends to run the fastest, be the most complete, and be more robust than the alternative systems. Figure 2-2 reflects CPython’s runtime architecture.

La implementación original y estándar de Python generalmente se llama CPython, cuando desea contrastarlo con los otros dos. Su nombre proviene del hecho de que está codificado en código portátil de lenguaje ANSI C. Este es el Python que obtiene de http://www.python.org, obtiene con la distribución ActivePython y lo tiene automáticamente en la mayoría de las máquinas Linux y Mac OS X. Si ha encontrado una versión preinstalada de Python en su máquina, probablemente sea CPython, a menos que su empresa utilice Python de formas muy especializadas. A menos que desee programar aplicaciones Java o .NET con Python, probablemente desee utilizar el sistema CPython estándar. Debido a que es la implementación de referencia del lenguaje, tiende a ejecutarse más rápido, ser el más completo y ser más robusto que los sistemas alternativos. La Figura 2-2 refleja la arquitectura de tiempo de ejecución de CPython.

**Jython**

The Jython system (originally known as JPython) is an alternative implementation of the Python language, targeted for integration with the Java programming language. Jython consists of Java classes that compile Python source code to Java byte code and then route the resulting byte code to the Java Virtual Machine (JVM). Programmers still code Python statements in .py text files as usual; the Jython system essentially just replaces the rightmost two bubbles in Figure 2-2 with Java-based equivalents. Jython’s goal is to allow Python code to script Java applications, much as CPython allows Python to script C and C++ components. Its integration with Java is remarkably seamless. Because Python code is translated to Java byte code, it looks and feels like a true Java program at runtime. Jython scripts can serve as web applets and servlets, build Java-based GUIs, and so on. Moreover, Jython includes integration support that allows Python code to import and use Java classes as though they were coded in Python. Because Jython is slower and less robust than CPython, though, it is usually seen as a tool of interest primarily to Java developers looking for a scripting language to be a frontend to Java code.

La implementación original y estándar de Python generalmente se llama El sistema Jython (originalmente conocido como JPython) es una implementación alternativa del lenguaje Python, destinada a la integración con el lenguaje de programación Java. Jython consta de clases de Java que compilan el código fuente de Python en código de bytes de Java y luego enrutan el código de bytes resultante a la máquina virtual de Java (JVM). Los programadores todavía codifican declaraciones de Python en archivos de texto .py como de costumbre; el sistema Jython esencialmente solo reemplaza las dos burbujas más a la derecha en la Figura 2-2 con equivalentes basados en Java. El objetivo de Jython es permitir que el código Python genere secuencias de comandos de aplicaciones Java, al igual que CPython permite que Python genere secuencias de comandos de componentes C y C++. Su integración con Java es notablemente fluida. Debido a que el código de Python se traduce a código de bytes de Java, se ve y se siente como un verdadero programa de Java en tiempo de ejecución. Los scripts Jython pueden servir como applets y servlets web, crear GUI basadas en Java, etc. Además, Jython incluye soporte de integración que permite que el código de Python importe y use clases de Java como si estuvieran codificadas en Python. Sin embargo, debido a que Jython es más lento y menos robusto que CPython, generalmente se considera una herramienta de interés principalmente para los desarrolladores de Java que buscan un lenguaje de secuencias de comandos para ser una interfaz para el código Java.

**IronPython**

A third (and, at this writing, still somewhat new) implementation of Python, IronPython is designed to allow Python programs to integrate with applications coded to work with Microsoft’s .NET Framework for Windows, as well as the Mono open source equivalent for Linux. .NET and its C# programming language runtime system are designed to be a language-neutral object communication layer, in the spirit of Microsoft’s earlier COM model. IronPython allows Python programs to act as both client and server components, accessible from other .NET languages.

IronPython, una tercera (y, en este momento, todavía algo nueva) implementación de Python, está diseñada para permitir que los programas de Python se integren con aplicaciones codificadas para funcionar con .NET Framework de Microsoft para Windows, así como con el equivalente de código abierto Mono para Linux. .NET y su sistema de tiempo de ejecución del lenguaje de programación C# están diseñados para ser una capa de comunicación de objetos independiente del lenguaje, en el espíritu del modelo COM anterior de Microsoft. IronPython permite que los programas de Python actúen como componentes de cliente y servidor, accesibles desde otros lenguajes .NET.

By implementation, IronPython is very much like Jython (and, in fact, is being developed by the same creator)—it replaces the last two bubbles in Figure 2-2 with equivalents for execution in the .NET environment. Also, like Jython, IronPython has a special focus—it is primarily of interest to developers integrating Python with .NET components. Because it is being developed by Microsoft, though, IronPython might also be able to leverage some important optimization tools for better performance. IronPython’s scope is still evolving as I write this; for more details, consult the Python online resources, or search the Web. \*

Por implementación, IronPython es muy parecido a Jython (y, de hecho, lo está desarrollando el mismo creador): reemplaza las dos últimas burbujas en la Figura 2-2 con equivalentes para la ejecución en el entorno .NET. Además, al igual que Jython, IronPython tiene un enfoque especial: es principalmente de interés para los desarrolladores que integran Python con componentes .NET. Sin embargo, debido a que está siendo desarrollado por Microsoft, IronPython también podría aprovechar algunas herramientas de optimización importantes para un mejor rendimiento. El alcance de IronPython aún está evolucionando mientras escribo esto; para obtener más detalles, consulte los recursos en línea de Python o busque en la Web.

**Execution Optimization Tools**

CPython, Jython, and IronPython all implement the Python language in similar ways: by compiling source code to byte code, and executing the byte code on an appropriate virtual machine. Still other systems, including the Psyco just-in-time compiler, and the Shedskin C++ translator, instead attempt to optimize the basic execution model. These systems are not required knowledge at this point in your Python career, but a quick look at their place in the execution model might help demystify the model in general.

CPython, Jython y IronPython implementan el lenguaje Python de manera similar: compilando el código fuente en código de bytes y ejecutando el código de bytes en una máquina virtual adecuada. Otros sistemas, incluido el compilador Psyco justo a tiempo y el traductor Shedskin C++, intentan optimizar el modelo de ejecución básico. Estos sistemas no son conocimientos necesarios en este punto de su carrera de Python, pero una mirada rápida a su lugar en el modelo de ejecución podría ayudar a desmitificar el modelo en general.

**The Psyco just-in-time compiler**

The Psyco system is not another Python implementation, but rather, a component that extends the byte code execution model to make programs run faster. In terms of Figure 2-2, Psyco is an enhancement to the PVM that collects and uses type information while the program runs to translate portions of the program’s byte code all the way down to real binary machine code for faster execution. Psyco accomplishes this translation without requiring changes to the code or a separate compilation step during development.

El sistema Psyco no es otra implementación de Python, sino un componente que amplía el modelo de ejecución de código de bytes para hacer que los programas se ejecuten más rápido. En términos de la Figura 2-2, Psyco es una mejora del PVM que recopila y usa información de tipos mientras el programa se ejecuta para traducir partes del código de bytes del programa hasta el código de máquina binario real para una ejecución más rápida. Psyco logra esta traducción sin requerir cambios en el código o un paso de compilación por separado durante el desarrollo.

\* Jython and IronPython are completely independent implementations of Python that compile Python source for different runtime architectures. It is also possible to access Java and .NET software from standard CPython programs: the JPype and Python for .NET systems, for example, allow CPython code to call out to Java and .NET components

Jython y IronPython son implementaciones completamente independientes de Python que compilan el código fuente de Python para diferentes arquitecturas de tiempo de ejecución. También es posible acceder al software Java y .NET desde programas CPython estándar: JPype y Python para sistemas .NET, por ejemplo, permiten que el código CPython llame a componentes Java y .NET.

Roughly, while your program runs, Psyco collects information about the kinds of objects being passed around; that information can be used to generate highly efficient machine code tailored for those object types. Once generated, the machine code then replaces the corresponding part of the original byte code to speed your program’s overall execution. The net effect is that, with Psyco, your program becomes much quicker over time, and as it is running. In ideal cases, some Python code may become as fast as compiled C code under Psyco.

Aproximadamente, mientras se ejecuta su programa, Psyco recopila información sobre los tipos de objetos que se pasan; esa información se puede utilizar para generar un código de máquina altamente eficiente adaptado a esos tipos de objetos. Una vez generado, el código de máquina reemplaza la parte correspondiente del código de bytes original para acelerar la ejecución general de su programa. El efecto neto es que, con Psyco, su programa se vuelve mucho más rápido con el tiempo y mientras se ejecuta. En casos ideales, algunos códigos Python pueden volverse tan rápidos como el código C compilado bajo Psyco.

Because this translation from byte code happens at program runtime, Psyco is generally known as a just-in-time (JIT) compiler. Psyco is actually a bit different from the JIT compilers some readers may have seen for the Java language, though. Really, Psyco is a specializing JIT compiler—it generates machine code tailored to the data types that your program actually uses. For example, if a part of your program uses different data types at different times, Psyco may generate a different version of machine code to support each different type combination. Psyco has been shown to speed Python code dramatically. According to its web page, Psyco provides “2x to 100x speed-ups, typically 4x, with an unmodified Python interpreter and unmodified source code, just a dynamically loadable C extension module.” Of equal significance, the largest speedups are realized for algorithmic code written in pure Python—exactly the sort of code you might normally migrate to C to optimize. With Psyco, such migrations become even less important.

Debido a que esta traducción del código de bytes ocurre en el tiempo de ejecución del programa, Psyco generalmente se conoce como un compilador justo a tiempo (JIT). Sin embargo, Psyco es un poco diferente de los compiladores JIT que algunos lectores pueden haber visto para el lenguaje Java. Realmente, Psyco es un compilador JIT especializado: genera código de máquina adaptado a los tipos de datos que su programa realmente usa. Por ejemplo, si una parte de su programa usa diferentes tipos de datos en diferentes momentos, Psyco puede generar una versión diferente de código de máquina para admitir cada combinación de tipos diferente. Se ha demostrado que Psyco acelera dramáticamente el código de Python. Según su página web, Psyco proporciona "aceleraciones de 2x a 100x, generalmente 4x, con un intérprete de Python sin modificar y un código fuente sin modificar, solo un módulo de extensión C cargable dinámicamente". De igual importancia, las mayores aceleraciones se realizan para el código algorítmico escrito en Python puro, exactamente el tipo de código que normalmente podría migrar a C para optimizar. Con Psico, tales migraciones se vuelven aún menos importantes.

Psyco is not yet a standard part of Python; you will have to fetch and install it separately. It is also still something of a research project, so you’ll have to track its evolution online. In fact, at this writing, although Psyco can still be fetched and installed by itself, it appears that much of the system may eventually be absorbed into the newer “PyPy” project—an attempt to reimplement Python’s PVM in Python code, to better support optimizations like Psyco. Perhaps the largest downside of Psyco is that it currently only generates machine code for Intel x86 architecture chips, though this includes Windows, Linux, and recent Macs. For more details on the Psyco extension, and other JIT efforts that may arise, consult http://www.python.org; you can also check out Psyco’s home page, which currently resides at <http://psyco.sourceforge.net>.

Psyco aún no es una parte estándar de Python; tendrá que buscarlo e instalarlo por separado. También es algo así como un proyecto de investigación, por lo que tendrá que seguir su evolución en línea. De hecho, en el momento de escribir este artículo, aunque Psyco todavía puede obtenerse e instalarse por sí mismo, parece que gran parte del sistema puede eventualmente ser absorbido por el nuevo proyecto "PyPy", un intento de volver a implementar el PVM de Python en el código de Python, para brindar un mejor soporte. optimizaciones como Psico. Quizás la mayor desventaja de Psyco es que actualmente solo genera código de máquina para chips de arquitectura Intel x86, aunque esto incluye Windows, Linux y Mac recientes. Para obtener más detalles sobre la extensión de Psyco y otros esfuerzos JIT que puedan surgir, consulte http://www.python.org; también puede consultar la página de inicio de Psyco, que actualmente se encuentra en <http://psyco.sourceforge.net>.

**The Shedskin C++ translator**

Shedskin is an emerging system that takes a different approach to Python program execution—it attempts to translate Python source code to C++ code, which your computer’s C++ compiler then compiles to machine code. As such, it represents a platform-neutral approach to running Python code. Shedskin is still somewhat experimental as I write these words, and it limits Python programs to an implicit statically typed constraint that is technically not normal Python, so we won’t go into further detail here. Initial results, though, show that it has the potential to outperform both standard Python and the Psyco extension in terms of execution speed, and it is a promising project. Search the Web for details on the project’s current status.

Shedskin es un sistema emergente que adopta un enfoque diferente para la ejecución del programa Python: intenta traducir el código fuente de Python a código C++, que luego el compilador C++ de su computadora compila en código de máquina. Como tal, representa un enfoque de plataforma neutral para ejecutar el código de Python. Shedskin todavía es algo experimental mientras escribo estas palabras, y limita los programas de Python a una restricción implícita de tipo estático que técnicamente no es Python normal, por lo que no entraremos en más detalles aquí. Sin embargo, los resultados iniciales muestran que tiene el potencial de superar tanto a Python estándar como a la extensión de Psyco en términos de velocidad de ejecución, y es un proyecto prometedor. Busque en la Web para obtener detalles sobre el estado actual del proyecto.

**Frozen Binaries**

Sometimes when people ask for a “real” Python compiler, what they’re really seeking is simply a way to generate standalone binary executables from their Python programs. This is more a packaging and shipping idea than an execution-flow concept, but it’s somewhat related. With the help of third-party tools that you can fetch off the Web, it is possible to turn your Python programs into true executables— known as frozen binaries in the Python world.

A veces, cuando las personas solicitan un compilador de Python "real", lo que realmente buscan es simplemente una forma de generar ejecutables binarios independientes a partir de sus programas de Python. Esta es más una idea de empaque y envío que un concepto de flujo de ejecución, pero está algo relacionado. Con la ayuda de herramientas de terceros que puede obtener de la Web, es posible convertir sus programas de Python en verdaderos ejecutables, conocidos como archivos binarios congelados en el mundo de Python.

Frozen binaries bundle together the byte code of your program files, along with the PVM (interpreter), and any Python support files your program needs, into a single package. There are some variations on this theme, but the end result can be a single binary executable program (e.g., an .exe file on Windows) that can easily be shipped to customers. In Figure 2-2, it is as though the byte code and PVM are merged into a single component—a frozen binary file.

Los archivos binarios congelados agrupan el código de bytes de sus archivos de programa, junto con el PVM (intérprete) y cualquier archivo de soporte de Python que necesite su programa, en un solo paquete. Hay algunas variaciones de este tema, pero el resultado final puede ser un solo programa ejecutable binario (por ejemplo, un archivo .exe en Windows) que se puede enviar fácilmente a los clientes. En la Figura 2-2, es como si el código de bytes y el PVM se fusionaran en un solo componente: un archivo binario congelado.

Today, three primary systems are capable of generating frozen binaries: py2exe (for Windows), PyInstaller (which is similar to py2exe, but works on Linux and Unix, too, and is also capable of generating self-installing binaries), and freeze (the original). You may have to fetch these tools separately from Python itself, but they are available free of charge. They are also constantly evolving, so see <http://www.python.org> and the Vaults of Parnassus web site (http://www.vex.net/parnassus/) for more on these tools. To give you an idea of the scope of these systems, py2exe can freeze standalone programs that use the Tkinter, Pmw, wxPython, and PyGTK GUI libraries; programs that use the pygame game programming toolkit; win32com client programs; and more.

Hoy en día, tres sistemas principales son capaces de generar binarios congelados: py2exe (para Windows), PyInstaller (que es similar a py2exe, pero también funciona en Linux y Unix, y también es capaz de generar binarios autoinstalables) y congelar ( el original). Es posible que deba obtener estas herramientas por separado de Python, pero están disponibles de forma gratuita. También están en constante evolución, así que consulte http://www.python.org y el sitio web Vaults of Parnassus (http://www.vex.net/parnassus/) para obtener más información sobre estas herramientas. Para darle una idea del alcance de estos sistemas, py2exe puede congelar programas independientes que usan las bibliotecas GUI Tkinter, Pmw, wxPython y PyGTK; programas que usan el kit de herramientas de programación de juegos pygame; programas cliente win32com; y más.

Frozen binaries are not the same as the output of a true compiler—they run byte code through a virtual machine. Hence, apart from a possible startup improvement, frozen binaries run at the same speed as the original source files. Frozen binaries are not small (they contain a PVM), but by current standards, they are not unusually large either. Because Python is embedded in the frozen binary, it does not have to be installed on the receiving end to run your program. Moreover, because your code is embedded in the frozen binary, it is effectively hidden from recipients. This single file-packaging scheme is especially appealing to developers of commercial software. For instance, a Python-coded user interface program based on the Tkinter toolkit can be frozen into an executable file and shipped as a self-contained program on a CD or on the Web. End users do not need to install (or even have to know about) Python to run the shipped program.

Los binarios congelados no son lo mismo que la salida de un compilador real: ejecutan código de bytes a través de una máquina virtual. Por lo tanto, además de una posible mejora de inicio, los archivos binarios congelados se ejecutan a la misma velocidad que los archivos fuente originales. Los binarios congelados no son pequeños (contienen un PVM), pero según los estándares actuales, tampoco son inusualmente grandes. Debido a que Python está incrustado en el binario congelado, no es necesario instalarlo en el extremo receptor para ejecutar su programa. Además, debido a que su código está incrustado en el binario congelado, está efectivamente oculto para los destinatarios. Este esquema de empaquetado de un solo archivo es especialmente atractivo para los desarrolladores de software comercial. Por ejemplo, un programa de interfaz de usuario codificado en Python basado en el kit de herramientas Tkinter puede congelarse en un archivo ejecutable y enviarse como un programa autónomo en un CD o en la Web. Los usuarios finales no necesitan instalar Python (ni siquiera deben conocerlo) para ejecutar el programa enviado.

Future Possibilities?

Finally, note that the runtime execution model sketched here is really an artifact of the current implementation of Python, and not the language itself. For instance, it’s not impossible that a full, traditional compiler for translating Python source code to machine code may appear during the shelf life of this book (although one has not in over a decade). New byte code formats and implementation variants may also be adopted in the future. For instance:

Finalmente, tenga en cuenta que el modelo de ejecución en tiempo de ejecución esbozado aquí es realmente un artefacto de la implementación actual de Python, y no del lenguaje en sí. Por ejemplo, no es imposible que un compilador completo y tradicional para traducir el código fuente de Python a código de máquina pueda aparecer durante la vida útil de este libro (aunque no ha aparecido en más de una década). Es posible que en el futuro también se adopten nuevos formatos de código de bytes y variantes de implementación. Por ejemplo:

• The emerging Parrot project aims to provide a common byte code format, virtual machine, and optimization techniques for a variety of programming languages (see <http://www.python.org>).

El proyecto emergente Parrot tiene como objetivo proporcionar un formato de código de bytes común, una máquina virtual y técnicas de optimización para una variedad de lenguajes de programación (ver <http://www.python.org>).

• The Stackless Python system is a standard CPython implementation variant that does not save state on the C language call stack. This makes Python more easily ported to small stack architectures, and opens up novel programming possibilities, such as coroutines.

El sistema Stackless Python es una variante de implementación estándar de CPython que no guarda el estado en la pila de llamadas del lenguaje C. Esto hace que Python se transfiera más fácilmente a arquitecturas de pila pequeña y abre posibilidades de programación novedosas, como las corrutinas.

• The new PyPy project is an attempt to reimplement the PVM in Python itself to enable new implementation techniques.

El nuevo proyecto PyPy es un intento de volver a implementar el PVM en Python para permitir nuevas técnicas de implementación.

Although such future implementation schemes may alter the runtime structure of Python somewhat, it seems likely that the byte code compiler will still be the standard for some time to come. The portability and runtime flexibility of byte code are important features of many Python systems. Moreover, adding type constraint declarations to support static compilation would break the flexibility, conciseness, simplicity, and overall spirit of Python coding. Due to Python’s highly dynamic nature, any future implementation will likely retain many artifacts of the current PVM.

Aunque tales esquemas de implementación futuros pueden alterar un poco la estructura del tiempo de ejecución de Python, parece probable que el compilador de código de bytes siga siendo el estándar durante algún tiempo. La portabilidad y la flexibilidad del tiempo de ejecución del código de bytes son características importantes de muchos sistemas de Python. Además, agregar declaraciones de restricciones de tipo para admitir la compilación estática rompería la flexibilidad, la concisión, la simplicidad y el espíritu general de la codificación de Python. Debido a la naturaleza altamente dinámica de Python, cualquier implementación futura probablemente retendrá muchos artefactos del PVM actual.

**Chapter Summary**

This chapter introduced the execution model of Python (how Python runs your programs) and explored some common variations on that model (just-in-time compilers and the like). Although you don’t really need to come to grips with Python internals to write Python scripts, a passing acquaintance with this chapter’s topics will help you truly understand how your programs run once you start coding them. In the next chapter, you’ll start actually running some code of your own. First, though, here’s the usual chapter quiz.

Este capítulo presentó el modelo de ejecución de Python (cómo Python ejecuta sus programas) y exploró algunas variaciones comunes de ese modelo (compiladores justo a tiempo y similares). Aunque en realidad no necesita familiarizarse con las partes internas de Python para escribir secuencias de comandos de Python, un conocimiento superficial de los temas de este capítulo lo ayudará a comprender realmente cómo se ejecutan sus programas una vez que comience a codificarlos. En el próximo capítulo, comenzará a ejecutar su propio código. Primero, sin embargo, aquí está el cuestionario de capítulo habitual.

**Chapter Quiz**

1. What is the Python interpreter?

2. What is source code?

3. What is byte code?

4. What is the PVM?

5. Name two variations on Python’s standard execution model.

6. How are CPython, Jython, and IronPython different?

**Quiz Answers**

1. The Python interpreter is a program that runs the Python programs you write.

2. Source code is the statements you write for your program—it consists of text in

text files that normally end with a .py extension.

3. Byte code is the lower-level form of your program after Python compiles it.

Python automatically stores byte code in files with a .pyc extension.

4. The PVM is the Python Virtual Machine—the runtime engine of Python that

interprets your compiled code.

5. Psyco, Shedskin, and frozen binaries are all variations on the execution model.

6. CPython is the standard implementation of the language. Jython and IronPython

implement Python programs for use in Java and .NET environments, respec-

tively; they are alternative compilers for Python