

O'REILLY®

# Pitón para Excel

Un entorno moderno para la automatización y  
el análisis de datos



Felix Zumstein

## Python para Excel

Si bien Excel sigue siendo omnipresente en el mundo empresarial, los foros de comentarios recientes de Microsoft están llenos de solicitudes para incluir Python como un lenguaje de programación de Excel. De hecho, es la característica principal solicitada. ¿Qué hace que esta combinación sea tan atractiva? En esta guía práctica, Felix Zumstein, creador de xlwings, un popular paquete de código abierto para automatizar Excel con Python, muestra a los usuarios experimentados de Excel cómo integrar estos dos mundos de manera eficiente.

Excel ha agregado bastantes capacidades nuevas en los últimos años, pero su lenguaje de automatización, VBA, dejó de evolucionar hace mucho tiempo. Muchos usuarios avanzados de Excel ya han adoptado Python para las tareas de automatización diarias. Esta guía le ayuda a empezar.

- Utilice Python sin amplios conocimientos de programación
- Empiece a utilizar herramientas modernas, incluidos los cuadernos de Jupyter y Visual Studio Code
- Utilice pandas para adquirir, limpiar y analizar datos y reemplazar los cálculos típicos de Excel
- Automatice tareas tediosas como la consolidación de libros de trabajo de Excel y la producción de informes de Excel
- Utilice xlwings para crear herramientas interactivas de Excel que utilicen Python como motor de cálculo
- Conecte Excel a bases de datos y archivos CSV y obtenga datos de Internet usando código Python
- Use Python como una herramienta única para reemplazar VBA, Power Query y Power Pivot

"Este libro explica cómo puede integrar Python en Excel y liberarse del desastre inevitable

de enormes libros de trabajo, miles de fórmulas, y horribles hacks de VBA.

*Python para Excel* es probablemente el single el libro más útil sobre Excel que he leído y absolutamente imprescindible para cualquier usuario avanzado de Excel".

- Andreas F. Clenow  
CIO, Aries Asset Management y autor de best-sellers internacionales

*Siguiendo la tendencia, Acciones en movimiento, y Trading evolucionó*

Felix Zumstein es creador y mantenedor de xlwings, un popular paquete de código abierto que permite la automatización de Excel con Python en Windows y macOS. Como director ejecutivo de xltrail, un sistema de control de versiones para archivos de Excel, ha obtenido una visión profunda de los casos de uso típicos y los problemas con Excel en varias industrias.

---

DATOS / PY THON

US \$ 59,99

79,99 dólares canadienses

ISBN: 978-1-492-08100-5

Gorjeo: @oreillymedia  
[facebook.com/oreilly](http://facebook.com/oreilly)



9 781492 081005

## **Elogios para *Python para Excel***

¿Qué puede hacer Python por Excel? Si alguna vez se ha enfrentado a bloqueos inesperados de libros de trabajo, cálculos rotos y tediosos procesos manuales, querrá averiguarlo. *Python para*

*Excel* es una descripción general completa y concisa para comenzar con Python como usuario de una hoja de cálculo y crear productos de datos potentes con ambos. No dejes que el miedo a aprender a codificar te aleje: Felix proporciona una base excepcional para aprender Python de la que incluso los programadores experimentados podrían beneficiarse. Además, enmarca esta información de una manera que sea rápidamente accesible y aplicable a usted como usuario de Excel. Al leer este libro, puede decir rápidamente que fue escrito por alguien con años de experiencia en la enseñanza y el trabajo con clientes sobre cómo usar Excel en toda su extensión con la ayuda de Programación Python. Felix está especialmente indicado para mostrarte las posibilidades de aprendizaje.

Python para Excel; Espero que disfrutes de la clase magistral tanto como yo.

- George Mount, fundador, Stringfest Analytics

Python es la progresión natural de Excel y es tentador simplemente descartar Excel todos juntos. Tentador, pero poco realista. Excel está aquí, y aquí para quedarse, tanto en el mundo corporativo y como una herramienta de escritorio útil en el hogar y en la oficina. Este libro proporciona el puente tan necesario entre estos dos mundos. Explica como puedes integrar Python en Excel y libérese del inevitable desastre de enormes libros de trabajo, miles de fórmulas y horribles hacks de VBA. *Python para Excel* es probable el libro más útil en Excel que he leído y una lectura absolutamente obligada para cualquier usuario avanzado de Excel. ¡Un libro muy recomendable!

- Andreas F. Clenow, CIO Aries Asset Management y autor de best-sellers internacionales Siguiendo la tendencia, Acciones en movimiento, y Trading evolucionó

Excel sigue siendo una herramienta fundamental del mundo financiero, pero una gran cantidad de estos Excel las aplicaciones son un lío irresponsable. Este libro hace un excelente trabajo mostrándote cómo crear aplicaciones mejores y más robustas con la ayuda de xlwings.

*- Werner Brönnimann, especialista en Derivados y DeFi y cofundador de Ubinetic AG*

Excel y Python son dos de las herramientas más importantes de la caja de herramientas de Business Analytics, y juntas son mucho más grandes que la suma de sus partes. En este libro, Felix Zumstein expone su dominio incomparable de las muchas formas de conectar Python y Excel utilizando soluciones multiplataforma de código abierto. Será una herramienta invaluable para los analistas de negocios.

y científicos de datos por igual, y cualquier usuario de Python que busque aprovechar la poder de Excel en su código.

*- Daniel Guetta, profesor asociado de práctica profesional y director de Business Analytics Initiative en Columbia Business School y coautor de Python para MBA*

---

# **Python para Excel**

*Un entorno moderno para la automatización  
y análisis de datos*

***Felix Zumstein***

Pekín Boston Farnham Sebastopol Tokio

**O'REILLY®**

**Python para Excel**

por Felix Zumstein

Copyright © 2021 Zoomer Analytics LLC. Reservados todos los derechos.

Impreso en los Estados Unidos de América.

Publicado por O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

Los libros de O'Reilly se pueden comprar con fines educativos, comerciales o promocionales de ventas. Las ediciones en línea también están disponibles para la mayoría de los títulos (<http://oreilly.com>). Para obtener más información, comuníquese con nuestro departamento de ventas corporativo / institucional: 800-998-9938 [ocorporate@oreilly.com](mailto:ocorporate@oreilly.com).

**Editor de adquisiciones:** Michelle Smith

**Indexador:** nSight Inc.

**Editor de desarrollo:** Melissa Potter

**Diseñador de interiores:** David Futato

**Editor de producción:** Daniel Elfanbaum

**Diseñador de la portada:** Karen Montgomery

**Editor de copia:** Piper Editorial Consulting, LLC

**Ilustrador:** Kate Dullea

**Corrector de pruebas:** nSight Inc.

Marzo de 2021: Primera edición

**Historial de revisiones de la primera edición**

2021-03-04: primer lanzamiento

Ver <http://oreilly.com/catalog/errata.csp?isbn=9781492081005> para conocer los detalles de la versión.

El logotipo de O'Reilly es una marca registrada de O'Reilly Media, Inc. *Python para Excel*, la imagen de portada y la imagen comercial relacionada son marcas comerciales de O'Reilly Media, Inc.

Las opiniones expresadas en este trabajo son las del autor y no representan las opiniones del editor. Si bien el editor y el autor han realizado esfuerzos de buena fe para garantizar que la información y las instrucciones contenidas en este trabajo sean precisas, el editor y el autor renuncian a toda responsabilidad por errores u omisiones, incluida, entre otras, la responsabilidad por daños resultantes del uso de o dependencia de este trabajo. El uso de la información y las instrucciones contenidas en este trabajo es bajo su propio riesgo. Si alguna muestra de código u otra tecnología que este trabajo contiene o describe está sujeta a licencias de código abierto o derechos de propiedad intelectual de otros, es su responsabilidad asegurarse de que su uso cumpla con dichas licencias y / o derechos.

978-1-492-08100-5

[LSI]

---

# Tabla de contenido

Prefacio .....	xii
----------------	-----

---

## Parte I. Introducción a Python

<b>1. ¿Por qué Python para Excel? .....</b>	<b>3</b>
Excel es un lenguaje de programación	4
Excel en las noticias Mejores	5
prácticas de programación Excel	6
moderno	11
Python para Excel	12
Legibilidad y capacidad de mantenimiento	13
Administrador de paquetes y biblioteca estándar	14
Computación científica	15
Conclusión de compatibilidad	dieciséis
multiplataforma de las características del	17
lenguaje moderno	17
<b>2. Entorno de desarrollo. ....</b>	<b>19</b>
La distribución de Anaconda Python	20
Instalación	20
Aviso Anaconda	21
Python REPL: una sesión interactiva de Python	24
Administradores de paquetes: Conda y pip	25
Ambientes Conda	27
Cuadernos Jupyter	27
Ejecución de Jupyter Notebooks	28
Notebook Cells	29

---

El orden de ejecución del modo de edición frente al modo de comando importa	31
<b>Cerrar Jupyter Notebooks</b>	33
<b>Visual Studio Code</b>	34
Instalación y configuración	36
ejecutando un script de Python	37
Conclusión	41
<b>3. Introducción a Python.....</b>	<b>43</b>
Tipos de datos	43
Objetos	44
Tipos numéricos	45
Booleanos	47
Instrumentos de cuerda	49
Indexación y corte	50
Indexación	51
Rebanar	52
Estructuras de datos	52
Liza	53
Diccionarios	55
Tuplas	57
Conjuntos	57
Flujo de control	58
Bloques de código y declaración de paso	58
La instrucción if y las expresiones condicionales	59
Los bucles for y while	60
Organización de códigos de comprensiones de listas, diccionarios y conjuntos	63
Funciones	sesenta y cinco
Los módulos y la declaración de importación La clase datetime	66
PEP 8: Guía de estilo para código Python	69
Sugerencias de tipo de código PEP 8 y VS	70
Conclusión	73
	73
	73
	74
<b>Parte II. Introducción a los pandas</b>	
<b>4. Fundaciones NumPy.....</b>	<b>77</b>
Empezando con NumPy	77
Matriz NumPy	77

Funciones universales de vectorización y difusión (ufunc)	79
Creación y manipulación de matrices	80
Obtención y configuración de elementos de matriz Constructores de matriz útiles	81
Ver vs. Copiar	82
Conclusión	83
	83
	84
<b>5. Análisis de datos con pandas.....</b>	<b>85</b>
DataFrame y Series	85
Índice	88
Columnas	90
Manipulación de datos	91
Seleccionar datos	92
Configuración de datos	97
Datos perdidos	100
Datos duplicados	102
Operaciones aritméticas	103
Trabajar con columnas de texto	105
aplicando una función	105
Ver vs. Copiar	107
Combinando DataFrames	107
Concatenar	108
Unirse y fusionarse	109
Estadística descriptiva y agregación de datos	111
Estadísticas descriptivas	111
Agrupamiento	112
Girar y derretir	113
Graficado	115
Matplotlib	115
Plotly	117
Importación y exportación de marcos de datos	119
Exportación de archivos CSV	120
Importación de archivos CSV	121
Conclusión	123
<b>6. Análisis de series de tiempo con pandas.....</b>	<b>125</b>
DatetimeIndex	126
Creación de un índice de fecha y hora Filtrado de un índice de fecha y hora Trabajo con zonas horarias Manipulaciones comunes de series de tiempo	126
	128
	129
	131

---

Cambio y cambios porcentuales	131
Rebase y correlación	133
Remuestreo	136
Ventanas enrollables	137
Limitaciones con pandas	138
Conclusión	139

---

## **Parte III. Leer y escribir archivos de Excel sin Excel**

<b>7. Manipulación de archivos de Excel con pandas. ....</b>	<b>143</b>
Estudio de caso: Informes de Excel	143
Leer y escribir archivos de Excel con pandas	147
La función read_excel y la clase ExcelFile El	147
método to_excel y las limitaciones de la clase	152
ExcelWriter al usar pandas con archivos Excel	154
Conclusión	154
<b>8. Manipulación de archivos de Excel con paquetes Reader y Writer. ....</b>	<b>155</b>
Los paquetes Reader y Writer	155
Cuándo usar Qué paquete	156
El módulo excel.py	157
OpenPyXL	159
XlsxWriter	163
pyxlsb	165
Temas avanzados de lectura y	166
escritura de xlrd, xlwt y xlutils	169
Trabajo con archivos grandes de Excel Formateo de	169
marcos de datos en el estudio de caso de Excel	173
(revisado): Conclusión de informes de Excel	178
	179

---

## **Parte IV. Programación de la aplicación Excel con xlwings**

<b>9. Automatización de Excel. ....</b>	<b>183</b>
Empezando con xlwings	184
Uso de Excel como visor de datos	184
El modelo de objetos de Excel	186
Ejecutando código VBA	193
Convertidores, opciones y colecciones	194
Trabajar con DataFrames	195

---

Convertidores y opciones	196
Gráficos, imágenes y casos de estudio de nombres definidos (revisado nuevamente): Temas avanzados de xlwings sobre informes de Excel	198
Fundaciones xlwings	202
Mejorando el desempeño	204
Cómo solucionar la falta de funcionalidad	206
Conclusión	207
	208
<b>10. Herramientas de Excel impulsadas por Python.....</b>	<b>209</b>
Usando Excel como Frontend con xlwings	209
Complemento de Excel	210
Comando de inicio rápido	212
Ejecutar principal	212
Función RunPython	213
Despliegue	218
Dependencia de Python	218
Libros de trabajo independientes: deshacerse de la jerarquía de configuración del complemento xlwings	219
Ajustes	220
Conclusión	221
	222
<b>11. El rastreador de paquetes de Python.....</b>	<b>223</b>
Lo que construiremos	223
Funcionalidad central	226
API web	226
Bases de datos	229
Excepciones	238
Estructura de la aplicación	240
Interfaz	241
Backend	245
Depuración	248
Conclusión	250
<b>12. Funciones definidas por el usuario (UDF).....</b>	<b>251</b>
Introducción a las UDF	252
Inicio rápido de UDF	252
Estudio de caso: Tendencias de Google	257
Introducción a Tendencias de Google	257
Trabajar con DataFrames y Matrices dinámicas Obtención de datos de Tendencias de Google	258
Trazado con UDF	263
	267

Depurar UDF	269
Temas avanzados de UDF	271
Almacenamiento en caché de optimización del rendimiento básico	272
El decorador secundario	274
Conclusión	276
<b>UNA. Ambientes Conda</b>	<b>281</b>
<b>B. Funcionalidad avanzada de VS Code</b>	<b>285</b>
<b>C. Conceptos avanzados de Python</b>	<b>291</b>
<b>Índice</b>	<b>299</b>

---

## Prefacio

Microsoft está ejecutando un foro de comentarios para Excel en [UserVoice](#) donde todos pueden enviar una nueva idea para que otros la voten. La solicitud de función más votada es "Python como lenguaje de programación de Excel" y tiene aproximadamente el doble de votos que la segunda solicitud de función más votada. Aunque no sucedió nada desde que se agregó la idea en 2015, los usuarios de Excel se sintieron llenos de nuevas esperanzas a fines de 2020 cuando Guido van Rossum, el creador de Python, [tuiteó](#) que su "jubilación era aburrida" y que se uniría a Microsoft. Si su movimiento tiene alguna influencia en la integración de Excel y Python, no lo sé. Sin embargo, sí sé qué hace que esta combinación sea tan atractiva y cómo puede comenzar a usar Excel y Python juntos, hoy. Y esto es, en pocas palabras, de lo que trata este libro.

La principal fuerza impulsora detrás de la *Python para Excel*/historia es el hecho de que vivimos en un mundo de datos. Hoy en día, hay enormes conjuntos de datos disponibles para todos y para todo. A menudo, estos conjuntos de datos son tan grandes que ya no caben en una hoja de cálculo. Hace unos años, esto puede haber sido denominado *big data*, pero hoy en día, un conjunto de datos de unos pocos millones de filas no es nada especial. Excel ha evolucionado para hacer frente a esa tendencia: introdujo Power Query para cargar y limpiar conjuntos de datos que no caben en una hoja de cálculo y Power Pivot, un complemento para realizar análisis de datos en estos conjuntos de datos y presentar los resultados. Power Query se basa en el lenguaje de fórmulas de Power Query M (M), mientras que Power Pivot define fórmulas utilizando Expresiones de análisis de datos (DAX). Si también desea automatizar algunas cosas en su archivo de Excel, entonces usaría el lenguaje de automatización integrado de Excel, Visual Basic para Aplicaciones (VBA). Es decir, para algo bastante simple, puede terminar usando VBA, M y DAX. Un problema con esto es que todos estos lenguajes solo le sirven en el mundo de Microsoft, principalmente en Excel y Power BI (presentaré Power BI brevemente en [Capítulo 1](#)).

Python, por otro lado, es un lenguaje de programación de propósito general que se ha convertido en una de las opciones más populares entre analistas y científicos de datos. Si usa Python con Excel, puede usar un lenguaje de programación que sea bueno en todos los aspectos de la historia, ya sea automatizar Excel, acceder y preparar conjuntos de datos o realizar análisis de datos y tareas de visualización. Lo más importante es que puedes

reutilice sus habilidades de Python fuera de Excel: si necesita ampliar su poder de cálculo, puede mover fácilmente su modelo cuantitativo, simulación o aplicación de aprendizaje automático a la nube, donde los recursos informáticos prácticamente ilimitados lo están esperando.

## Por qué escribí este libro

A través de mi trabajo en xlwings, el paquete de automatización de Excel que encontraremos en [Parte IV](#) de este libro, estoy en estrecho contacto con muchos usuarios que utilizan Python para Excel - ya sea a través del [rastreador de problemas](#) en GitHub, una pregunta sobre [Desbordamiento de pila](#) o en un evento físico como una reunión o una conferencia.

De forma regular, se me pide que recomiende recursos para comenzar con Python. Si bien ciertamente no hay escasez de introducciones de Python, a menudo son demasiado generales (nada sobre el análisis de datos) o demasiado específicas (introducciones científicas completas). Sin embargo, los usuarios de Excel tienden a estar en algún punto intermedio: ciertamente trabajan con datos, pero una introducción científica completa puede ser demasiado técnica. También suelen tener requisitos específicos y preguntas que no se responden en ninguno de los materiales existentes. Algunas de estas preguntas son:

- ¿Qué paquete de Python-Excel necesito para qué tarea?
- ¿Cómo muevo mi conexión de base de datos de Power Query a Python?
- ¿Cuál es el equivalente del autofiltro de Excel o la tabla dinámica en Python?

Escribí este libro para que no tenga conocimientos de Python para poder automatizar sus tareas centradas en Excel y aprovechar las herramientas de análisis de datos y de computación científica de Python en Excel sin ningún desvío.

## Para quien es este libro

Si es un usuario avanzado de Excel que quiere superar los límites de Excel con un lenguaje de programación moderno, este libro es para usted. Por lo general, esto significa que pasas horas al mes descargando, limpiando y copiando / pegando grandes cantidades de datos en hojas de cálculo de misión crítica. Si bien hay diferentes formas de superar los límites de Excel, este libro se centrará en cómo usar Python para esta tarea.

Debe tener un conocimiento básico de programación: ayuda si ya ha escrito una función o un bucle for (sin importar en qué lenguaje de programación) y tiene una idea de lo que es un entero o una cadena. Incluso podría dominar este libro si está acostumbrado a escribir fórmulas de celdas complejas o si tiene experiencia con el ajuste de macros VBA grabadas. Sin embargo, no se espera que tenga ninguna experiencia específica de Python, ya que hay introducciones a todas las herramientas que usaremos, incluida una introducción a Python en sí.

Si usted es un desarrollador experimentado de VBA, encontrará comparaciones regulares entre Python y VBA que le permitirán sortear los errores comunes y comenzar a funcionar.

Este libro también puede ser útil si es un desarrollador de Python y necesita aprender acerca de las diferentes formas en que Python puede manejar la aplicación de Excel y los archivos de Excel para poder elegir el paquete correcto dados los requisitos de sus usuarios comerciales.

## Cómo está organizado este libro

En este libro, le mostraré todos los aspectos de la *Python para Excel*/historia dividida en cuatro partes:

### *Parte I: Introducción a Python*

Esta parte comienza analizando las razones por las que Python es un compañero tan agradable para Excel antes de presentar las herramientas que usaremos en este libro: la distribución de Anaconda Python, Visual Studio Code y los cuadernos de Jupyter. Esta parte también le enseñará suficiente Python para poder dominar el resto de este libro.

### *Parte II: Introducción a los pandas*

pandas es la biblioteca de referencia de Python para el análisis de datos. Aprenderemos cómo reemplazar los libros de Excel con una combinación de cuadernos de Jupyter y pandas. Por lo general, el código de pandas es más fácil de mantener y más eficiente que un libro de trabajo de Excel, y puede trabajar con conjuntos de datos que no caben en una hoja de cálculo. A diferencia de Excel, pandas te permite ejecutar tu código donde quieras, incluida la nube.

### *Parte III: Lectura y escritura de archivos de Excel sin Excel*

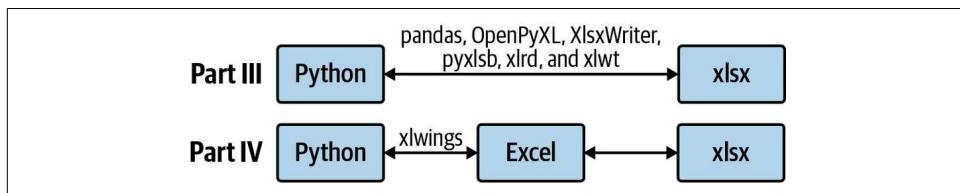
Esta parte trata sobre la manipulación de archivos de Excel mediante el uso de uno de los siguientes paquetes de Python: pandas, OpenPyXL, XlsxWriter, pyxlsb, xlrd y xlwt. Estos paquetes pueden leer y escribir libros de trabajo de Excel directamente en el disco y, como tales, reemplazan la aplicación de Excel: como no requiere una instalación de Excel, funcionan en cualquier plataforma compatible con Python, incluidos Windows, macOS y Linux. Un caso de uso típico de un paquete lector es leer datos de archivos de Excel que recibe cada mañana de una empresa o sistema externo y almacenar su contenido en una base de datos. Un caso de uso típico de un paquete de escritor es proporcionar la funcionalidad detrás del famoso botón "Exportar a Excel" que se encuentra en casi todas las aplicaciones.

### *Parte IV: Programación de la aplicación Excel con xlwings*

En esta parte, veremos cómo podemos usar Python con el paquete xlwings para automatizar la aplicación de Excel en lugar de leer y escribir archivos de Excel en el disco. Por lo tanto, esta parte requiere que tenga una instalación local de Excel. Aprenderemos a abrir libros de Excel y a manipularlos delante de nuestros ojos. Además de leer y escribir archivos a través de Excel, crearemos Excel interactivo

herramientas: nos permiten hacer clic en un botón para que Python realice algo que usted pudo haber hecho anteriormente con macros VBA, como un cálculo computacionalmente costoso. También aprenderemos a escribir funciones definidas por el usuario.<sup>1</sup> (UDF) en Python en lugar de VBA.

Es importante comprender la diferencia fundamental entre leer y escribir Excel *archivos* ([Parte III](#)) y programar el Excel *solicitud* ([Parte IV](#)) como se visualiza en [Figura P-1](#).



*Figura P-1. Leer y escribir archivos de Excel (Parte III) vs. programar Excel (Parte IV)*

Ya que [Parte III](#) no requiere una instalación de Excel, todo funciona en todas las plataformas compatibles con Python, principalmente Windows, macOS y Linux. [Parte IV](#), sin embargo, solo funcionará en aquellas plataformas compatibles con Microsoft Excel, es decir, Windows y macOS, ya que el código se basa en una instalación local de Microsoft Excel.

## Versiones de Python y Excel

Este libro está basado en Python 3.8, que es la versión de Python que viene con la última versión de la distribución de Anaconda Python en el momento de escribir este artículo. Si desea utilizar una versión más reciente de Python, siga las instrucciones en [el página de inicio del libro](#), pero asegúrese de no utilizar una versión anterior. Ocasionalmente haré un comentario si algo cambia con Python 3.9.

Este libro también espera que utilice una versión moderna de Excel, es decir, al menos Excel 2007 en Windows y Excel 2016 en macOS. La versión de Excel instalada localmente que viene con la suscripción a Microsoft 365 también funcionará perfectamente; de hecho, incluso la recomiendo, ya que tiene las funciones más recientes que no encontrará en otras versiones de Excel. También fue la versión que usé para escribir este libro, por lo que si usa otra versión de Excel, a veces puede ver una pequeña diferencia en el nombre o la ubicación de un elemento del menú.

---

<sup>1</sup> Microsoft ha comenzado a utilizar el término *funciones personalizadas* en lugar de UDF. En este libro, seguiré llamando ellos UDF.

## Las convenciones usadas en este libro

En este libro se utilizan las siguientes convenciones tipográficas:

### *Itálico*

Indica nuevos términos, URL, direcciones de correo electrónico, nombres de archivo y extensiones de archivo.

### Ancho constante

Se utiliza para listas de programas, así como dentro de párrafos para referirse a elementos de programa como nombres de variables o funciones, bases de datos, tipos de datos, variables de entorno, declaraciones y palabras clave.

### **Ancho constante en negrita**

Muestra comandos u otro texto que el usuario debe escribir literalmente.

### *Cursiva de ancho constante*

Muestra texto que debe ser reemplazado por valores proporcionados por el usuario o por valores determinados por contexto.



Este elemento significa un consejo o sugerencia.



Este elemento significa una nota general.



Este elemento indica una advertencia o precaución.

## Usar ejemplos de código

Estoy manteniendo un [Página web](#) con información adicional para ayudarle con este libro. Asegúrese de revisarlo, especialmente si tiene algún problema.

El material complementario (ejemplos de código, ejercicios, etc.) está disponible para descargar en <https://github.com/fzumstein/python-for-excel>. Para descargar este repositorio complementario, haga clic en el botón verde Código, luego seleccione Descargar ZIP. Una vez descargado, haga clic derecho en el archivo en Windows y seleccione Extraer todo para descomprimir los archivos contenidos en una carpeta. En macOS, simplemente haga doble clic en el archivo para descomprimirlo. Si sabes trabajar

con Git, también puede usar Git para clonar el repositorio en su disco duro local. Puede colocar la carpeta en cualquier lugar que desee, pero me referiré a ella ocasionalmente de la siguiente manera en este libro:

```
C:\ Usuarios \nombre de usuario\ python-para-excel
```

Simplemente descargando y descomprimiendo el archivo ZIP en Windows, terminará con una estructura de carpetas similar a esta (tenga en cuenta los nombres de carpeta repetidos):

```
C:\ ... \ Descargas \ python-for-excel-1st-edition \ python-for-excel-1st-edition
```

Copiar el contenido de esta carpeta en una que cree en *C:\ Users \ <nombre de usuario> \ python-for-excel* podría facilitarle el seguimiento. Las mismas observaciones son válidas para macOS, es decir, copie los archivos *a/Usuarios / <nombre de usuario> / python-for-excel*.

Si tiene una pregunta técnica o un problema al utilizar los ejemplos de código, envíe un correo electrónico a [bookquestions@oreilly.com](mailto:bookquestions@oreilly.com).

Este libro está aquí para ayudarlo a hacer su trabajo. En general, si se ofrece un código de ejemplo con este libro, puede utilizarlo en sus programas y documentación. No es necesario que se comunique con nosotros para obtener permiso a menos que esté reproduciendo una parte significativa del código. Por ejemplo, escribir un programa que utiliza varios fragmentos de código de este libro no requiere permiso. Vender o distribuir ejemplos de libros de O'Reilly requiere permiso. Responder una pregunta citando este libro y citando un código de ejemplo no requiere permiso. La incorporación de una cantidad significativa de código de ejemplo de este libro en la documentación de su producto requiere permiso.

Apreciamos la atribución, pero generalmente no la exigimos. Una atribución generalmente incluye el título, el autor, el editor y el ISBN. Por ejemplo: "Python para Excel por Felix Zumstein (O'Reilly). Copyright 2021 Zoomer Analytics LLC, 978-1-492-08100-5".

Si cree que el uso que hace de los ejemplos de código está fuera del uso legítimo o del permiso otorgado anteriormente, no dude en contactarnos en [permissions@oreilly.com](mailto:permissions@oreilly.com).

## Aprendizaje en Línea O'Reilly



Durante más de 40 años, *O'Reilly Media* ha proporcionado capacitación, conocimiento y conocimientos en tecnología y negocios para ayudar a las empresas a tener éxito.

Nuestra red única de expertos e innovadores comparte su conocimiento y experiencia a través de libros, artículos y nuestra plataforma de aprendizaje en línea. La plataforma de aprendizaje en línea de O'Reilly le brinda acceso bajo demanda a cursos de capacitación en vivo, rutas de aprendizaje en profundidad, entornos de codificación interactivos y una amplia colección de texto y video de O'Reilly y más de 200 editores. Para más información visite <http://oreilly.com>.

## Cómo contactarnos

Dirija sus comentarios y preguntas sobre este libro al editor:

O'Reilly Media, Inc.  
1005 Gravenstein Highway North  
Sebastopol, CA 95472  
800-998-9938 (en los Estados Unidos o Canadá)  
707-829-0515 (internacional o local)  
707-829-0104 (fax)

Tenemos una página web para este libro, donde enumeramos erratas, ejemplos y cualquier información adicional. Puede acceder a esta página en <https://oreil.ly/py4excel>.

Correo electrónico [bookquestions@oreilly.com](mailto:bookquestions@oreilly.com) para comentar o hacer preguntas técnicas sobre este libro.

Para obtener más información sobre nuestros libros, cursos, conferencias y noticias, visite nuestro sitio web en <http://www.oreilly.com>.

Encuentranos en Facebook: <http://facebook.com/oreilly>. Síguenos

en Twitter: <http://twitter.com/oreillymedia>. Míranos en YouTube:

<http://www.youtube.com/oreillymedia>.

## Expresiones de gratitud

Como autor por primera vez, estoy increíblemente agradecido por la ayuda que recibí de tantas personas en el camino. ¡Ellos hicieron este viaje mucho más fácil para mí!

En O'Reilly, me gustaría agradecer a mi editora, Melissa Potter, quien hizo un gran trabajo manteniéndome motivada y dentro del programa y quien me ayudó a convertir este libro en una forma legible. También me gustaría agradecer a Michelle Smith, que trabajó conmigo en la propuesta del libro inicial, y a Daniel Elfanbaum, que nunca se cansó de responder a mis preguntas técnicas.

Un gran agradecimiento a todos mis colegas, amigos y clientes que invirtieron muchas horas en leer las primeras versiones de mis borradores. Sus comentarios fueron cruciales para hacer que el libro fuera más fácil de entender, y algunos de los estudios de casos están inspirados en problemas de Excel del mundo real que compartieron conmigo. Mi agradecimiento a Adam Rodríguez, Mano Beeslar, Simon Schiegg, Rui Da Costa, Jürg Nager y Christophe de Montrichard.

También recibí comentarios útiles de los lectores de la versión de lanzamiento anticipado que se publicó en la plataforma de aprendizaje en línea de O'Reilly. ¡Gracias Felipe Maion, Ray Doue, Kolyu Minevski, Scott Drummond, Volker Roth y David Ruggles!

Tuve mucha suerte de que el libro fuera revisado por revisores de tecnología altamente calificados y realmente aprecio el arduo trabajo que pusieron bajo mucha presión de tiempo. ¡Gracias por toda su ayuda, Jordan Goldmeier, George Mount, Andreas Clenow, Werner Brönnimann y Eric Moreira!

Un agradecimiento especial para Björn Stiel, que no era solo un revisor de tecnología, sino de quien también aprendí muchas de las cosas sobre las que estoy escribiendo en este libro. ¡He disfrutado trabajando contigo estos últimos años!

Por último, pero no menos importante, me gustaría extender mi gratitud a Eric Reynolds, quien fusionó su proyecto ExcelPython en la base de código xlwings en 2016. También rediseñó todo el paquete desde cero, haciendo de mi horrible API de los primeros días una cosa de el pasado. ¡Muchos gracias!

# **Introducción a Python**



## **¿Por qué Python para Excel?**

Por lo general, los usuarios de Excel comienzan a cuestionar sus herramientas de hoja de cálculo cuando encuentran una limitación. Un ejemplo clásico es cuando los libros de Excel contienen tantos datos y fórmulas que se vuelven lentos o, en el peor de los casos, se bloquean. Sin embargo, tiene sentido cuestionar su configuración antes de que las cosas vayan mal: si trabaja en libros de trabajo de misión crítica donde los errores pueden resultar en daños financieros o de reputación o si pasa horas todos los días actualizando libros de Excel manualmente, debe aprender cómo automatizar sus procesos con un lenguaje de programación. La automatización elimina el riesgo de errores humanos y le permite dedicar su tiempo a tareas más productivas que copiar / pegar datos en una hoja de cálculo de Excel.

En este capítulo, le daré algunas razones por las que Python es una excelente opción en combinación con Excel y cuáles son sus ventajas en comparación con el lenguaje de automatización integrado de Excel, VBA. Después de presentar Excel como lenguaje de programación y comprender sus particularidades, señalaré las características específicas que hacen que Python sea mucho más fuerte en comparación con VBA. Sin embargo, para empezar, ¡echemos un vistazo rápido a los orígenes de nuestros dos personajes principales!

En términos de tecnología informática, Excel y Python han existido durante mucho tiempo: Excel fue lanzado por primera vez en 1985 por Microsoft, y esto puede sorprender a muchos, solo estaba disponible para Apple Macintosh. No fue hasta 1987 que Microsoft Windows obtuvo su primera versión en forma de Excel 2.0. Sin embargo, Microsoft no fue el primer jugador en el mercado de las hojas de cálculo: VisiCorp salió con VisiCalc en 1979, seguido de Lotus Software en 1983 con Lotus 1-2-3. Y Microsoft no lideró con Excel: tres años antes, lanzaron Multiplan, un programa de hoja de cálculo que podría usarse en MS-DOS y algunos otros sistemas operativos, pero no en Windows.

Python nació en 1991, solo seis años después de Excel. Si bien Excel se hizo popular desde el principio, Python tardó un poco más en ser adoptado en ciertas áreas como el desarrollo web o la administración de sistemas. En 2005, Python comenzó a convertirse en un serio

alternativa para la informática científica cuando *NumPy*, se lanzó por primera vez un paquete para cálculo basado en matrices y álgebra lineal. NumPy combinó dos paquetes predecesores y, por lo tanto, agilizó todos los esfuerzos de desarrollo en torno a la computación científica en un solo proyecto. Hoy en día, constituye la base de innumerables paquetes científicos, que incluyen *pandas*, que salió en 2008 y que es en gran parte responsable de la adopción generalizada de Python en el mundo de la ciencia de datos y las finanzas que comenzó a suceder después de 2010. Gracias a los *pandas*, Python, junto con R, se ha convertido en uno de los más comunes utilizados lenguajes para tareas de ciencia de datos como análisis de datos, estadísticas y aprendizaje automático.

El hecho de que Python y Excel se inventaron hace mucho tiempo no es lo único que tienen en común: Excel y Python son también un lenguaje de programación. Si bien probablemente no se sorprenda al escuchar eso sobre Python, es posible que requiera una explicación para Excel, que le daré a continuación.

## **Excel es un lenguaje de programación**

Esta sección comienza con la introducción de Excel como lenguaje de programación, lo que le ayudará a comprender por qué los problemas relacionados con las hojas de cálculo aparecen en las noticias con regularidad. Luego, veremos algunas de las mejores prácticas que han surgido en la comunidad de desarrollo de software y que pueden salvarlo de muchos errores típicos de Excel. Concluiremos con una breve introducción a Power Query y Power Pivot, dos herramientas modernas de Excel que cubren el tipo de funcionalidad para la que usaremos *pandas*.

Si usa Excel para más que su lista de compras, definitivamente está usando funciones como =SUMA(A1:A4) para resumir un rango de celdas. Si piensa por un momento en cómo funciona esto, notará que el valor de una celda generalmente depende de una o más celdas, que nuevamente pueden usar funciones que dependen de una o más celdas, y así sucesivamente. Hacer tales llamadas a funciones anidadas no es diferente de cómo funcionan otros lenguajes de programación, solo que usted escribe el código en celdas en lugar de archivos de texto. Y si eso no te ha convencido todavía: a finales de 2020, Microsoft anunció la introducción de *funciones lambda*, que le permiten escribir funciones reutilizables en el propio lenguaje de fórmulas de Excel, es decir, sin tener que depender de un lenguaje diferente como VBA. Según Brian Jones, jefe de producto de Excel, esta fue la pieza que faltaba y que finalmente convierte a Excel en un lenguaje de programación "real".<sup>1</sup> Esto también significa que los usuarios de Excel realmente deberían llamarse programadores de Excel.

Sin embargo, hay algo especial acerca de los programadores de Excel: la mayoría de ellos son usuarios comerciales o expertos en dominios sin una educación formal en informática. Son comerciantes, contables o ingenieros, por mencionar solo algunos ejemplos. Sus herramientas de hoja de cálculo están diseñadas para resolver un problema comercial y, a menudo, ignoran las mejores prácticas en

---

1 Puede leer el anuncio de las funciones lambda en la [Blog de Excel](#).

desarrollo de software. Como consecuencia, estas herramientas de hoja de cálculo a menudo combinan entradas, cálculos y resultados en las mismas hojas, pueden requerir la realización de pasos no obvios para que funcionen correctamente y los cambios críticos se realizan sin ninguna red de seguridad. En otras palabras, las herramientas de hoja de cálculo carecen de una arquitectura de aplicación sólida y, a menudo, no están documentadas ni probadas. A veces, estos problemas pueden tener consecuencias devastadoras: si olvida volver a calcular su libro de operaciones antes de realizar una operación, es posible que compre o venda la cantidad incorrecta de acciones, lo que puede hacer que pierda dinero. Y si no es solo su propio dinero lo que está negociando, podemos leerlo en las noticias, como veremos a continuación.

## Excel en las noticias

Excel es un invitado habitual en las noticias, y durante el transcurso de este escrito, dos nuevas historias llegaron a los titulares. El primero fue sobre el Comité de Nomenclatura Genética HUGO, que cambió el nombre de algunos genes humanos para que Excel ya no los interprete como fechas. Por ejemplo, para evitar que el genMARZO 1 se convertiría en 1-mar, fue renombrado a MARZO F1.<sup>2</sup> En la segunda historia, se culpó a Excel por el retraso en la notificación de 16.000 resultados de pruebas de COVID-19 en Inglaterra. El problema se debió a que los resultados de la prueba se escribieron en el formato de archivo de Excel anterior (.xls) que se limitó a aproximadamente 65.000 filas. Esto significó que los conjuntos de datos más grandes simplemente se cortaron más allá de ese límite.<sup>3</sup> Si bien estas dos historias muestran la importancia y el dominio continuos de Excel en el mundo actual, probablemente no haya otro "incidente de Excel" que sea más famoso que la ballena de Londres.

*Ballena de Londres* es el apodo de un comerciante cuyos errores comerciales obligaron a JP Morgan a anunciar una pérdida asombrosa de \$ 6 mil millones en 2012. La fuente de la explosión fue un modelo de valor en riesgo basado en Excel que subestimaba sustancialmente el verdadero riesgo de perder dinero en una de sus carteras. *los Informe del grupo de trabajo de gestión de JPMorgan Chase & Co. con respecto a las pérdidas de CIO de 2012*<sup>4</sup> (2013) menciona que "el modelo operaba a través de una serie de hojas de cálculo de Excel, las cuales debían ser completadas manualmente, mediante un proceso de copiar y pegar datos de una hoja de cálculo a otra". Además de estos problemas operativos, tenían un error lógico: en un cálculo, estaban dividiendo por una suma en lugar de un promedio.

Si desea ver más de estas historias, eche un vistazo a [Historias de terror](#), una página web mantenida por el European Spreadsheet Risks Interest Group (EuSpRIG).

---

<sup>2</sup> James Vincent, "Los científicos cambian el nombre de los genes humanos para evitar que Microsoft Excel los malinterprete como fechas", *los Bordes*, 6 de agosto de 2020, <https://oreil.ly/0qo-n>.

<sup>3</sup> Leo Kelion, "Excel: Por qué el uso de la herramienta de Microsoft hizo que se perdieran los resultados de COVID-19", *noticias de la BBC*, 5 de octubre, 2020, <https://oreil.ly/vvB6o>.

<sup>4</sup> Enlaces de Wikipedia al documento en uno de los [notas al pie](#) en su artículo sobre el caso.

Para evitar que su empresa termine en las noticias con una historia similar, echemos un vistazo a algunas de las mejores prácticas a continuación que hacen que su trabajo con Excel sea mucho más seguro.

## Mejores prácticas de programación

Esta sección le presentará las mejores prácticas de programación más importantes, incluida la separación de preocupaciones, el principio DRY, las pruebas y el control de versiones. Como veremos, seguirlos será más fácil cuando empieces a usar Python junto con Excel.

### Separación de intereses

Uno de los principios de diseño más importantes en programación es *separación de intereses*, a veces también conocido como *modularidad*. Significa que una parte independiente del programa debe encargarse de un conjunto relacionado de funcionalidades para que pueda ser reemplazado fácilmente sin afectar al resto de la aplicación. En el nivel más alto, una aplicación a menudo se divide en las siguientes capas:<sup>5</sup>

- Capa de presentación
- Capa empresarial
- Capa de datos

Para explicar estas capas, considere un conversor de moneda simple como el que se muestra en [Figura 1-1](#). Encontrarás el `currency_converter.xlsx` Archivo de Excel en el `SG` carpeta del repositorio complementario.

Así es como funciona la aplicación: escriba el monto y la moneda en las celdas A4 y B4, respectivamente, y Excel lo convertirá en dólares estadounidenses en la celda D4. Muchas aplicaciones de hojas de cálculo siguen este diseño y las empresas las utilizan todos los días. Permítanme dividir la aplicación en sus capas:

#### *Capa de presentación*

Esto es lo que ve y con lo que interactúa, es decir, la interfaz de usuario: los valores de las celdas A4, B4 y D4 junto con sus etiquetas forman la capa de presentación del convertidor de moneda.

#### *Capa empresarial*

Esta capa se encarga de la lógica específica de la aplicación: la celda D4 define cómo se convierte la cantidad a USD. La fórmula `=A4 * BUSCARV(B4, F4: G11, 2, FALSO)` se traduce en Monto multiplicado por el tipo de cambio.

---

<sup>5</sup> La terminología se toma de *Guía de arquitectura de aplicaciones de Microsoft, 2.a edición*, que está disponible [en línea](#).

### Capa de datos

Como sugiere el nombre, esta capa se encarga de acceder a los datos: la BUSCARV parte de la celda D4 está haciendo este trabajo.

La capa de datos accede a los datos de la tabla de tipos de cambio que comienza en la celda F3 y que actúa como la base de datos de esta pequeña aplicación. Si prestó mucha atención, probablemente notó que la celda D4 aparece en las tres capas: esta sencilla aplicación combina las capas de presentación, negocios y datos en una sola celda.

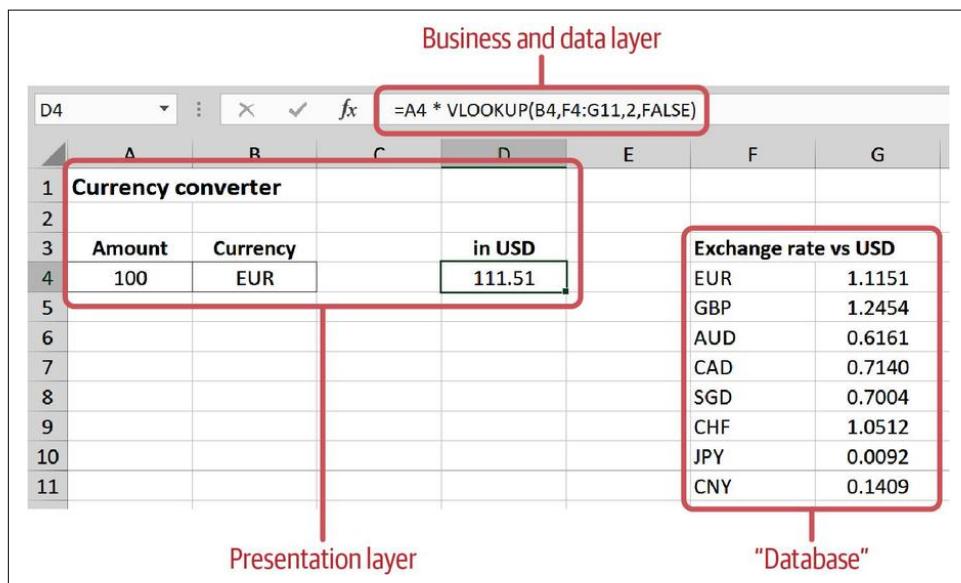


Figura 1-1. currency\_converter.xlsx

Esto no es necesariamente un problema para este simple convertidor de moneda, pero a menudo, lo que comienza como un pequeño archivo de Excel se convierte muy pronto en una aplicación mucho más grande. ¿Cómo se puede mejorar esta situación? La mayoría de los recursos profesionales para desarrolladores de Excel le aconsejan que utilice una hoja separada para cada capa, en la terminología de Excel que suele denominarse *entradas, cálculos, y salidas*. A menudo, esto se combina con la definición de un cierto código de color para cada capa, por ejemplo, un fondo azul para todas las celdas de entrada. En [Capítulo 11](#), construiremos una aplicación real basada en estas capas: Excel será la capa de presentación, mientras que las capas comerciales y de datos se trasladarán a Python, donde es mucho más fácil estructurar su código correctamente.

Ahora que sabe lo que significa la separación de preocupaciones, ¡descubramos qué es el principio DRY!

## **Principio SECO**

*El programador pragmático* por Hunt y Thomas (Pearson Education) popularizaron el principio DRY: *no te repitas*. Ningún código duplicado significa menos líneas de código y menos errores, lo que hace que el código sea más fácil de mantener. Si su lógica empresarial se encuentra en las fórmulas de su celda, es prácticamente imposible aplicar el principio DRY, ya que no existe ningún mecanismo que le permita reutilizarlo en otro libro de trabajo. Desafortunadamente, esto significa que una forma común de iniciar un nuevo proyecto de Excel es copiar el libro de trabajo del proyecto anterior o de una plantilla.

Si escribe VBA, la parte más común de código reutilizable es una función. Una función le da acceso al mismo bloque de código desde múltiples macros, por ejemplo. Si tiene varias funciones que usa todo el tiempo, es posible que desee compartirlas entre libros de trabajo. El instrumento estándar para compartir código VBA entre libros de trabajo son los complementos, pero los complementos VBA carecen de una forma sólida de distribuirlos y actualizarlos. Si bien Microsoft ha introducido una tienda de complementos internos de Excel para resolver ese problema, esto solo funciona con complementos basados en JavaScript, por lo que no es una opción para los codificadores VBA. Esto significa que todavía es muy común usar el enfoque de copiar / pegar con VBA: supongamos que necesita una *interpolación cúbica* función en Excel. La función spline cúbica es una forma de interpolar una curva basada en algunos puntos dados en un sistema de coordenadas y, a menudo, los operadores de renta fija la utilizan para derivar una curva de tasa de interés para todos los vencimientos basándose en algunas combinaciones conocidas de vencimiento / tasa de interés. Si busca "Cubic Spline Excel" en Internet, no pasará mucho tiempo hasta que tenga una página de código VBA que haga lo que desea. El problema con esto es que, por lo general, estas funciones fueron escritas por una sola persona con probablemente buenas intenciones, pero sin documentación o pruebas formales. Tal vez funcionen para la mayoría de las entradas, pero ¿qué pasa con algunos casos extremos poco comunes? Si está negociando una cartera de renta fija multimillonaria, quiere tener algo en lo que sabe que puede confiar. Por lo menos,

Python facilita la distribución de código mediante el uso de un administrador de paquetes, como veremos en la última sección de este capítulo. Sin embargo, antes de llegar allí, continuemos con las pruebas, una de las piedras angulares del desarrollo de software sólido.

## **Pruebas**

Cuando le dice a un desarrollador de Excel que pruebe sus libros de trabajo, lo más probable es que realice algunas verificaciones aleatorias: haga clic en un botón y vea si la macro todavía hace lo que se supone que debe hacer o cambie algunas entradas y verifique si la salida se ve razonable. Sin embargo, esta es una estrategia arriesgada: Excel facilita la introducción de errores que son difíciles de detectar. Por ejemplo, puede sobrescribir una fórmula con un valor codificado. O te olvidas de ajustar una fórmula en una columna oculta.

Cuando le dice a un desarrollador de software profesional que pruebe su código, escribirá *pruebas unitarias*. Como sugiere el nombre, es un mecanismo para probar componentes individuales de su programa. Por ejemplo, las pruebas unitarias aseguran que una sola función de un programa funcione correctamente. La mayoría de los lenguajes de programación ofrecen una forma de ejecutar pruebas unitarias automáticamente. La ejecución de pruebas automatizadas aumentará drásticamente la confiabilidad de su base de código y garantizará razonablemente que no romperá nada que funcione actualmente cuando edite su código.

Si observa la herramienta de conversión de moneda en [Figura 1-1](#), podría escribir una prueba que verifique si la fórmula en la celda D4 devuelve correctamente USD 105 con las siguientes entradas: 100 EUR como monto y 1.05 como tipo de cambio EURUSD. ¿Por qué ayuda esto? Suponga que borra accidentalmente la celda D4 con la fórmula de conversión y tiene que volver a escribirla: en lugar de multiplicar la cantidad por el tipo de cambio, divide por ella - después de todo, trabajar con monedas puede resultar confuso. Cuando ejecute la prueba anterior, obtendrá una prueba fallida ya que 100 EUR / 1.05 ya no dará como resultado 105 USD como espera la prueba. De esta forma, puede detectar y corregir la fórmula antes de entregar la hoja de cálculo a sus usuarios.

Casi todos los lenguajes de programación tradicionales ofrecen uno o más marcos de prueba para escribir pruebas unitarias sin mucho esfuerzo, pero no Excel. Afortunadamente, el concepto de pruebas unitarias es bastante simple y al conectar Excel con Python, obtienes acceso a los poderosos marcos de pruebas unitarias de Python. Si bien una presentación más detallada de las pruebas unitarias está más allá del alcance de este libro, los invito a echar un vistazo a [mi entrada en el blog](#), en el que les guío a través del tema con ejemplos prácticos.

Las pruebas unitarias a menudo se configuran para ejecutarse automáticamente cuando envías tu código a tu sistema de control de versiones. La siguiente sección explica qué son los sistemas de control de versiones y por qué son difíciles de usar con archivos de Excel.

#### **Control de versiones**

Otra característica de los programadores profesionales es que utilizan un sistema para *control de versiones* o *fuente de control*. Así como un sistema de control de versiones (VCS) rastrea los cambios en su código fuente a lo largo del tiempo, lo que le permite ver quién cambió qué, cuándo y por qué, y le permite volver a versiones anteriores en cualquier momento. El sistema de control de versiones más popular hoy en día es [Git](#). Originalmente se creó para administrar el código fuente de Linux y desde entonces ha conquistado el mundo de la programación; incluso Microsoft adoptó Git en 2017 para administrar el código fuente de Windows. En el mundo de Excel, por el contrario, el sistema de control de versiones más popular viene en forma de una carpeta donde los archivos se archivan así:

```
currency_converter_v1.xlsx  
currency_converter_v2_2020_04_21.xlsx  
currency_converter_final_edits_Bob.xlsx  
currency_converter_final_final.xlsx
```

Si, a diferencia de esta muestra, el desarrollador de Excel se adhiere a una determinada convención en el nombre del archivo, no hay nada intrínsecamente malo en eso. Pero mantener un historial de versiones de sus archivos localmente lo excluye de aspectos importantes del control de fuente en forma de colaboración más fácil, revisiones de pares, procesos de aprobación y registros de auditoría. Y si desea que sus libros de trabajo sean más seguros y estables, no querrá perderse estas cosas. Por lo general, los programadores profesionales usan Git en conexión con una plataforma basada en web como GitHub, GitLab, Bitbucket o Azure DevOps. Estas plataformas le permiten trabajar con los llamados *solicitudes de extracción* o *fusionar solicitudes*. Permiten a los desarrolladores solicitar formalmente que sus cambios se fusionen en la base de código principal. Una solicitud de extracción ofrece la siguiente información:

- Quién es el autor de los cambios
- ¿Cuándo se hicieron los cambios?
- ¿Cuál es el propósito de los cambios descritos en el *cometer mensaje*
- ¿Cuáles son los detalles de los cambios que se muestran en *vista de diferencias*, es decir, una vista que resalta los cambios en verde para el código nuevo y rojo para el código eliminado

Esto permite que un compañero de trabajo o un jefe de equipo revise los cambios y detecte irregularidades. A menudo, un par de ojos extra podrá detectar uno o dos fallos o dar una retroalimentación valiosa al programador. Con todas estas ventajas, ¿por qué los desarrolladores de Excel prefieren utilizar el sistema de archivos local y su propia convención de nomenclatura en lugar de un sistema profesional como Git?

- Muchos usuarios de Excel simplemente no conocen Git o se dan por vencidos desde el principio, ya que Git tiene una curva de aprendizaje relativamente empinada.
- Git permite que varios usuarios trabajen en copias locales del mismo archivo en paralelo. Después de que todos ellos comprometen su trabajo, Git generalmente puede fusionar todos los cambios sin ninguna intervención manual. Esto no funciona para los archivos de Excel: si se cambian en paralelo en copias separadas, Git no sabe cómo fusionar estos cambios en un solo archivo.
- Incluso si logra lidiar con los problemas anteriores, Git simplemente no ofrece tanto valor con los archivos de Excel como lo hace con los archivos de texto: Git no puede mostrar cambios entre los archivos de Excel, lo que impide un proceso de revisión por pares adecuado.

Debido a todos estos problemas, mi empresa ha creado [xtrail](#), un sistema de control de versiones basado en Git que sabe cómo manejar archivos de Excel. Oculta la complejidad de Git para que los usuarios empresariales se sientan cómodos usándolo y también le permite conectarse a sistemas Git externos, en caso de que ya esté rastreando sus archivos con GitHub, por ejemplo. xtrail rastrea los diferentes componentes de un libro de trabajo, incluidas las fórmulas de celda, rangos con nombre, Power Queries y código VBA, lo que le permite aprovechar los beneficios clásicos del control de versiones, incluidas las revisiones por pares.

Otra opción para facilitar el control de versiones con Excel es mover su lógica de negocios de Excel a archivos de Python, algo que haremos en [Capítulo 10](#). Como los archivos de Python son fáciles de rastrear con Git, tendrás la parte más importante de tu herramienta de hoja de cálculo bajo control.

Si bien esta sección se llama Mejores prácticas de programación, principalmente señala por qué son más difíciles de seguir con Excel que con un lenguaje de programación tradicional como Python. Antes de centrar nuestra atención en Python, me gustaría presentar brevemente Power Query y Power Pivot, el intento de Microsoft de modernizar Excel.

## **Excel moderno**

La era moderna de Excel comenzó con Excel 2007 cuando el menú de la cinta y los nuevos formatos de archivo (por ejemplo, *xlsx* en lugar de *xls*) fueron introducidos. Sin embargo, la comunidad de Excel usa *Excel moderno* para referirse a las herramientas que se agregaron con Excel 2010: lo más importante, Power Query y Power Pivot. Le permiten conectarse a fuentes de datos externas y analizar datos que son demasiado grandes para caber en una hoja de cálculo. Como su funcionalidad se superpone con lo que haremos con los pandas en [Capítulo 5](#), Los presentaré brevemente en la primera parte de esta sección. La segunda parte trata sobre Power BI, que podría describirse como una aplicación de inteligencia empresarial independiente que combina la funcionalidad de Power Query y Power Pivot con capacidades de visualización, ¡y tiene soporte integrado para Python!

## **Power Query y Power Pivot**

Con Excel 2010, Microsoft introdujo un complemento llamado *Consulta de energía*. Power Query se conecta a una multitud de fuentes de datos, incluidos libros de trabajo de Excel, archivos CSV y bases de datos SQL. También ofrece conexiones a plataformas como Salesforce e incluso se puede ampliar para conectarse con sistemas que no están cubiertos de forma inmediata. La funcionalidad principal de Power Query se ocupa de conjuntos de datos que son demasiado grandes para caber en una hoja de cálculo. Después de cargar los datos, puede realizar pasos adicionales para limpiarlos y manipularlos para que lleguen en una forma utilizable en Excel. Por ejemplo, podría dividir una columna en dos, fusionar dos tablas o filtrar y agrupar sus datos. Desde Excel 2016, Power Query ya no es un complemento, pero se puede acceder a él directamente en la pestaña Datos de la cinta a través del botón Obtener datos. Power Query solo está disponible parcialmente en macOS; sin embargo, se está desarrollando activamente, por lo que debería ser totalmente compatible en una versión futura de Excel.

*Power Pivot* va de la mano con Power Query: conceptualmente, es el segundo paso después de adquirir y limpiar sus datos con Power Query. Power Pivot le ayuda a analizar y presentar sus datos de forma atractiva directamente en Excel. Piense en ello como una tabla dinámica tradicional que, como Power Query, puede manejar grandes conjuntos de datos. Power Pivot le permite definir modelos de datos formales con relaciones y jerarquías, y puede agregar columnas calculadas a través del lenguaje de fórmulas DAX. Power Pivot también fue

introducido con Excel 2010, pero sigue siendo un complemento y hasta ahora no está disponible en macOS.

Si le gusta trabajar con Power Query y Power Pivot y desea crear paneles sobre ellos, puede que valga la pena echarle un vistazo a Power BI. ¡Vemos por qué!

#### **Power BI**

*Power BI* es una aplicación independiente que se lanzó en 2015. Es la respuesta de Microsoft a las herramientas de inteligencia empresarial como Tableau o Qlik. Power BI Desktop es gratuito, así que si quiere jugar con él, vaya a la [Página de inicio de Power BI](#) y descárguelo; tenga en cuenta, sin embargo, que Power BI Desktop solo está disponible para Windows. Power BI quiere dar sentido a los grandes conjuntos de datos visualizándolos en paneles interactivos. En esencia, se basa en la misma funcionalidad de Power Query y Power Pivot que Excel. Los planes comerciales le permiten colaborar y compartir paneles en línea, pero estos están separados de la versión de escritorio. La razón principal por la que Power BI es interesante en el contexto de este libro es que ha estado respaldando scripts de Python desde 2018. Python se puede usar para la parte de consulta, así como para la parte de visualización, haciendo uso de las bibliotecas de trazado de Python. Para mí, usar Python en Power BI se siente un poco torpe, pero la parte importante aquí es que Microsoft ha reconocido la importancia de Python con respecto al análisis de datos. Respectivamente,

Entonces, ¿qué tiene de bueno Python que se convirtió en Power BI de Microsoft? ¡La siguiente sección tiene algunas respuestas!

## **Python para Excel**

Excel se trata de almacenar, analizar y visualizar datos. Y dado que Python es particularmente sólido en el área de la computación científica, es un ajuste natural en combinación con Excel. Python es también uno de los pocos lenguajes que atrae tanto al programador profesional como al usuario principiante que escribe unas pocas líneas de código cada pocas semanas. A los programadores profesionales, por un lado, les gusta trabajar con Python porque es un lenguaje de programación de propósito general y, por lo tanto, le permite lograr prácticamente cualquier cosa sin saltar por los obstáculos. A los principiantes, por otro lado, les gusta Python porque es más fácil de aprender que otros idiomas. Como consecuencia, Python se utiliza tanto para análisis de datos ad hoc y tareas de automatización más pequeñas, así como en enormes bases de código de producción como el backend de Instagram.<sup>6</sup> Esto también significa que cuando su herramienta de Excel impulsada por Python se vuelve realmente popular, es fácil agregar un desarrollador web al proyecto que convertirá su prototipo de Excel-Python en una aplicación web completa. La ventaja única de Python es que la parte con la

---

<sup>6</sup> Puede obtener más información sobre cómo Instagram usa Python en sus [blog de ingeniería](#).

Lo más probable es que no sea necesario reescribir la lógica empresarial, pero se puede mover tal cual desde el prototipo de Excel al entorno web de producción.

En esta sección, presentaré los conceptos centrales de Python y los compararé con Excel y VBA. Me referiré a la legibilidad del código, la biblioteca estándar y el administrador de paquetes de Python, la pila de computación científica, las características del lenguaje moderno y la compatibilidad entre plataformas. ¡Vamos a sumergirnos primero en la legibilidad!

## Legibilidad y mantenibilidad

Si su código es legible, significa que es fácil de seguir y comprender, especialmente para personas externas que no han escrito el código por sí mismos. Esto facilita la detección de errores y el mantenimiento del código en el futuro. Es por eso que una línea en *El Zen de Python* es "la legibilidad cuenta". El Zen de Python es un resumen conciso de los principios básicos de diseño de Python, y aprenderemos cómo imprimirla en el próximo capítulo. Echemos un vistazo al siguiente fragmento de código en VBA:

```
Si I < 5 Luego
    Depurar.Impresión "i es menor que 5" De lo
    contrario I <= 10 Luego
        Depurar.Impresión "i está entre 5 y 10"
    Demás
        Depurar.Impresión "yo es mayor que 10"
Terminara si
```

En VBA, puede reformatear el fragmento de la siguiente manera, que es completamente equivalente:

```
Si I < 5 Luego
    Depurar.Impresión "i es menor que 5" De lo
    contrario I <= 10 Luego
        Depurar.Impresión "i está entre 5 y 10"
    Demás
        Depurar.Impresión "yo es mayor que 10"
Terminara si
```

En la primera versión, la sangría visual se alinea con la lógica del código. Esto facilita la lectura y la comprensión del código, lo que nuevamente facilita la detección de errores. En la segunda versión, es posible que un desarrollador que sea nuevo en el código no vea el De lo contrario y Demás condición cuando se mira por primera vez; obviamente, esto es aún más cierto si el código es parte de una base de código más grande.

Python no acepta código formateado como el segundo ejemplo: te obliga a alinear la sangría visual con la lógica del código, evitando problemas de legibilidad. Python puede hacer esto porque se basa en la sangría para definir bloques de código a medida que los usa en declaraciones o por bucles. En lugar de sangría, la mayoría de los otros lenguajes usan llaves y VBA usa palabras clave como `Terminara si`, como acabamos de ver en los fragmentos de código. La razón detrás del uso de sangría para bloques de código es que en

programación, la mayor parte del tiempo se dedica a mantener el código en lugar de escribirlo en primer lugar. Tener un código legible ayuda a los nuevos programadores (o a usted mismo unos meses después de escribir el código) a volver atrás y comprender lo que está sucediendo.

Aprenderemos todo sobre las reglas de sangría de Python en [Capítulo 3](#), pero por ahora sigamos con la biblioteca estándar: la funcionalidad que viene con Python lista para usar.

### Administrador de paquetes y biblioteca estándar

Python viene con un rico conjunto de funcionalidades integradas entregadas por su *biblioteca estándar*. A la comunidad de Python le gusta referirse a él diciendo que Python viene con "baterías incluidas". Ya sea que necesite descomprimir un archivo ZIP, leer los valores de un archivo CSV o desee obtener datos de Internet, la biblioteca estándar de Python lo tiene cubierto, y puede lograr todo esto en solo unas pocas líneas de código. La misma funcionalidad en VBA requeriría que escribiera una cantidad considerable de código o instale un complemento. Y, a menudo, las soluciones que encuentra en Internet solo funcionan en Windows, pero no en macOS.

Si bien la biblioteca estándar de Python cubre una cantidad impresionante de funcionalidad, todavía hay tareas que son engorrosas de programar o lentas cuando solo confías en la biblioteca estándar. Aquí es donde PyPI entra. PyPI significa *Índice de paquetes de Python* y es un repositorio gigante donde todos (¡incluido usted!) pueden cargar paquetes Python de código abierto que agregan funcionalidad adicional a Python.



PyPI frente a PyPy

PyPI se pronuncia "ojo de guisante de tarta". Esto es para diferenciar PyPI de PyPy, que se pronuncia "pastel" y que es una implementación alternativa rápida de Python.

Por ejemplo, para facilitar la obtención de datos de fuentes en Internet, puede instalar el paquete Requests para obtener acceso a un conjunto de comandos que son potentes pero fáciles de usar. Para instalarlo, usaría el administrador de paquetes de Python *pepita*, que se ejecuta en un símbolo del sistema o terminal. pip es un acrónimo recursivo de *pip instala paquetes*. No se preocupe si esto suena un poco abstracto en este momento; Explicaré cómo funciona esto en detalle en el próximo capítulo. Por ahora, es más importante comprender por qué los administradores de paquetes son tan importantes. Una de las principales razones es que cualquier paquete razonable no solo dependerá de la biblioteca estándar de Python, sino también de otros paquetes de código abierto que también están alojados en PyPI. Estas dependencias pueden volver a depender de subdependencias, etc. pip comprueba de forma recursiva las dependencias y subdependencias de un paquete y las descarga e instala. pip también facilita la actualización de sus paquetes para que pueda mantener sus dependencias actualizadas. Esto hace que adherirse al principio DRY sea mucho más fácil, ya que no es necesario reinventar o copiar / pegar lo que ya está disponible en PyPI. Con pip y PyPI, también tiene un sólido

mecanismo para distribuir e instalar estas dependencias, algo que le falta a Excel con sus complementos tradicionales.

### Software de código abierto (OSS)

En este punto, me gustaría decir algunas palabras sobre *fuente abierta*, ya que he usado esa palabra varias veces en esta sección. Si el software se distribuye bajo una licencia de código abierto, significa que su código fuente está disponible gratuitamente sin costo, lo que permite a todos contribuir con nuevas funcionalidades, corrección de errores o documentación. Python en sí y casi todos los paquetes de Python de terceros son de código abierto y los desarrolladores suelen mantenerlos en su tiempo libre. Este no es siempre un estado ideal: si su empresa confía en ciertos paquetes, tiene interés en el desarrollo y mantenimiento continuo de estos paquetes por parte de programadores profesionales. Afortunadamente, la comunidad científica de Python ha reconocido que algunos paquetes son demasiado importantes para dejar su destino en manos de unos pocos voluntarios que trabajan por las tardes y los fines de semana.

Por eso en 2012, [NumFOCUS](#), una organización sin fines de lucro, fue creada para patrocinar varios paquetes y proyectos de Python en el área de la computación científica. Los paquetes de Python más populares patrocinados por NumFOCUS son pandas, NumPy, SciPy, Matplotlib y Project Jupyter, pero hoy en día también admiten paquetes de varios otros lenguajes, incluidos R, Julia y JavaScript. Hay algunos patrocinadores corporativos importantes, pero todos pueden unirse a NumFOCUS como miembros gratuitos de la comunidad; las donaciones son deducibles de impuestos.

Con pip, puede instalar paquetes para casi cualquier cosa, pero para los usuarios de Excel, algunos de los más interesantes son sin duda los paquetes para informática científica.

¡Aprendamos un poco más sobre computación científica con Python en la siguiente sección!

## Computación científica

Una razón importante del éxito de Python es el hecho de que fue creado como un lenguaje de programación de propósito general. Las capacidades para la computación científica se agregaron más tarde en forma de paquetes de terceros. Esto tiene la ventaja única de que un científico de datos puede usar el mismo lenguaje para experimentos e investigación que un desarrollador web, quien eventualmente puede construir una aplicación lista para producción alrededor del núcleo computacional. Ser capaz de crear aplicaciones científicas a partir de un idioma reduce la fricción, el tiempo de implementación y los costos. Los paquetes científicos como NumPy, SciPy y pandas nos dan acceso a una forma muy concisa de formular problemas matemáticos. Como ejemplo, echemos un vistazo a una de las fórmulas financieras más famosas que se utilizan para calcular la variación de la cartera de acuerdo con la teoría moderna de la cartera:

$$\sigma^2 = w^T C w$$

La varianza de la cartera se denota por  $\sigma^2$ , tiempo  $w$  es el vector de ponderación de los activos individuales y  $C$  es la matriz de covarianza de la cartera. Si  $w$  y  $C$  son rangos de Excel, puede calcular la variación de la cartera en VBA así:

```
diferencia = Solicitud.MMult(Solicitud.MMult(Solicitud.Transponer(w), C), w)
```

Compare esto con la notación casi matemática en Python, asumiendo que  $w$  y  $C$  son pandas DataFrames o matrices NumPy (los presentaré formalmente en [Parte II](#)):

```
diferencia = w.T @ C @ w
```

Pero no se trata solo de estética y legibilidad: NumPy y pandas usan código Fortran y C compilado debajo del capó, lo que le brinda un aumento de rendimiento cuando trabaja con matrices grandes en comparación con VBA.

La falta de soporte para la computación científica es una limitación obvia en VBA. Pero incluso si observa las características principales del lenguaje, VBA se ha quedado atrás, como señalaré en la siguiente sección.

### Características del lenguaje moderno

Desde Excel 97, el lenguaje VBA no ha tenido cambios importantes en términos de características del lenguaje. Sin embargo, eso no significa que VBA ya no sea compatible: Microsoft está enviando actualizaciones con cada nueva versión de Excel para poder automatizar las nuevas funciones de Excel introducidas con esa versión. Por ejemplo, Excel 2016 agregó soporte para automatizar Power Query. Un lenguaje que dejó de evolucionar hace más de veinte años se está perdiendo los conceptos del lenguaje moderno que se introdujeron en los principales lenguajes de programación a lo largo de los años. Como ejemplo, el manejo de errores en VBA realmente está mostrando su edad. Si desea manejar un error con gracia en VBA, es algo como esto:

```
Sub Print Recíproco(número como variante)
    'Habrá un error si el número es 0 o una cadenaEn caso de
    error Ir a ErrorHandler
    resultado = 1 / número en
    caso de error Ir a 0
    Depurar.Impresión ";No hubo ningún error!"
    Finalmente:
    'Se ejecuta tanto si se produce un error como si no
    Si resultado = "" Luego
        resultado = "N / A"
    Terminara si
    Depurar.Impresión "El recíproco es:" Y resultado
    Salir Sub
ErrorHandler:
    'Se ejecuta solo en caso de error
    Depurar.Impresión "Hubo un error:" Y Errar.Descripción
    Reanudar finalmente
End Sub
```

El manejo de errores de VBA implica el uso de *etiquetas* igual que Finalmente y ErrorHandler en el ejemplo. Le indica al código que salte a estas etiquetas a través del Ir a o Reanudar declaraciones. Al principio, se reconoció que las etiquetas eran responsables de lo que muchos programadores llamarían *código de espagueti*: una buena forma de decir que el flujo del código es difícil de seguir y, por lo tanto, difícil de mantener. Es por eso que casi todos los lenguajes desarrollados activamente han introducido la traza de atraparlo mecanismo: en Python llamado probar / excepto—que voy a introducir en [Capítulo 11](#). Si es un desarrollador experto en VBA, también puede disfrutar del hecho de que Python admite la herencia de clases, una característica de la programación orientada a objetos que falta en VBA.

Además de las características del lenguaje moderno, existe otro requisito para un lenguaje de programación moderno: compatibilidad multiplataforma. ¡Veamos por qué esto es importante!

## Compatibilidad multiplataforma

Incluso si desarrolla su código en una computadora local que se ejecuta en Windows o macOS, es muy probable que desee ejecutar su programa en un servidor o en la nube en algún momento. Los servidores permiten que su código se ejecute en un horario y hacen que su aplicación sea accesible desde cualquier lugar que desee, con la potencia informática que necesita. De hecho, en el próximo capítulo le mostraré cómo ejecutar código Python en un servidor al presentarle los cuadernos Jupyter alojados. La gran mayoría de los servidores se ejecutan en Linux, ya que es un sistema operativo estable, seguro y rentable. Y dado que los programas de Python se ejecutan sin cambios en todos los sistemas operativos principales, esto eliminará gran parte del dolor cuando realice la transición de su máquina local a una configuración de producción.

Por el contrario, aunque Excel VBA se ejecuta en Windows y macOS, es fácil introducir funciones que solo se ejecutan en Windows. En la documentación oficial de VBA o en los foros, a menudo verá un código como este:

```
Colocar fso = Crear objeto("Scripting.FileSystemObject")
```

Siempre que tengas un Crear objeto llamar o se le dice que vaya a Herramientas> Referencias en el editor de VBA para agregar una referencia, casi siempre se trata de un código que solo se ejecutará en Windows. Otra área destacada en la que debe tener cuidado si desea que sus archivos de Excel funcionen en Windows y macOS son *Controles ActiveX*. Los controles ActiveX son elementos como botones y menús desplegables que puede colocar en sus hojas, pero funcionan solo en Windows. ¡Asegúrese de evitarlos si desea que su libro de trabajo también se ejecute en macOS!

## Conclusión

En este capítulo, conocimos a Python y Excel, dos tecnologías muy populares que han existido durante varias décadas, mucho tiempo en comparación con muchas otras tecnologías que usamos hoy. London Whale sirvió como ejemplo de cuánto puede salir mal (en términos de dólares) cuando no se usa Excel correctamente con misiones críticas

libros de trabajo. Esta fue nuestra motivación para analizar un conjunto mínimo de mejores prácticas de programación: aplicar la separación de preocupaciones, seguir el principio DRY y hacer uso de pruebas automatizadas y control de versiones. Luego echamos un vistazo a Power Query y Power Pivot, el enfoque de Microsoft para manejar datos que son más grandes que su hoja de cálculo. Sin embargo, creo que a menudo no son la solución adecuada, ya que lo encierran en el mundo de Microsoft y le impiden aprovechar la flexibilidad y el poder de las soluciones modernas basadas en la nube.

Python viene con características convincentes que faltan en Excel: la biblioteca estándar, el administrador de paquetes, bibliotecas para computación científica y compatibilidad multiplataforma. Al aprender a combinar Excel con Python, puede tener lo mejor de ambos mundos y ahorrar tiempo a través de la automatización, cometer menos errores ya que es más fácil seguir las mejores prácticas de programación y podrá tomar su aplicación y escalarla fuera de Excel si alguna vez lo necesita.

Ahora que sabe por qué Python es un compañero tan poderoso para Excel, es hora de configurar su entorno de desarrollo para poder escribir sus primeras líneas de código Python.

# Entorno de desarrollo

Probablemente no pueda esperar para aprender los conceptos básicos de Python, pero antes de llegar allí, primero debe configurar su computadora en consecuencia. Para escribir código VBA o Power Queries, basta con iniciar Excel y abrir el editor VBA o Power Query, respectivamente. Con Python, es un poco más de trabajo.

Comenzaremos este capítulo instalando la distribución Anaconda Python. Además de instalar Python, Anaconda también nos dará acceso a los cuadernos Anaconda Prompt y Jupyter, dos herramientas esenciales que usaremos a lo largo de este libro. *losAviso Anaconda* es un símbolo del sistema (Windows) o terminal (macOS) especial; nos permite ejecutar scripts de Python y otras herramientas de línea de comandos que encontraremos en este libro. *CuadernosJupyter* nos permiten trabajar con datos, código y gráficos de forma interactiva, lo que los convierte en un serio competidor de los libros de Excel. Después de jugar con los cuadernos de Jupyter, instalaremos *Código de Visual Studio* (VS Code), un potente editor de texto. VS Code funciona muy bien para escribir, ejecutar y depurar scripts de Python y viene con una Terminal integrada. [Figura 2-1](#) resume lo que se incluye en Anaconda y VS Code.

Como este libro trata sobre Excel, me centraré en Windows y macOS en este capítulo. Sin embargo, todo hasta e incluyendo [Parte III](#) también se ejecuta en Linux. ¡Comencemos instalando Anaconda!

<b>Anaconda</b> A scientific Python distribution	<b>VS Code</b> A powerful text editor
Python	Python extension
Anaconda Prompt	Debugger
Jupyter notebook, pandas, xlwings, etc.	Terminal
Conda & pip	Git

Figura 2-1. Entorno de desarrollo

## La distribución de Anaconda Python

Anaconda es posiblemente la distribución de Python más popular utilizada para la ciencia de datos y viene con cientos de paquetes de terceros preinstalados: incluye cuadernos de notas de Jupyter y la mayoría de los otros paquetes que este libro usará ampliamente, incluidos pandas, OpenPyXL y xlwings. La Anaconda Individual Edition es gratuita para uso privado y garantiza que todos los paquetes incluidos son compatibles entre sí. Se instala en una sola carpeta y se puede desinstalar fácilmente de nuevo. Después de instalarlo, aprenderemos algunos comandos básicos en Anaconda Prompt y ejecutaremos una sesión interactiva de Python. Luego nos reuniremos con los administradores de paquetes Conda y pip antes de envolver esta sección con entornos Conda. ¡Comencemos descargando e instalando Anaconda!

## Instalación

Ve a la [Página de inicio de Anaconda](#) y descargue la última versión del instalador de Anaconda (Edición Individual). Asegúrese de descargar el instalador gráfico de 64 bits para la versión Python 3.x.<sup>1</sup> Una vez descargado, haga doble clic en el instalador para iniciar el proceso de instalación y asegúrese de aceptar todos los valores predeterminados. Para obtener instrucciones de instalación más detalladas, siga las[documentación oficial](#).

---

<sup>1</sup> Los sistemas de 32 bits solo existen con Windows y se han vuelto raros. Una forma sencilla de averiguar qué versión de Windows tiene es yendo a la C:\ drive en el Explorador de archivos. Si puedes ver tanto el Archivos de programa y Archivos de programa (x86) carpetas, está en una versión de Windows de 64 bits. Si solo puedes ver el Archivos de programas carpeta, está en un sistema de 32 bits.



#### Otras distribuciones de Python

Si bien las instrucciones de este libro asumen que tiene instalada Anaconda Individual Edition, el código y los conceptos mostrados también funcionarán con cualquier otra instalación de Python. En este caso, sin embargo, tendrá que instalar las dependencias necesarias siguiendo las instrucciones incluidas en *requirements.txt* en el repositorio complementario.

Con Anaconda instalado, ahora podemos comenzar a usar Anaconda Prompt. ¡Veamos qué es esto y cómo funciona!

## Aviso Anaconda

los *Aviso Anaconda* es solo un símbolo del sistema en Windows y una terminal en macOS que se han configurado para ejecutarse con el intérprete de Python correcto y los paquetes de terceros. Anaconda Prompt es la herramienta más básica para ejecutar código Python, y haremos un uso extensivo de ella en este libro para ejecutar scripts Python y todo tipo de herramientas de línea de comandos que se ofrecen en varios paquetes.



#### Anaconda Prompt sin Anaconda

Si no usa la distribución Anaconda Python, tendrá que usar la *Símbolo del sistema* en Windows y el *Terminal* en macOS siempre que le indique que use Anaconda Prompt.

Si nunca ha utilizado un símbolo del sistema en Windows o una terminal en macOS, no se preocupe: solo necesita conocer un puñado de comandos que ya le darán mucho poder. Una vez que se acostumbre, usar Anaconda Prompt suele ser más rápido y conveniente que hacer clic en los menús gráficos de usuario. Empecemos:

### Ventanas

Haga clic en el botón del menú Inicio y comience a escribir **Aviso Anaconda**. En las entradas que aparecen, elija Anaconda Prompt, no Anaconda Powershell Prompt. Selecciónelo con las teclas de flecha y presione Enter o use su mouse para hacer clic en él. Si prefiere abrirlo a través del menú Inicio, lo encontrará en Anaconda3. Es una buena idea anclar Anaconda Prompt en la barra de tareas de Windows, ya que lo usará con regularidad a lo largo de este libro. La línea de entrada de Anaconda Prompt comenzará con (base):

```
(base) C:\ Usuarios\felix>
```

### Mac OS

En macOS, no encontrará una aplicación llamada Anaconda Prompt. En cambio, por Anaconda Prompt, me refiero a la Terminal que ha sido configurada por el instalador de Anaconda para activar automáticamente un entorno Conda (diré más sobre los entornos Conda en un momento): presione la barra espaciadora de Comando o abra el Launchpad, luego escribe **Terminal** y presione Enter. Alternativamente, abra el Finder y navegue hasta *Aplicaciones > Utilidades*, donde encontrará la aplicación Terminal en la que puede hacer doble clic. Una vez que aparece la Terminal, debería verse algo así, es decir, la línea de entrada debe comenzar con (base):

```
(base) felix @ MacBook-Pro ~%
```

Si tiene una versión anterior de macOS, se parece a esto:

```
(base) MacBook-Pro: ~ felix $
```

A diferencia del símbolo del sistema en Windows, la Terminal en macOS no muestra la ruta completa del directorio actual. En cambio, la tilde representa el directorio de inicio, que suele ser */Usuarios / <nombre de usuario>*. Para ver la ruta completa de su directorio actual, escriba **pwd** seguido de Enter. **pwd** representa *imprimir directorio de trabajo*.

Si la línea de entrada en su Terminal no comienza con (base) después de la instalación de Anaconda, aquí hay una razón común: si tenía la Terminal ejecutándose durante la instalación de Anaconda, deberá reiniciarla. Tenga en cuenta que al hacer clic en la cruz roja en la parte superior izquierda de la ventana de Terminal solo se ocultará, pero no se cerrará. En su lugar, haga clic derecho en la Terminal en el dock y seleccione Salir o presione Comando-Q mientras la Terminal es su ventana activa. Cuando lo inicie de nuevo y la Terminal muestre (base) al comienzo de una nueva línea, ya está todo listo. Es una buena idea mantener la Terminal en su base, ya que la usará regularmente a lo largo de este libro.

Con Anaconda Prompt en funcionamiento, pruebe los comandos descritos en [Tabla 2-1](#). Estoy explicando cada comando con más detalle después de la tabla.

*Tabla 2-1. Comandos para el indicador Anaconda*

Mando	Ventanas	Mac OS
Lista de archivos en el directorio actual	dir	ls -la
Cambiar directorio (relativo)	ruta de cd \ a \ dir	ruta de cd / a / dir
Cambiar directorio (absoluto)	cd C: \ ruta \ a \ dir	cd / ruta / a / dir
Cambiar a unidad D	D:	(no existe)
Cambiar al directorio principal	CD ..	CD ..
Desplazarse por los comandos anteriores	↑ (flecha hacia arriba)	↑ (flecha hacia arriba)

#### *Lista de archivos en el directorio actual*

En Windows, escriba **dir** por *directorio*, luego presione Enter. Esto imprimirá el contenido del directorio en el que se encuentra actualmente.

En macOS, escriba **ls -la** seguido de Enter. ls es la abreviatura de *enumerar el contenido del directorio*, y -la imprimirá la salida en el *lista larga* formatear e incluir *todos* archivos, incluidos los ocultos.

#### *Cambio de directorio*

Escribe **cd abajo** y presione la tecla Tab. CD representa *cambio de directorio*. Si está en su carpeta de inicio, lo más probable es que Anaconda Prompt pueda autocompletarlo para Descargas de cd. Si está en una carpeta diferente o no tiene una carpeta llamada *Descargas*, simplemente comience a escribir el comienzo de uno de los nombres de directorio que vio con el comando anterior (dir o ls -la) antes de presionar la tecla Tab para autocompletar. Luego presione Enter para cambiar al directorio autocompletado. Si está en Windows y necesita cambiar su unidad, primero debe escribir el nombre de la unidad antes de poder cambiar al directorio correcto:

```
C:\ Usuarios \ felix> D:  
D:\> datos de cd  
D:\ datos>
```

Tenga en cuenta que al comenzar su ruta con un directorio o nombre de archivo que se encuentra dentro de su directorio actual, está utilizando un *camino relativo*, p.ej, Descargas de cd. Si desea salir de su directorio actual, puede escribir un *camino absoluto*, p.ej, cd C:\ Usuarios en Windows o cd / Usuarios en macOS (tenga en cuenta la barra inclinada al principio).

#### *Cambiar al directorio principal*

Para ir a su directorio principal, es decir, un nivel más arriba en su jerarquía de directorios, escriba **CD ..** seguido de Enter (asegúrese de que haya un espacio entre CD y los puntos). Puede combinar esto con un nombre de directorio, por ejemplo, si desea subir un nivel y luego cambiar al *Escritorio*, ingresar **cd .. \ Escritorio**. En macOS, reemplace la barra invertida por una barra inclinada hacia adelante.

#### *Desplazarse por los comandos anteriores*

Utilice la tecla de flecha hacia arriba para desplazarse por los comandos anteriores. Esto le ahorrará muchas pulsaciones de teclas si necesita ejecutar los mismos comandos una y otra vez. Si se desplaza demasiado lejos, use la tecla de flecha hacia abajo para desplazarse hacia atrás.

#### **Extensiones de archivo**

Desafortunadamente, Windows y macOS ocultan las extensiones de archivo de forma predeterminada en el Explorador de Windows o en el Finder de macOS, respectivamente. Esto puede dificultar el trabajo con scripts de Python y Anaconda Prompt, ya que requerirán que consulte archivos que incluyen

sus extensiones. Cuando trabaja con Excel, mostrar las extensiones de archivo también le ayuda a comprender si está tratando con un archivo .xlsx, una macro habilitada .xlsm o cualquier otro formato de archivo. Así es como puede hacer visibles las extensiones de archivo:

#### Ventanas

Abra un Explorador de archivos y haga clic en la pestaña Ver. En el grupo Mostrar / Ocultar, active la casilla de verificación "Extensiones de nombre de archivo".

#### Mac OS

Abra el Finder y vaya a Preferencias presionando Comando-, (Comandocomma). En la pestaña Avanzado, marque la casilla junto a "Mostrar todas las extensiones de nombre de archivo".

¡Y eso ya es todo! Ahora puede iniciar Anaconda Prompt y ejecutar comandos en el directorio deseado. Lo usaré de inmediato en la siguiente sección, donde le mostraré cómo iniciar una sesión interactiva de Python.

## Python REPL: una sesión interactiva de Python

Puede iniciar una sesión interactiva de Python ejecutando el comando `pitón` en un indicador de Anaconda:

```
(base) C:\ Usuarios\ felix>pitón
Python 3.8.5 (predeterminado, 3 de septiembre de 2020, 21:29:08) [...] :: Anaconda, Inc. en win32 Escriba
"ayuda", "derechos de autor", "créditos" o "licencia" para obtener más información.
>>>
```

El texto que se imprime en una Terminal en macOS será ligeramente diferente, pero por lo demás, funciona igual. Este libro se basa en Python 3.8. Si desea utilizar una versión más reciente de Python, asegúrese de consultar el [página de inicio del libro](#) para obtener instrucciones.



#### Notación rápida de Anaconda

En el futuro, comenzaré líneas de código con `(base)>` para indicar que se escriben en un mensaje de Anaconda. Por ejemplo, para lanzar un intérprete interactivo de Python, escribiré:

```
(base)> pitón
```

que en Windows se verá similar a esto:

```
(base) C:\ Usuarios\ felix> pitón
```

y en macOS similar a esto (recuerde, en macOS, la Terminal es su Anaconda Prompt):

```
(base) felix @ MacBook-Pro ~%pitón
```

¡Juguemos un poco! Tenga en cuenta que >>> en una sesión interactiva significa que Python espera su entrada; no tiene que escribir esto. Siga escribiendo en cada línea que comience con >>> y confirme con la tecla Enter:

```
>>> 3 + 4  
7  
>>> "pitón" * 3'pitón  
pitón pitón'
```

Esta sesión interactiva de Python también se conoce como Python *REPL*, Lo que significa *bucle de lectura-evaluación-impresión*: Python lee su entrada, la evalúa e imprime el resultado instantáneamente mientras espera su próxima entrada. ¿Recuerda el Zen de Python que mencioné en el capítulo anterior? Ahora puede leer la versión completa para obtener una idea de los principios rectores de Python (sonrisa incluida). Simplemente ejecute esta línea presionando Enter después de escribirla:

```
>>> importar esto
```

Para salir de su sesión de Python, escriba **dejar()** seguido de la tecla Enter. Alternativamente, presione Ctrl + Z en Windows, luego presione la tecla Enter. En macOS, simplemente presione Ctrl-D no es necesario presionar Enter.

Habiendo salido de Python REPL, es un buen momento para jugar con Conda y pip, los administradores de paquetes que vienen con la instalación de Anaconda.

## Gestores de paquetes: Conda y pip

Ya dije algunas palabras sobre pip, el administrador de paquetes de Python en el capítulo anterior: pip se encarga de descargar, instalar, actualizar y desinstalar los paquetes de Python, así como sus dependencias y subdependencias. Si bien Anaconda funciona con pip, tiene un administrador de paquetes alternativo incorporado llamado Conda. Una ventaja de Conda es que puede instalar más que solo paquetes de Python, incluidas versiones adicionales del intérprete de Python. Como breve recapitulación: los paquetes agregan funcionalidad adicional a su instalación de Python que no está cubierta por la biblioteca estándar. pandas, que introduciré correctamente en Capítulo 5, es un ejemplo de un paquete de este tipo. Como viene preinstalado en la instalación de Python de Anaconda, no es necesario que lo instale manualmente.



Conda contra pip

Con Anaconda, debe instalar todo lo que pueda a través de Conda y solo usar pip para instalar aquellos paquetes que Conda no puede encontrar. De lo contrario, Conda puede sobrescribir archivos que se instalaron previamente con pip.

**Tabla 2-2** le ofrece una descripción general de los comandos que utilizará con más frecuencia. Estos comandos deben escribirse en un indicador de Anaconda y le permitirán instalar, actualizar y desinstalar sus paquetes de terceros.

*Tabla 2-2. Comandos conda y pip*

Acción	Conda	pepita
Lista de todos los paquetes instalados	lista de conda	congelación de pepitas
Instalar la última versión del paquete	instalación de conda <i>paquete</i>	instalación de pip <i>paquete</i>
Instalar una versión de paquete específica	instalación de conda <i>paquete = 1.0.0</i>	instalación de pip <i>paquete == 1.0.0</i>
Actualizar un paquete	actualización de conda <i>paquete</i>	pip install --upgrade <i>paquete</i>
Desinstalar un paquete	conda quitar <i>paquete</i>	desinstalar pip <i>paquete</i>

Por ejemplo, para ver qué paquetes ya están disponibles en su distribución de Anaconda, escriba:

(base)> **lista de conda**

Siempre que este libro requiera un paquete que no esté incluido en la instalación de Anaconda, lo señalaré explícitamente y le mostraré cómo instalarlo. Sin embargo, puede ser una buena idea encargarse de instalar los paquetes que faltan ahora para que no tenga que ocuparse de ellos más adelante. Primero instalemos plotly y xlutils, los paquetes que están disponibles a través de Conda:

(base)> **conda instalar plotly xlutils**

Después de ejecutar este comando, Conda le mostrará lo que va a hacer y le pedirá que confirme escribiendo **y** y presionando Enter. Una vez hecho esto, puede instalar pyxlsb y pytrends con pip, ya que estos paquetes no están disponibles a través de Conda:

(base)> **pip instalar pyxlsb pytrends**

A diferencia de Conda, pip instalará los paquetes de inmediato cuando presione Enter sin necesidad de confirmar.



#### Versiones del paquete

Muchos paquetes de Python se actualizan con frecuencia y, a veces, introducen cambios que no son compatibles con versiones anteriores. Esto probablemente romperá algunos de los ejemplos de este libro. Intentaré mantenerme al día con estos cambios y publicar correcciones en el [página de inicio del libro](#), pero también puede crear un entorno Conda que use las mismas versiones de los paquetes que yo estaba usando al escribir este libro. Presentaré los entornos Conda en la siguiente sección, y encontrará instrucciones detalladas sobre cómo crear un entorno Conda con los paquetes específicos en [Apéndice A](#).

Ahora sabe cómo usar Anaconda Prompt para iniciar un intérprete de Python e instalar paquetes adicionales. En la siguiente sección, explicaré qué (base) al comienzo de sus medios Anaconda Prompt.

## Ambientes Conda

Es posible que se haya preguntado por qué Anaconda Prompt muestra (base) al comienzo de cada línea de entrada. Es el nombre del activo *Entorno Conda*. Un entorno Conda es un "mundo Python" separado con una versión específica de Python y un conjunto de paquetes instalados con versiones específicas. ¿Por qué es esto necesario? Cuando comience a trabajar en diferentes proyectos en paralelo, tendrán diferentes requisitos: un proyecto puede usar Python 3.8 con pandas 0.25.0, mientras que otro proyecto puede usar Python 3.9 con pandas 1.0.0. El código escrito para pandas 0.25.0 a menudo requerirá cambios para ejecutarse con pandas 1.0.0, por lo que no puede simplemente actualizar sus versiones de Python y pandas sin realizar cambios en su código. El uso de un entorno Conda para cada proyecto asegura que cada proyecto se ejecute con las dependencias correctas. Si bien los entornos Conda son específicos de la distribución de Anaconda, el concepto existe con cada instalación de Python bajo el nombre *ambiente virtual*. Los entornos Conda son más poderosos porque facilitan el manejo de diferentes versiones de Python en sí, no solo paquetes.

Mientras trabaja en este libro, no tendrá que cambiar su entorno Conda, ya que siempre usaremos el base medio ambiente. Sin embargo, cuando comienza a construir proyectos reales, es una buena práctica utilizar un entorno virtual o Conda para cada proyecto para evitar posibles conflictos entre sus dependencias. Todo lo que necesita saber sobre cómo lidiar con múltiples entornos de Conda se explica en [Apéndice A](#). Allí también encontrará instrucciones sobre cómo crear un entorno Conda con las versiones exactas de los paquetes que utilicé para escribir este libro. Esto le permitirá ejecutar los ejemplos de este libro tal como están durante muchos años. La otra opción es mirar el [página de inicio del libro](#) para conocer los posibles cambios necesarios para las versiones más recientes de Python y los paquetes.

Habiendo resuelto el misterio en torno a los entornos Conda, es hora de presentar la siguiente herramienta, una que usaremos intensamente en este libro: ¡los cuadernos de Jupyter!

## Cuadernos Jupyter

En la sección anterior, le mostré cómo iniciar una sesión interactiva de Python desde un indicador de Anaconda. Esto es útil si desea un entorno básico para probar algo simple. Sin embargo, para la mayor parte de su trabajo, desea un entorno que sea más fácil de usar. Por ejemplo, volver a los comandos anteriores y mostrar gráficos es difícil con un REPL de Python ejecutándose en un indicador de Anaconda. Afortunadamente, Anaconda viene con mucho más que el intérprete de Python: también incluye *Cuadernos Jupyter*, que se han convertido en una de las formas más populares de ejecutar

Código Python en un contexto de ciencia de datos. Los cuadernos de Jupyter le permiten contar una historia combinando código Python ejecutable con texto formateado, imágenes y gráficos en un cuaderno interactivo que se ejecuta en su navegador. Son amigables para principiantes y, por lo tanto, especialmente útiles para los primeros pasos de su viaje en Python. Sin embargo, también son muy populares para la enseñanza, la creación de prototipos y la investigación, ya que facilitan una investigación reproducible.

Los cuadernos de Jupyter se han convertido en un serio competidor de Excel, ya que cubren aproximadamente el mismo caso de uso que un libro de trabajo: puede preparar, analizar y visualizar datos rápidamente. La diferencia con Excel es que todo sucede al escribir código Python en lugar de hacer clic en Excel con el mouse. Otra ventaja es que los cuadernos de Jupyter no mezclan datos y lógica empresarial: el cuaderno de Jupyter contiene su código y gráficos, mientras que normalmente consume datos de un archivo CSV externo o una base de datos. Tener el código Python visible en su cuaderno facilita ver lo que está sucediendo en comparación con Excel, donde las fórmulas están ocultas detrás del valor de una celda. Los cuadernos de Jupyter también son fáciles de ejecutar tanto localmente como en un servidor remoto. Los servidores suelen tener más potencia que su máquina local y pueden ejecutar su código sin supervisión,

En esta sección, le mostraré los conceptos básicos de cómo ejecutar y navegar en un cuaderno de Jupyter: aprenderemos sobre las celdas del cuaderno y veremos cuál es la diferencia entre el modo de edición y el de comando. Entonces entenderemos por qué el orden de ejecución de las celdas es importante antes de terminar esta sección aprendiendo cómo cerrar correctamente los cuadernos de notas. ¡Comencemos con nuestro primer cuaderno!

### Ejecución de cuadernos de Jupyter

En su Anaconda Prompt, cambie al directorio de su repositorio complementario, luego inicie un servidor de cuadernos Jupyter:

```
(base)> cd C:\ Usuarios \nombre de usuario\python-para-
excel(base)> cuaderno jupyter
```

Esto abrirá automáticamente su navegador y mostrará el panel de Jupyter con los archivos en el directorio desde donde estaba ejecutando el comando. En la parte superior derecha del panel de Jupyter, haga clic en Nuevo, luego seleccione Python 3 en la lista desplegable (consulte [Figura 2-2](#)).



Figura 2-2. El tablero de Jupyter

Esto abrirá una nueva pestaña del navegador con su primer cuaderno de Jupyter vacío como se muestra en Figura 2-3.

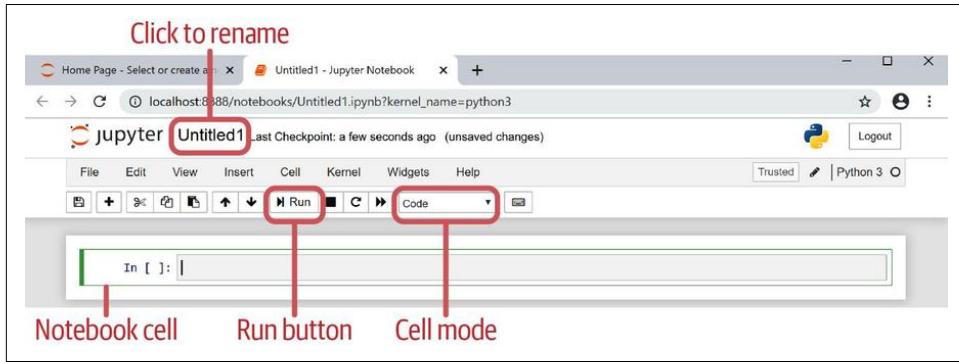


Figura 2-3. Un cuaderno de Jupyter vacío

Es un buen hábito hacer clic en Sin título1 junto al logotipo de Jupyter para cambiar el nombre de su libro de trabajo a algo más significativo, por ejemplo, *primer\_portátil*. La parte inferior de Figura 2-3 muestra una celda de cuaderno; pase a la siguiente sección para obtener más información sobre ellos.

## Celdas portátiles

En Figura 2-3, verá una celda vacía con un cursor parpadeante. Si el cursor no parpadea, haga clic en la celda con el mouse, es decir, a la derecha de En [ ]. Ahora repita el ejercicio de la última sección: escriba **3 + 4** y ejecute la celda haciendo clic en el botón Ejecutar en la barra de menú en la parte superior o, mucho más fácil, presionando Shift + Enter. Esto ejecutará el código en la celda, imprimirá el resultado debajo de la celda y saltará a la siguiente celda. En este caso, inserta una celda vacía a continuación, ya que solo tenemos una celda hasta ahora. Entrando un poco

más detalles: mientras una celda está calculando, muestra En [\*] y cuando está hecho, el asterisco se convierte en un número, por ejemplo, En 1]. Debajo de la celda, tendrá la salida correspondiente etiquetada con el mismo número: Fuera [1]. Cada vez que ejecuta una celda, el contador aumenta en uno, lo que le ayuda a ver en qué orden se ejecutaron las celdas. En el futuro, mostraré los ejemplos de código en este formato, por ejemplo, el ejemplo de REPL anterior se ve así:

En [1]: 3 + 4

Fuera [1]: 7

Esta notación le permite seguirla fácilmente escribiendo **3 + 4** en una celda de portátil. Al ejecutarlo presionando Shift + Enter, obtendrá lo que muestro como salida debajo Fuera [1]. Si lee este libro en un formato electrónico que admita colores, notará que la celda de entrada formatea cadenas, números, etc. con diferentes colores para facilitar la lectura. Se llama *resaltado de sintaxis*.



Salida de celda

Si la última línea de una celda devuelve un valor, el cuaderno de Jupyter lo imprime automáticamente en Fuera [ ]. Sin embargo, cuando usa el impresión función o cuando obtiene una excepción, se imprime directamente debajo de la En celda sin el Fuera [ ] etiqueta. Los ejemplos de código de este libro están formateados para reflejar este comportamiento.

Las células pueden tener diferentes tipos, dos de los cuales nos interesan:

#### Código

Este es el tipo predeterminado. Úselo siempre que desee ejecutar código Python.

#### Reducción

Markdown es una sintaxis que utiliza caracteres de texto estándar para formatear y se puede utilizar para incluir explicaciones e instrucciones con un formato agradable en su cuaderno.

Para cambiar el tipo de celda a Markdown, seleccione la celda, luego elija Markdown en el menú desplegable del modo de celda (consulte [Figura 2-3](#)). Te mostraré un atajo de teclado para cambiar el modo de celda en [Tabla 2-3](#). Despues de convertir una celda vacía en una celda de Markdown, escriba el siguiente texto, que explica algunas reglas de Markdown:

```
# Este es un encabezado de primer nivel
```

```
# # Este es un título de segundo nivel
```

Puede hacer que su texto sea \* cursiva \* o \*\* negrita \*\* o `monoespaciado`.

\* Este es un punto con viñetas

\* Este es otro punto de viñeta

Después de presionar Shift + Enter, el texto se renderizará en HTML con un formato agradable. En este punto, su computadora portátil debería verse como lo que está en Figura 2-4. Las celdas de Markdown también le permiten incluir imágenes, videos o fórmulas; ver el Documentos del cuaderno de Jupyter.

The screenshot shows a Jupyter Notebook interface. On the left, there is a code cell with the prompt "In [1]:" followed by the expression "3 + 4". To its right, the output "Out[1]: 7" is displayed. Below these, there are two Markdown cells. The first is a bold heading: "This is a first-level heading". The second is a bold heading: "This is a second-level heading". Underneath the second heading, there is a note: "You can make your text *italic* or **bold** or `monospaced`." Below this note is a bulleted list: "• This is a bullet point" and "• This is another bullet point". At the bottom of the screenshot, there is an empty input cell with the prompt "In [ ]:".

Figura 2-4. El portátil después de ejecutar una celda de código y una celda de Markdown

Ahora que conoce el código y los tipos de celdas de Markdown, es hora de aprender una forma más fácil de navegar entre celdas: la siguiente sección presenta el modo de edición y comando junto con algunos atajos de teclado.

#### Modo de edición frente a modo de comando

Cuando interactúa con celdas en un cuaderno de Jupyter, está en el *modo de edición* o en el *modo de comando*:

##### Modo de edición

Al hacer clic en una celda se inicia el modo de edición: el borde alrededor de la celda seleccionada se vuelve verde y el cursor en la celda parpadea. En lugar de hacer clic en una celda, también puede presionar Enter cuando la celda está seleccionada.

##### Modo de comando

Para cambiar al modo de comando, presione la tecla Escape; el borde alrededor de la celda seleccionada será azul y no habrá ningún cursor parpadeante. Los atajos de teclado más importantes que puede usar mientras está en el modo de comando se muestran en Tabla 2-3.

Tabla 2-3. Atajos de teclado (modo comando)

Atajo	Acción
Mayús + Entrar	Ejecutar la celda (también funciona en el modo de edición)
↑ (flecha hacia arriba)	Mover el selector de celda hacia arriba
↓ (flecha hacia abajo)	Mueva el selector de celdas hacia abajo
B	Insertar una nueva celda <i>debajo</i> la celda actual
a	Insertar una nueva celda <i>encima</i> la celda actual
dd	Elimina la celda actual (escribe dos veces la letra D)Cambiar
metró	el tipo de celda a Markdown Cambiar el tipo de celda a
y	código

Conocer estos atajos de teclado le permitirá trabajar con computadoras portátiles de manera eficiente sin tener que cambiar entre el teclado y el mouse todo el tiempo. En la siguiente sección, le mostraré un problema común que debe tener en cuenta al usar los cuadernos de Jupyter: la importancia de ejecutar las celdas en orden.

#### Ejecutar asuntos de orden

Como los portátiles fáciles y fáciles de usar son para empezar, también facilitan entrar en estados confusos si no ejecuta las celdas de forma secuencial. Suponga que tiene las siguientes celdas del cuaderno que se ejecutan de arriba a abajo:

En [2]: a = 1

En [3]: a

Fuera [3]: 1

En [4]: a = 2

Celda Fuera [3] imprime el valor 1 como se esperaba. Sin embargo, si ahora regresa y corre En 3] nuevamente, terminará en esta situación:

En [2]: a = 1

En [5]: a

Fuera [5]: 2

En [4]: a = 2

Fuera [5] muestra ahora el valor 2, que probablemente no sea lo que esperaría cuando lea el cuaderno desde la parte superior, especialmente si el celular En [4] estaría más lejos, lo que le obligaría a desplazarse hacia abajo. Para evitar tales casos, le recomendaría que vuelva a ejecutar no solo una celda, sino también todas sus celdas anteriores. Los cuadernos de Jupyter le ofrecen una manera fácil de lograr esto en el menú Celda> Ejecutar todo lo anterior. Después de estas palabras de advertencia, ¡veamos cómo apaga correctamente una computadora portátil!

### **Apagado de las computadoras portátiles Jupyter**

Cada portátil se ejecuta en un *Núcleo de Jupyter*. Un kernel es el "motor" que ejecuta el código Python que escribe en una celda del cuaderno. Cada kernel usa recursos de su sistema operativo en forma de CPU y RAM. Por lo tanto, cuando cierra un cuaderno, también debe apagar su kernel para que los recursos puedan ser usados nuevamente por otras tareas; esto evitará que su sistema se ralentice. La forma más sencilla de lograr esto es cerrando un cuaderno a través de Archivo> Cerrar y Detener. Si simplemente cierra la pestaña del navegador, el kernel no se cerrará automáticamente. Como alternativa, en el panel de Jupyter, puede cerrar los cuadernos en ejecución desde la pestaña En ejecución.

Para apagar todo el servidor de Jupyter, haga clic en el botón Salir en la parte superior derecha del panel de Jupyter. Si ya ha cerrado su navegador, puede escribir Ctrl + C dos veces en el indicador de Anaconda donde se está ejecutando el servidor de la computadora portátil o cerrar el indicador de Anaconda por completo.

## **Jupyter Notebooks en la nube**

Los portátiles Jupyter se han vuelto tan populares que varios proveedores de la nube los ofrecen como una solución alojada. Aquí presento tres servicios que son de uso gratuito. La ventaja de estos servicios es que se ejecutan instantáneamente y en cualquier lugar al que pueda acceder a un navegador, sin la necesidad de instalar nada localmente. Por ejemplo, podría ejecutar las muestras en una tableta mientras lee las tres primeras partes. Ya que [Parte IV](#) requiere una instalación local de Excel, sin embargo, esto no funcionará allí.

#### *Aglutinante*

[Aglutinante](#) es un servicio proporcionado por Project Jupyter, la organización detrás de los cuadernos de Jupyter. Binder está destinado a probar los cuadernos de Jupyter de los repositorios públicos de Git; no almacena nada en Binder y, por lo tanto, no necesita registrarse o iniciar sesión para usarlo.

#### *Cuadernos Kaggle*

[Kaggle](#) es una plataforma para la ciencia de datos. Como alberga competiciones de ciencia de datos, puede acceder fácilmente a una gran colección de conjuntos de datos. Kaggle forma parte de Google desde 2017.

#### *Google Colab*

[Google Colab](#) (abreviatura de Colaboratory) es la plataforma de portátiles de Google.

Desafortunadamente, la mayoría de los atajos de teclado de las computadoras portátiles Jupyter no funcionan, pero puede acceder a los archivos de su Google Drive, incluidas las Hojas de cálculo de Google.

La forma más sencilla de ejecutar los cuadernos de Jupyter del repositorio complementario en la nube es ir a su [URL de la carpeta](#). Trabajará en una copia del repositorio complementario, ¡así que síntase libre de editar y romper cosas como deseé!

Ahora que sabe cómo trabajar con los cuadernos de Jupyter, sigamos adelante y aprendamos a escribir y ejecutar scripts estándar de Python. Para hacer esto, usaremos Visual Studio Code, un poderoso editor de texto con gran soporte para Python.

## Código de Visual Studio

En esta sección, instalaremos y configuraremos *Código de Visual Studio* (VS Code), un editor de texto de código abierto y gratuito de Microsoft. Después de presentar sus componentes más importantes, escribiremos un primer script de Python y lo ejecutaremos de diferentes formas. Sin embargo, para empezar, explicaré cuándo usaremos los cuadernos de Jupyter en lugar de ejecutar scripts de Python y por qué elegí VS Code para este libro.

Si bien los cuadernos de Jupyter son increíbles para los flujos de trabajo interactivos como la investigación, la enseñanza y la experimentación, son menos ideales si desea escribir scripts de Python orientados a un entorno de producción que no necesita las capacidades de visualización de los cuadernos. Además, los proyectos más complejos que involucran muchos archivos y desarrolladores son difíciles de administrar con los cuadernos de Jupyter. En este caso, desea utilizar un editor de texto adecuado para escribir y ejecutar archivos Python clásicos. En teoría, podría usar casi cualquier editor de texto (incluso el Bloc de notas funcionaría), pero en realidad, quiere uno que "entienda" Python. Es decir, un editor de texto que admite al menos las siguientes funciones:

### *Resaltado de sintaxis*

El editor colorea las palabras de manera diferente en función de si representan una función, una cadena, un número, etc. Esto hace que sea mucho más fácil leer y comprender el código.

### *Autocompletar*

Autocompletar o *IntelliSense*, como lo llama Microsoft, sugiere automáticamente componentes de texto para que tenga que escribir menos, lo que genera menos errores.

Y muy pronto, tendrá otras necesidades a las que le gustaría acceder directamente desde el editor:

### *Ejecutar código*

Cambiar entre el editor de texto y un indicador Anaconda externo (es decir, símbolo del sistema o terminal) para ejecutar su código puede ser una molestia.

### *Depurador*

Un depurador le permite recorrer el código línea por línea para ver qué está pasando.

### *Control de versiones*

Si usa Git para controlar la versión de sus archivos, tiene sentido manejar las cosas relacionadas con Git directamente en el editor para que no tenga que alternar entre dos aplicaciones.

Existe un amplio espectro de herramientas que pueden ayudarte con todo eso y, como es habitual, cada desarrollador tiene diferentes necesidades y preferencias. Algunos pueden querer usar un editor de texto nofrills junto con un símbolo del sistema externo. Otros pueden preferir un *entorno de desarrollo integrado* (IDE): los IDE intentan poner todo lo que necesitará en una sola herramienta, lo que puede hacerlos inflar.

Elegí VS Code para este libro, ya que se ha convertido rápidamente en uno de los editores de código más populares entre los desarrolladores después de su lanzamiento inicial en 2015: en el [Encuesta para desarrolladores de StackOverflow 2019](#), resultó ser el entorno de desarrollo más popular. ¿Qué hace que VS Code sea una herramienta tan popular? En esencia, es la combinación perfecta entre un editor de texto básico y un IDE completo: VS Code es un mini IDE que viene con todo lo que necesita para programar de inmediato, pero no más:

#### *Multiplataforma*

VS Code se ejecuta en Windows, macOS y Linux. También hay versiones alojadas en la nube como [Espacios de código de GitHub](#).

#### *Herramientas integradas*

VS Code viene con un depurador, soporte para el control de versiones de Git y tiene una Terminal integrada que puede usar como Anaconda Prompt.

#### *Extensiones*

Todo lo demás, por ejemplo, la compatibilidad con Python, se agrega a través de extensiones que se pueden instalar con un solo clic.

#### *Ligero*

Dependiendo de su sistema operativo, el instalador de VS Code ocupa solo 50–100 MB.



#### **Visual Studio Code frente a Visual Studio**

¡No confunda Visual Studio Code con Visual Studio, el IDE! Si bien puede usar Visual Studio para el desarrollo de Python (viene con PTVS, el *Herramientas de Python para Visual Studio*), es una instalación realmente pesada y se usa tradicionalmente para trabajar con lenguajes .NET como C #.

Para saber si está de acuerdo con mis elogios por VS Code, no hay mejor manera que instalarlo y probarlo usted mismo. ¡La siguiente sección te ayuda a empezar!

## Instalacion y configuracion

Descargue el instalador del [Página de inicio de VS Code](#). Para obtener las últimas instrucciones de instalación, consulte siempre los documentos oficiales.

### Ventanas

Haga doble clic en el instalador y acepte todos los valores predeterminados. Luego abra VS Code a través del menú Inicio de Windows, donde lo encontrará en Visual Studio Code.

### Mac OS

Haga doble clic en el archivo ZIP para descomprimir la aplicación. Entonces arrastra y suelta *Visual Studio Code.app* en el *Aplicaciones* carpeta: ahora puede iniciarla desde el Launchpad. Si la aplicación no se inicia, vaya a Preferencias del sistema> Seguridad y privacidad> General y elija Abrir de todos modos.

Cuando abre VS Code por primera vez, parece [Figura 2-5](#). Tenga en cuenta que he cambiado del tema oscuro predeterminado a un tema claro para que las capturas de pantalla sean más fáciles de leer.

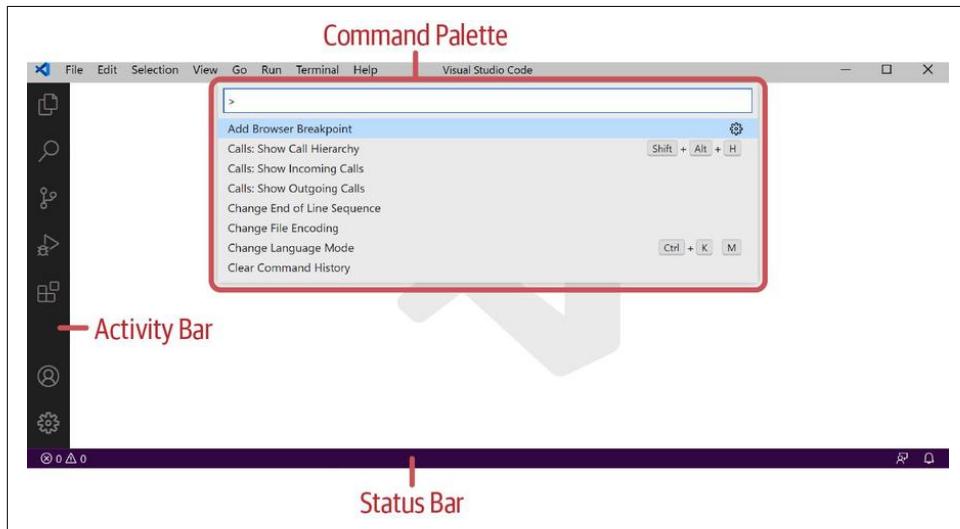


Figura 2-5. Código de Visual Studio

#### Barra de actividades

En el lado izquierdo, verá la barra de actividad con los siguientes iconos de arriba a abajo:

- Explorador
- Buscar
- Fuente de control

- Correr
- Extensiones

#### *Barra de estado*

En la parte inferior del editor, tienes la barra de estado. Una vez que haya completado la configuración y edite un archivo de Python, verá el intérprete de Python aparecer allí.

#### *Paleta de comandos*

Puede mostrar la paleta de comandos a través de F1 o con el atajo de teclado Ctrl + Mayús + P (Windows) o Comando-Mayús-P (macOS). Si no está seguro de algo, su primera parada siempre debe ser la paleta de comandos, ya que le brinda acceso fácil a casi todo lo que puede hacer con VS Code. Por ejemplo, si está buscando atajos de teclado, escriba **atajos de teclado**, seleccione la entrada "Ayuda: Referencia de métodos abreviados de teclado" y presione Entrar.

VS Code es un excelente editor de texto listo para usar, pero para que funcione bien con Python, hay algunas cosas más para configurar: haga clic en el ícono Extensiones en la barra de actividad y busque Python. Instale la extensión oficial de Python que muestra a Microsoft como autor. La instalación llevará un momento y, una vez hecho, es posible que deba hacer clic en el botón Recargar requerido para finalizar; alternativamente, también puede reiniciar VS Code por completo. Finalice la configuración según su plataforma:

#### *Ventanas*

Abra la paleta de comandos y escriba **shell predeterminado**. Seleccione la entrada que dice "Terminal: seleccione Shell predeterminado" y presione Entrar. En el menú desplegable, seleccione Símbolo del sistema y confirme presionando Enter. Esto es necesario porque, de lo contrario, VS Code no puede activar correctamente los entornos Conda.

#### *Mac OS*

Abra la paleta de comandos y escriba **comando de shell**. Seleccione la entrada que dice "Comando de Shell: Instale el comando 'código' en PATH" y presione Enter. Esto es necesario para que pueda iniciar VS Code cómodamente desde Anaconda Prompt (es decir, la Terminal).

Ahora que VS Code está instalado y configurado, júsemoslo para escribir y ejecutar nuestro primer script de Python!

#### **Ejecutando una secuencia de comandos de Python**

Si bien puede abrir VS Code a través del menú Inicio en Windows o Launchpad en macOS, a menudo es más rápido abrir VS Code desde Anaconda Prompt, donde puede iniciarlo a través del código mando. Por lo tanto, abra un nuevo indicador de Anaconda y cambie al directorio donde desea trabajar usando el `CD` comando, luego indique a VS Code que abra el directorio actual (representado por el punto):

```
(base)> cd C:\ Usuarios \nombre de usuario\ python-para-excel  
(base)> codigo.
```

Iniciar VS Code de esta manera hará que el Explorador en la barra de actividad muestre automáticamente el contenido del directorio en el que se encontraba cuando ejecutó el código mando.

Alternativamente, también puede abrir un directorio a través de Archivo> Abrir carpeta (en macOS: Archivo > Abrir), pero esto podría causar errores de permisos en macOS cuando comenzamos a usar xlwings en la Parte IV. Cuando pase el cursor sobre la lista de archivos en el Explorador en la barra de actividad, verá aparecer el botón Nuevo archivo como se muestra en Figura 2-6. Haga clic en Nuevo archivo y llame a su archivo *hello\_world.py*, luego presione Enter. Una vez que se abre en el editor, escribe la siguiente línea de código:

```
impresión("¡Hola Mundo!")
```

¿Recuerda que los cuadernos de Jupyter imprimen convenientemente el valor de retorno de la última línea de forma automática? Cuando ejecuta un script de Python tradicional, debe decirle a Python explícitamente qué imprimir, por lo que debe usar el impresión funcionar aquí. En la barra de estado, ahora debería ver su versión de Python, por ejemplo, "Python 3.8.5 64-bit (conda)". Si hace clic en él, la paleta de comandos se abrirá y le permitirá seleccionar un intérprete de Python diferente si tiene más de uno (esto incluye entornos Conda). Su configuración ahora debería verse como la de Figura 2-6.

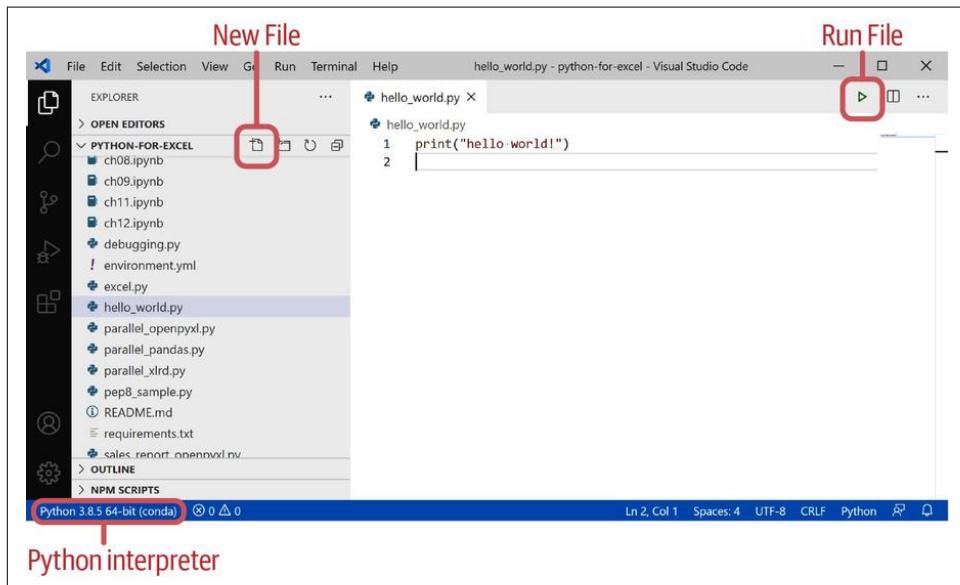


Figura 2-6. VS Code con *hello\_world.py* abierto

Antes de que podamos ejecutar el script, asegúrese de guardar lo presionando Ctrl + S en Windows o Command-S en macOS. Con los cuadernos de Jupyter, simplemente podríamos seleccionar una celda y presionar Shift + Enter para ejecutar esa celda. Con VS Code, puede ejecutar su código desde Anaconda Prompt o haciendo clic en el botón Ejecutar. Ejecutar código Python desde Anaconda Prompt es la forma más probable de ejecutar scripts que están en un servidor, por lo que es importante saber cómo funciona.

### Aviso Anaconda

Abra un mensaje de Anaconda, CD en el directorio con el script, luego ejecute el script así:

```
(base)> cd C:\ Usuarios \nombre de usuario\ python-para-excel  
base> python hello_world.py¡Hola Mundo!
```

La última línea es la salida que imprime el script. Tenga en cuenta que si no está en el mismo directorio que su archivo Python, debe usar la ruta completa a su archivo Python:

```
(base)> python C:\ Usuarios \nombre de usuario\ python-para-excel \ hello_world.py  
¡Hola Mundo!
```



Rutas de archivo largas en el indicador Anaconda

Una forma conveniente de lidar con rutas de archivo largas es arrastrar y soltar el archivo en su Anaconda Prompt. Esto escribirá la ruta completa donde sea que esté el cursor.

### Mensaje de Anaconda en VS Code

No necesita cambiar de VS Code para trabajar con Anaconda Prompt: VS Code tiene una Terminal integrada que puede mostrar a través del atajo de teclado Ctrl + ` o vía Ver > Terminal. Dado que se abre en la carpeta del proyecto, no es necesario que cambie el directorio primero:

```
(base)> python hello_world.py  
¡Hola Mundo!
```

### Botón Ejecutar en Código VS

En el código VS, hay una manera fácil de ejecutar su código sin tener que usar el indicador Anaconda: cuando edita un archivo Python, verá un ícono verde de Reproducir en la parte superior derecha; este es el botón Ejecutar archivo, como se muestra en la [Figura 2-6](#). Al hacer clic en él, se abrirá la Terminal en la parte inferior automáticamente y se ejecutará el código allí.



### Abrir archivos en VS Code

VS Code tiene un comportamiento predeterminado poco convencional cuando hace un solo clic en un archivo en el Explorador (barra de actividad): el archivo se abre en modo de vista previa, lo que significa que el siguiente archivo en el que haga clic lo reemplazará en la pestaña a menos que tenga hizo algunos cambios en el archivo. Si desea desactivar el comportamiento de un solo clic (por lo que un solo clic seleccionará un archivo y un doble clic lo abrirá), vaya a Preferencias > Configuración (Ctrl +, en Windows o Command-, en macOS) y configure el menú desplegable en Workbench> "Lista: Modo abierto" en "doubleClick".

En este punto, sabe cómo crear, editar y ejecutar scripts de Python en VS Code. Sin embargo, VS Code puede hacer bastante más: en [apéndice B](#), Explico cómo usar el depurador y cómo se pueden ejecutar los cuadernos de Jupyter con VS Code.

## Editores de texto alternativos e IDE

Las herramientas son algo individual, y el hecho de que este libro esté basado en los cuadernos de notas de Jupyter y VS Code no significa que no deba echar un vistazo a otras opciones.

Algunos editores de texto populares incluyen:

### *Texto sublime*

**Sublime** es un editor de texto comercial rápido.

### *Bloc de notas ++*

**Bloc de notas ++** es gratuito y ha existido durante mucho tiempo, pero es solo para Windows.

### *Vim o Emacs*

**Empuje** o **Emacs** Puede que no sean las mejores opciones para los programadores principiantes debido a su pronunciada curva de aprendizaje, pero son muy populares entre los profesionales. La rivalidad entre los dos editores gratuitos es tan grande que Wikipedia la describe como la "guerra de los editores".

Los IDE populares incluyen:

### *PyCharm*

los **PyCharm** La edición comunitaria es gratuita y muy potente, mientras que la edición profesional es comercial y añade soporte para herramientas científicas y desarrollo web.

### *Spyder*

**Spyder** es similar al IDE de MATLAB y viene con un explorador de variables. Dado que está incluido en la distribución de Anaconda, puede intentarlo ejecutando lo siguiente emitiendo un mensaje de Anaconda: (base)> **espía**.

### *JupyterLab*

**JupyterLab** es un IDE basado en web desarrollado por el equipo que está detrás de los cuadernos de Jupyter y, por supuesto, puede ejecutar cuadernos de Jupyter. Aparte de eso, intenta integrar todo lo demás que necesita para sus tareas de ciencia de datos en una sola herramienta.

### *IDE de Wing Python*

**IDE de Wing Python** es un IDE que existe desde hace mucho tiempo. Hay versiones simplificadas gratuitas y una versión comercial llamada Wing Pro.

### *IDE de Komodo*

**IDE de Komodo** es un IDE comercial desarrollado por ActiveState y es compatible con muchos otros lenguajes además de Python.

### *PyDev*

**PyDev** es un IDE de Python basado en el popular IDE de Eclipse.

## Conclusión

En este capítulo, le mostré cómo instalar y usar las herramientas con las que trabajaremos: Anaconda Prompt, Jupyter notebooks y VS Code. También ejecutamos un poquito de código Python en un REPL de Python, en un cuaderno de Jupyter y como script en VS Code.

Te recomiendo que te sientas cómodo con Anaconda Prompt, ya que te dará mucho poder una vez que te acostumbres. La capacidad de trabajar con cuadernos de Jupyter en la nube también es muy cómoda, ya que le permite ejecutar las muestras de código de las tres primeras partes de este libro en su navegador.

Con un entorno de desarrollo funcional, ahora está listo para abordar el próximo capítulo, donde aprenderá suficiente Python para poder seguir el resto del libro.



# Empezando con Python

Con Anaconda instalado y los cuadernos de Jupyter en funcionamiento, tiene todo en su lugar para comenzar con Python. Aunque este capítulo no va mucho más allá de lo básico, todavía cubre mucho terreno. Si está al comienzo de su carrera como codificador, es posible que haya mucho que digerir. Sin embargo, la mayoría de los conceptos se aclararán una vez que los use en capítulos posteriores como parte de un ejemplo práctico, por lo que no debe preocuparse si no comprende algo completamente la primera vez. Siempre que Python y VBA difieran significativamente, señalaré esto para asegurarme de que pueda realizar la transición de VBA a Python sin problemas y sea consciente de las trampas obvias. Si no ha hecho ningún VBA antes, no dude en ignorar estas partes.

Comenzaré este capítulo con los tipos de datos básicos de Python, como enteros y cadenas. Después de eso, presentaré la indexación y el corte, un concepto central en Python que le brinda acceso a elementos específicos de una secuencia. A continuación, se encuentran las estructuras de datos como listas y diccionarios que pueden contener varios objetos. Seguiré con elsi declaración y el por y tiempo bucles antes de llegar a la introducción de funciones y módulos que le permiten organizar y estructurar su código. Para concluir este capítulo, le mostraré cómo formatear su código Python correctamente. Como probablemente ya habrá adivinado, este capítulo es lo más técnico posible. Por lo tanto, ejecutar los ejemplos usted mismo en un cuaderno de Jupyter es una buena idea para hacer que todo sea un poco más interactivo y divertido. Escriba los ejemplos usted mismo o ejecútelo utilizando los cuadernos proporcionados en el repositorio complementario.

## Tipos de datos

Python, como cualquier otro lenguaje de programación, trata los números, el texto, los valores booleanos, etc. de manera diferente asignándoles una *tipo de datos*. Los tipos de datos que usaremos con mayor frecuencia son enteros, flotantes, booleanos y cadenas. En esta sección, voy a

Preséntelos uno tras otro con algunos ejemplos. Sin embargo, para poder comprender los tipos de datos, primero necesito explicar qué es un objeto.

## Objetos

En Python, *todo* es un objeto, incluidos números, cadenas, funciones y todo lo demás que veremos en este capítulo. Los objetos pueden hacer que las cosas complejas sean fáciles e intuitivas al brindarle acceso a un conjunto de variables y funciones. Entonces, antes que nada, ¡permítanme decir algunas palabras sobre variables y funciones!

### Variables

En Python, un *variable* es un nombre que asigna a un objeto mediante el signo igual. En la primera línea del siguiente ejemplo, el nombre *a* está asignado al objeto 3:

```
En [1]: a = 3
        B = 4
        a + B
```

Fuera [1]: 7

Esto funciona igual para todos los objetos, que es más simple en comparación con VBA, donde usa el signo igual para tipos de datos como números y cadenas y el Colocar declaración para objetos como libros u hojas de trabajo. En Python, cambia el tipo de una variable simplemente asignándola a un nuevo objeto. Esto se conoce como *mecanografía dinámica*:

```
En [2]: a = 3
        impresión(a)
        a = "Tres"
        impresión(a)
```

3  
Tres

A diferencia de VBA, Python distingue entre mayúsculas y minúsculas, por lo que *a* y *A* son dos variables diferentes. Los nombres de las variables deben seguir ciertas reglas:

- Deben comenzar con una letra o un guión bajo
- Deben constar de letras, números y guiones bajos.

Después de esta breve introducción a las variables, ¡veamos cómo podemos hacer llamadas a funciones!

### Funciones

Presentaré funciones con mucho más detalle más adelante en este capítulo. Por ahora, simplemente debería saber cómo llamar a funciones integradas como *impresión* que usamos en la muestra de código anterior. Para llamar a una función, agregue paréntesis al nombre de la función y proporcione los argumentos entre paréntesis, que es bastante equivalente a la notación matemática:

nombre de la función(argumento1, argumento2, ...)

¡Veamos ahora cómo funcionan las variables y funciones en el contexto de los objetos!

## Atributos y métodos

En el contexto de los objetos, las variables se denominan *atributos* y las funciones se llaman *métodos*: los atributos le dan acceso a los datos de un objeto y los métodos le permiten realizar una acción. Para acceder a los atributos y métodos, usa la notación de puntos así: `myobject.attribute` y `myobject.method ()`.

Hagamos esto un poco más tangible: si escribes un juego de carreras de coches, lo más probable es que uses un objeto que represente un coche. Loscoche el objeto podría tener un velocidad atributo que le permite obtener la velocidad actual a través de velocidad del coche, y es posible que pueda acelerar el automóvil llamando al método de aceleración `coche.acelerar (10)`, lo que aumentaría la velocidad en diez millas por hora.

El tipo de un objeto y con eso su comportamiento está definido por un *clase*, por lo que el ejemplo anterior requiere que escribas un Coche clase. El proceso de obtener uno coche objeto de un Coche la clase se llama *instanciación*, y crea una instancia de un objeto llamando a la clase de la misma manera que llama a una función: `coche = Coche ()`. No escribiremos nuestras propias clases en este libro, pero si está interesado en cómo funciona esto, eche un vistazo a [Apéndice C](#).

Usaremos un método de primer objeto en la siguiente sección para hacer una cadena de texto en mayúsculas, y volveremos al tema de objetos y clases cuando hablaremos de fecha y hora objetos hacia el final de este capítulo. Ahora, sin embargo, sigamos adelante con aquellos objetos que tienen un tipo de datos numérico.

## Tipos numéricos

Los tipos de datos En t y flotador representan *enteros* y *Números de punto flotante*, respectivamente. Para averiguar el tipo de datos de un objeto determinado, utilice la función incorporada `escribe`:

En [3]: `escribe(4)`

Fuera [3]: `int`

En [4]: `escribe(4.4)`

Fuera [4]: `flotar`

Si quiere forzar un número a ser un flotador en lugar de un En t, es lo suficientemente bueno para usar un punto decimal final o el flotador constructor:

En [5]: `escribe(4.)`

Fuera [5]: `flotar`

En [6]: `flotador(4)`

Fuera [6]: 4.0

El último ejemplo también se puede dar la vuelta: usando el `int` constructor, puedes convertir un flotador en una `int`. Si la parte fraccionaria no es cero, se truncará:

En [7]: `int(4.9)`

Fuera [7]: 4



Las celdas de Excel siempre almacenan flotantes

Es posible que deba convertir un flotador a una `int` cuando lee un número de una celda de Excel y lo proporciona como argumento a una función de Python que espera un número entero. La razón es que los números en las celdas de Excel siempre se almacenan como flotantes detrás de escena, incluso si Excel muestra lo que parece un número entero.

Python tiene algunos tipos numéricos más que no usaré ni discutiré en este libro: están los fracción decimal, y complejo tipos de datos. Si las inexactitudes de punto flotante son un problema (consulte la barra lateral), utilice `decimal` escriba para obtener resultados exactos. Sin embargo, estos casos son muy raros. Como regla general: si Excel fuera lo suficientemente bueno para los cálculos, use flotantes.

### Inexactitudes de punto flotante

De forma predeterminada, Excel a menudo muestra números redondeados: escriba `=1.125-1.1` en una celda, y verás 0,025. Si bien esto puede ser lo que espera, no es lo que Excel almacena internamente. Cambie el formato de visualización para mostrar al menos 16 decimales, y cambiará a 0.0249999999999999. Este es el efecto de *inexactitud de punto flotante*: las computadoras viven en un mundo binario, es decir, calculan solo con ceros y unos. Ciertas fracciones decimales como 0,1 no se puede almacenar como un número de coma flotante binario finito, lo que explica el resultado de la resta. En Python, verá el mismo efecto, pero Python no le oculta los decimales:

En [8]: `1.125 - 1.1` Fuera [8]:

0.0249999999999999

### Operadores matemáticos

Calcular con números requiere el uso de operadores matemáticos como el signo más o menos. A excepción del operador de energía, no debería sorprendernos si viene de Excel:

En [9]: `3 + 4 # Suma`

Fuera [9]: 7

```
En [10]: 3 - 4      # Resta
Fuera [10]: -1
En [11]: 3 / 4      # División
Fuera [11]: 0,75
En [12]: 3 * 4      # Multiplicación
Fuera [12]: 12
En [13]: 3**4      # El operador de energía (Excel usa 3 ^ 4)
Fuera [13]: 81
En [14]: 3 * (3 + 4)    # Uso de paréntesis
Fuera [14]: 21
```

### Comentarios

En los ejemplos anteriores, estaba describiendo el funcionamiento del ejemplo usando *comentarios* (p.ej, `# Suma`). Los comentarios ayudan a otras personas (y a usted mismo unas semanas después de escribir el código) a comprender lo que está sucediendo en su programa. Es una buena práctica comentar solo aquellas cosas que aún no son evidentes al leer el código: en caso de duda, es mejor no tener ningún comentario que un comentario desactualizado que contradiga el código. Todo lo que comience con un signo de almohadilla es un comentario en Python y se ignora cuando ejecuta el código:

```
En [15]: # Esta es una muestra que hemos visto antes.
          # Cada línea de comentario debe comenzar con #3
          + 4
```

Fuera [15]: 7

```
En [dieciséis]: 3 + 4 # Este es un comentario en línea
```

Fuera [16]: 7

La mayoría de los editores tienen un atajo de teclado para comentar / descomentar líneas. En los cuadernos Jupyter y VS Code, es Ctrl + / (Windows) o Command- / (macOS). Tenga en cuenta que las celdas de Markdown en los cuadernos de Jupyter no aceptarán comentarios; si comienza una línea con un `#` allí, Markdown lo interpretará como un encabezado.

Habiendo cubierto los números enteros y flotantes, ¡pasemos directamente a la siguiente sección sobre booleanos!

### Booleanos

Los tipos booleanos en Python son Cierto o Falso, exactamente como en VBA. Los operadores booleanos `y`, `o`, `y no`, sin embargo, están todos en minúsculas, mientras que VBA los muestra

en mayúsculas. Las expresiones booleanas son similares a cómo funcionan en Excel, excepto por los operadores de igualdad y desigualdad:

En [17]: `3 == 4` # Igualdad (Excel usa `3 = 4`)

Fuera [17]: Falso

En [18]: `3 != 4` Fuera # Desigualdad (Excel usa `3 <> 4`)

[18]: Verdadero

En [19]: `3 < 4` Fuera # Menor que. Utilice `>` para más grande que.

[19]: Verdadero

En [20]: `3 <= 4` Fuera # Más pequeño o igual. Utilice `=` para mayor o igual.

[20]: Verdadero

En [21]: # Puedes encadenar expresiones lógicas

# En VBA, esto sería: `10 < 12 y 12 < 17`

# En fórmulas de Excel, esto sería: = Y(`10 < 12, 12 < 17`)

Fuera [21]: Verdadero

En [22]: `no` Cierto # "no "operador Fuera

[22]: Falso

En [23]: Falso `y` Cierto # operador "y"

Fuera [23]: Falso

En [24]: Falso `o` Cierto Fuera # operador "o"

[24]: Verdadero

Cada objeto de Python se evalúa como Cierto o Falso. La mayoría de los objetos son Cierto, pero hay algunos que evalúan Falso incluso Ninguno (ver barra lateral), Falso, 0 o tipos de datos vacíos, por ejemplo, una cadena vacía (introduciré cadenas en la siguiente sección).

### Ninguno

Ninguno es una constante incorporada y representa "la ausencia de un valor" según los documentos oficiales. Por ejemplo, si una función no devuelve nada explícitamente, devuelve Ninguno. También es una buena opción representar celdas vacías en Excel como veremos en [Parte III](#) y [Parte IV](#).

Para comprobar si un objeto es Cierto o Falso, utilizar el bool constructor:

En [25]: `bool(2)` Fuera

[25]: Verdadero

En [26]: `bool(0)`

Fuera [26]: Falso

En [27]: bool("algún texto") Fuera # Llegaremos a las cuerdas en un momento

[27]: Verdadero

En [28]: bool("")

Fuera [28]: Falso

En [29]: bool(Ninguno)

Fuera [29]: Falso

Con los valores booleanos en nuestro bolsillo, queda un tipo de datos básico más: datos textuales, mejor conocidos como *instrumentos de cuerda*.

#### Instrumentos de cuerda

Si alguna vez ha trabajado con cadenas en VBA que son más largas que una línea y contienen variables y comillas literales, probablemente deseaba que fuera más fácil. Afortunadamente, esta es un área donde Python es particularmente fuerte. Las cadenas se pueden expresar usando comillas dobles ("") o comillas simples (''). La única condición es que debe comenzar y terminar la cadena con el mismo tipo de comillas. Puede usar + para concatenar cadenas o \* para repetir. Como ya le mostré el caso repetido cuando probaba Python REPL en el capítulo anterior, aquí hay una muestra con el signo más:

En [30]: "Una cadena de comillas dobles". + 'Una cadena de comillas simples.'Out

[30]: 'Una cadena de comillas dobles. Una cadena de comillas simples'.

Dependiendo de lo que desee escribir, el uso de comillas simples o dobles puede ayudarlo a imprimir fácilmente comillas literales sin necesidad de escapar de ellas. Si aún necesita escapar de un carácter, lo precede con una barra invertida:

En [31]: impresión("¡No espere!" + 'Aprenda a "hablar" Python.') ¡No

espere! Aprenda a "hablar" Python.

En [32]: impresión("Es fácil de \ "escapar\ " personajes con un protagonista \\\ ".")

Es fácil "escapar" de los caracteres con \.

Cuando mezcla cadenas con variables, normalmente trabaja con *cuerdas f*, corto para *literal de cadena formateada*. Simplemente ponga un F delante de su cadena y use variables entre llaves:

En [33]: # Note cómo Python le permite asignar convenientemente múltiples  
# valores a múltiples variables en una sola líneafirst\_adjective,  
segundo\_adjetivo = "gratis", "fuente abierta" F"Python es  
{first\_adjective} y {second\_adjective}".

Out [33]: "Python es gratuito y de código abierto".

Como mencioné al principio de esta sección, las cadenas son objetos como todo lo demás y ofrecen algunos métodos (es decir, funciones) para realizar una acción en esa cadena. Por ejemplo, así es como se transforma entre letras mayúsculas y minúsculas:

En [34]: "PITÓN".[más bajo\(\)](#)

Fuera [34]: 'python'

En [35]: "[pitón](#)".[superior\(\)](#)

Fuera [35]: 'PYTHON'

#### Obteniendo ayuda

¿Cómo saber qué atributos ofrecen ciertos objetos como cadenas y qué argumentos aceptan sus métodos? La respuesta depende un poco de la herramienta que utilice: con los cuadernos de Jupyter, presione la tecla Tab después de escribir el punto que sigue a un objeto, por ejemplo "pitón".<Tab>. Esto hará que aparezca un menú desplegable con todos los atributos y métodos que ofrece este objeto. Si su cursor está en un método, por ejemplo, entre paréntesis de "python ".upper (), presione Shift + Tab para obtener la descripción de esa función. VS Code mostrará esta información automáticamente como información sobre herramientas. Si ejecuta Python REPL en Anaconda Prompt, usedir ("python") para obtener los atributos disponibles y ayuda ("python" .upper) para imprimir la descripción del superior método. Aparte de eso, siempre es una buena idea volver a Python[documentación en línea](#). Si está buscando la documentación de paquetes de terceros como pandas, es útil buscarlos en[PyPI](#), El índice de paquetes de Python, donde encontrará los enlaces a las respectivas páginas de inicio y documentación.

Cuando se trabaja con cadenas, una tarea normal es seleccionar partes de una cadena: por ejemplo, es posible que desee obtener el Dólar estadounidense parte de la EURUSD notación de tipo de cambio. La siguiente sección le muestra el poderoso mecanismo de indexación y división de Python que le permite hacer exactamente esto.

## Indexación y corte

La indexación y el corte le dan acceso a elementos específicos de una secuencia. Dado que las cadenas son secuencias de caracteres, podemos usarlas para aprender cómo funciona. En la siguiente sección, conoceremos secuencias adicionales como listas y tuplas que también admiten la indexación y la división.

## Indexación

Figura 3-1 introduce el concepto de *indexación*. Python está basado en cero, lo que significa que el primer elemento de una secuencia se denomina índice0. Índices negativos de -1 le permite hacer referencia a elementos del final de la secuencia.

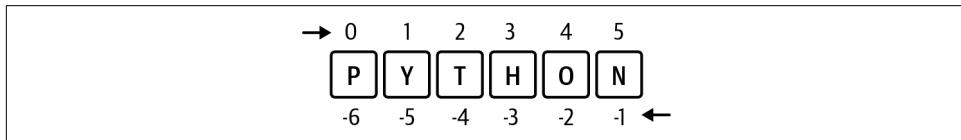


Figura 3-1. Indexación desde el principio y el final de una secuencia



Trampas de errores comunes para desarrolladores de VBA

Si viene de VBA, la indexación es una trampa de error común. VBA usa indexación basada en uno para la mayoría de las colecciones, como hojas (Sábanas (1)) pero usa indexación de base cero para matrices (MyArray (0)), aunque ese valor predeterminado se puede cambiar. Otra diferencia es que VBA usa paréntesis para indexar mientras que Python usa corchetes.

La sintaxis para indexar es la siguiente:

`secuencia[índice]`

En consecuencia, accede a elementos específicos de una cadena como esta:

En [36]: idioma = "PITÓN" En [37]

]: idioma[0] Fuera [37]: 'P'

En [38]: idioma[1]

Fuera [38]: 'Y'

En [39]: idioma[-1]

Fuera [39]: 'N'

En [40]: idioma[-2]

Fuera [40]: 'O'

A menudo, querrá extraer más que un solo carácter; aquí es donde entra en juego el corte.

## Rebanar

Si desea obtener más de un elemento de una secuencia, utilice el *rebanar* sintaxis, que funciona de la siguiente manera:

```
secuencia[comienzo:parada:paso]
```

Python usa intervalos semiabiertos: el comienzo se incluye el índice mientras que el parada el índice no lo es. Si dejas el comienzo o parada argumentos de distancia, incluirá todo desde el principio o hasta el final de la secuencia, respectivamente. paso determina la dirección y el tamaño del paso: por ejemplo, 2 devolverá cada segundo elemento de izquierda a derecha y -3 devolverá cada tercer elemento de derecha a izquierda. El tamaño de paso predeterminado es uno:

```
En [41]: idioma[:3]      # Igual que el idioma [0: 3]
```

```
Fuera [41]: 'PYT'
```

```
En [42]: idioma[1:3]
```

```
Fuera [42]: 'YT'
```

```
En [43]: idioma[-3:] Fuera  # Igual que el idioma [-3: 6]
```

```
[43]: 'HON'
```

```
En [44]: idioma[-3:-1] Fuera
```

```
[44]: 'HO'
```

```
En [45]: idioma[::-2] # Cada segundo elementoFuera
```

```
[45]: 'PTO'
```

```
En [46]: idioma[-1:-4:-1] # El paso negativo va de derecha a izquierdaFuera
```

```
[46]: 'NOH'
```

Hasta ahora, hemos analizado solo una operación de índice o segmento, pero Python también le permite *cadena* múltiples operaciones de índice y corte juntas. Por ejemplo, si desea obtener el segundo carácter de los últimos tres caracteres, puede hacerlo así:

```
En [47]: idioma[-3:] [1] Fuera
```

```
[47]: 'O'
```

Esto es lo mismo que idioma [-2] así que en este caso, no tendría mucho sentido usar encadenamiento, pero tendrá más sentido cuando usemos indexación y segmentación con listas, una de las estructuras de datos que voy a presentar en la siguiente sección.

## Estructuras de datos

Python ofrece potentes estructuras de datos que facilitan mucho el trabajo con una colección de objetos. En esta sección, voy a presentar listas, diccionarios, tuplas y conjuntos. Si bien cada una de estas estructuras de datos tiene características ligeramente diferentes, todas son

capaz de sostener múltiples objetos. En VBA, es posible que haya utilizado colecciones o matrices para contener varios valores. VBA incluso ofrece una estructura de datos llamada diccionario que funciona conceptualmente igual que el diccionario de Python. Sin embargo, solo está disponible en la versión de Windows de Excel lista para usar. Comencemos con las listas, la estructura de datos que probablemente usará más.

## Liza

*Liza* son capaces de contener múltiples objetos de diferentes tipos de datos. Son tan versátiles que los usará todo el tiempo. Crea una lista de la siguiente manera:

```
[elemento1, elemento2, ...]
```

Aquí hay dos listas, una con los nombres de los archivos de Excel y la otra con algunos números:

```
En [48]: nombres_archivo = ["one.xlsx", "dos.xlsx", "tres.xlsx"]
números = [1, 2, 3]
```

Al igual que las cadenas, las listas se pueden concatenar fácilmente con el signo más. Esto también le muestra que las listas pueden contener diferentes tipos de objetos:

```
En [49]: nombres_archivo + números
```

```
Fuera [49]: ['one.xlsx', 'two.xlsx', 'three.xlsx', 1, 2, 3]
```

Como las listas son objetos como todo lo demás, las listas también pueden tener otras listas como elementos. Me referiré a ellos como *listas anidadas*:

```
En [50]: lista_anidada = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Si reorganiza esto para abarcar varias líneas, puede reconocer fácilmente que se trata de una representación muy agradable de una matriz o un rango de celdas de la hoja de cálculo. Tenga en cuenta que los corchetes le permiten implícitamente romper las líneas (consulte el recuadro). A través de la indexación y el corte, obtiene los elementos que desea:

```
En [51]: células = [[1, 2, 3],
[4, 5, 6],
[7, 8, 9]]
```

```
En [52]: células[1] # Segunda fila
```

```
Fuera [52]: [4, 5, 6]
```

```
En [53]: células[1][1:] # Segunda fila, segunda y tercera columna
```

```
Fuera [53]: [5, 6]
```

### Continuación de línea

A veces, una línea de código puede ser tan larga que deberá dividirla en dos o más líneas para mantener su código legible. Técnicamente, puede usar paréntesis o una barra invertida para dividir la línea:

En [54]: `a = (1 + 2  
+ 3)`

En [55]: `a = 1 + 2 \  
+ 3`

La guía de estilo de Python, sin embargo, prefiere que use *saltos de línea implícitos*. Si es posible: siempre que utilice una expresión que contenga paréntesis, corchetes o llaves, utilícelas para introducir un salto de línea sin tener que introducir un carácter adicional. Diré más sobre la guía de estilo de Python hacia el final de este capítulo.

Puede cambiar elementos en listas:

En [56]: `usuarios = ["Linda", "Brian"]`

En [57]: `usuarios.adjuntar("Jennifer") # Lo más común es que agregue al final  
usuarios`

Fuera [57]: `['Linda', 'Brian', 'Jennifer']`

En [58]: `usuarios.insertar(0, "Kim") # Inserte "Kim" en el índice 0  
usuarios`

Fuera [58]: `['Kim', 'Linda', 'Brian', 'Jennifer']`

Para eliminar un elemento, use música pop o del. Tiempo música pop es un método, del se implementa como una declaración en Python:

En [59]: `usuarios.música pop() # Elimina y devuelve el último elemento de forma predeterminada.`

Fuera [59]: `'Jennifer'`

En [60]: `usuarios`

Fuera [60]: `['Kim', 'Linda', 'Brian']`

En [61]: `del usuarios[0] # del elimina un elemento en el índice dado`

Algunas otras cosas útiles que puede hacer con las listas son:

En [62]: `len(usuarios) # Largo`

Fuera [62]: `2`

En [63]: `"Linda" en usuarios # Comprueba si los usuarios contienen "Linda"`

Fuera [63]: `Verdadero`

```
En [64]: impresión(ordenado(usuarios))      # Devuelve una nueva lista ordenada
          impresión(usuarios)      # La lista original no ha cambiado
```

```
['Brian', 'Linda']
['Linda', 'Brian']
```

```
En [sesenta y cinco]: usuarios.clasificar()      # Ordena la lista original
                     usuarios
```

```
Fuera [65]: ['Brian', 'Linda']
```

Tenga en cuenta que puede usar len y en con cuerdas también:

```
En [66]: len("Pitón")
```

```
Fuera [66]: 6
```

```
En [67]: "gratis" en "Python es gratuito y de código abierto ".Fuera
```

```
[67]: Verdadero
```

Para tener acceso a los elementos de una lista, se hace referencia a ellos por su posición o índice; eso no siempre es práctico. Los diccionarios, el tema de la siguiente sección, le permiten acceder a los elementos a través de una clave (a menudo un nombre).

## Diccionarios

*Diccionarios* mapear claves a valores. Encontrará combinaciones clave / valor todo el tiempo. La forma más sencilla de crear un diccionario es la siguiente:

```
{clave1: valor1, clave2: valor2, ...}
```

Mientras que las listas le permiten acceder a elementos por índice, es decir, posición, los diccionarios le permiten acceder a elementos por clave. Al igual que con los índices, se accede a las claves mediante corchetes. Los siguientes ejemplos de código utilizarán un par de divisas (clave) que se correlaciona con el tipo de cambio (valor):

```
En [68]: los tipos de cambio = {"EURUSD": 1.1152,
          "GBPUSD": 1.2454,
          "AUDUSD": 0.6161}
```

```
En [69]: los tipos de cambio["EURUSD"] # Acceda al tipo de cambio EURUSDFuera
```

```
[69]: 1.1152
```

Los siguientes ejemplos le muestran cómo cambiar los valores existentes y agregar nuevos pares clave / valor:

```
En [70]: los tipos de cambio["EURUSD"] = 1.2 # Cambiar un valor existente
          los tipos de cambio
```

```
Salida [70]: {'EURUSD': 1.2, 'GBPUSD': 1.2454, 'AUDUSD': 0.6161}
```

```
En [71]: los tipos de cambio["CADUSD"] = 0.714 # Agregar un nuevo par clave / valor
          los tipos de cambio
```

```
Salida [71]: {'EURUSD': 1.2, 'GBPUSD': 1.2454, 'AUDUSD': 0.6161, 'CADUSD': 0.714}
```

La forma más sencilla de fusionar dos o más diccionarios es mediante *desembalaje* en uno nuevo. Desempaquetá un diccionario usando dos asteriscos iniciales. Si el segundo diccionario contiene claves del primero, se anularán los valores del primero. Puede ver que esto sucede mirando el GBPUSD tipo de cambio:

```
En [72]: {**los tipos de cambio, **{"SGDUSD": 0,7004, "GBPUSD": 1.2222}}
```

```
Fuera [72]: {'EURUSD': 1.2,  
             'GBPUSD': 1.2222,  
             'AUDUSD': 0,6161,  
             'CADUSD': 0,714,  
             'SGDUSD': 0,7004}
```

Python 3.9 introdujo el carácter de barra vertical como un operador de combinación dedicado para diccionarios, lo que le permite simplificar la expresión anterior a esto:

```
los tipos de cambio | {"SGDUSD": 0,7004, "GBPUSD": 1.2222}
```

Muchos objetos pueden servir como claves; el siguiente es un ejemplo con números enteros:

```
En [73]: monedas = {1: "EUR", 2: "DÓLAR ESTADOUNIDENSE", 3:
```

```
"AUD"}En [74]: monedas[1] Fuera [74]: 'EUR'
```

Usando el obtener método, los diccionarios le permiten usar un valor predeterminado en caso de que la clave no exista:

```
En [75]: # monedas [100] plantearía una excepción. En lugar de 100,  
          # también puede usar cualquier otra clave que no exista.  
          monedas.obtener(100, "N / A")
```

```
Fuera [75]: 'N / A'
```

Los diccionarios se pueden utilizar a menudo cuando se utilizaría un Caso declaración en VBA. El ejemplo anterior podría escribirse así en VBA:

```
Seleccione el caso X  
Caso 1  
Depurar.Impresión "EUR"  
Caso 2  
Depurar.Impresión "DÓLAR  
ESTADOUNIDENSE"Caso 3  
Depurar.Impresión "AUD"  
Caso otro  
Depurar.Impresión "N / A"  
Finalizar Seleccionar
```

Ahora que sabe cómo trabajar con diccionarios, pasemos a la siguiente estructura de datos: tuplas. Son similares a las listas con una gran diferencia, como veremos en la siguiente sección.

## Tuplas

*Tuplas* son similares a las listas con la diferencia de que son *inmutable*: una vez creados, sus elementos no se pueden cambiar. Si bien a menudo puede usar tuplas y listas de manera intercambiable, las tuplas son la opción obvia para una colección que nunca cambia a lo largo del programa. Las tuplas se crean separando valores con comas:

```
mytuple = elemento1, element2, ...
```

El uso de paréntesis a menudo facilita la lectura:

```
En [76]: monedas = ("EUR", "GBP", "AUD")
```

Las tuplas le permiten acceder a elementos de la misma forma que las listas, pero no le permitirán cambiar elementos. En cambio, la concatenación de tuplas creará una nueva tupla detrás de escena, luego vinculará su variable a esta nueva tupla:

```
En [77]: monedas[0] # Accediendo al primer elementoFuera
```

```
[77]: 'EUR'
```

```
En [78]: # La concatenación de tuplas devolverá una nueva tupla.  
monedas + ("SGD",)
```

```
Salida [78]: ('EUR', 'GBP', 'AUD', 'SGD')
```

Explico la diferencia entre *mudable* vs. *inmutable* objetos en detalle en [Apéndice C](#), pero por ahora, echemos un vistazo a la última estructura de datos de esta sección: conjuntos.

## Conjuntos

*Conjuntos* son colecciones que no tienen elementos duplicados. Si bien puede usarlos para operaciones de teoría de conjuntos, en la práctica a menudo lo ayudan a obtener los valores únicos de una lista o tupla. Crea conjuntos usando llaves:

```
{elemento1, element2, ...}
```

Para obtener los objetos únicos en una lista o tupla, use el colocar constructor así:

```
En [79]: colocar(["DÓLAR ESTADOUNIDENSE", "DÓLAR ESTADOUNIDENSE", "SGD", "EUR", "DÓLAR
```

```
ESTADOUNIDENSE", "EUR"]) Salida [79]: {'EUR', 'SGD', 'USD'}
```

A parte de eso, puede aplicar operaciones de teoría de conjuntos como intersección y unión:

```
En [80]: portfolio1 = {"DÓLAR ESTADOUNIDENSE", "EUR", "SGD", "CHF"}  
portfolio2 = {"EUR", "SGD", "CANALLA"}
```

```
En [81]: # Igual que portfolio2.union(portfolio1)  
portfolio1.Unión(portfolio2)
```

```
Salida [81]: {'CAD', 'CHF', 'EUR', 'SGD', 'USD'}
```

```
En [82]: # Igual que portfolio2.intersection(portfolio1)  
portfolio1.intersección(portfolio2)
```

Fuera [82]: {'EUR', 'SGD'}

Para obtener una descripción general completa de las operaciones de conjuntos, consulte la [documentos oficiales](#).

Antes de continuar, revisemos rápidamente las cuatro estructuras de datos que acabamos de conocer en [Tabla 3-1](#).

Muestra una muestra para cada estructura de datos en la notación que usé en los párrafos anteriores, la llamada *literales*. Además, también estoy enumerando sus constructores que ofrecen una alternativa al uso de literales y que a menudo se usan para convertir de una estructura de datos a otra. Por ejemplo, para convertir una tupla en una lista, haga lo siguiente:

```
En [83]: monedas = "DÓLAR ESTADOUNIDENSE", "EUR", "CHF"  
monedas
```

```
Salida [83]: ('USD', 'EUR', 'CHF')En [  
84]: lista(monedas) Fuera [84]:  
['USD', 'EUR', 'CHF']
```

*Tabla 3-1. Estructuras de datos*

Estructura de datos	Literales	Constructor
Lista	[1, 2, 3]	lista ((1, 2, 3))
Diccionario	{"a": 1, "b": 2}	dict(a = 1, b = 2)
Tupla	(1, 2, 3)	tupla ((1, 2, 3))
Colocar	{1, 2, 3}	conjunto ((1, 2, 3))

En este punto, conoce todos los tipos de datos importantes, incluidos los básicos, como flotantes y cadenas, y las estructuras de datos, como listas y diccionarios. En la siguiente sección, pasamos al control del flujo.

## Flujo de control

Esta sección presenta la `si` declaración así como la `por` y `tiempo` bucles. La `si` declaración le permite ejecutar ciertas líneas de código solo si se cumple una condición, y la `por` y `tiempo` los bucles ejecutarán un bloque de código repetidamente. Al final de la sección, también presentaré listas por comprensión, que son una forma de construir listas que pueden servir como una alternativa a los bucles. Comenzaré esta sección con la definición de bloques de código, para lo cual necesito presentar una de las particularidades más notables de Python: espacios en blanco significativos.

## Bloques de código y declaración de paso

A *bloque de código* define una sección en su código fuente que se usa para algo especial. Por ejemplo, usa un bloque de código para definir las líneas sobre las cuales su programa está en bucle o constituye la definición de una función. En Python, usted define bloques de código al sangrarlos, no al usar palabras clave como en VBA o llaves como en la mayoría

otros idiomas. Esto se conoce como *espacio en blanco significativo*. La comunidad de Python se ha establecido en cuatro espacios como sangría, pero generalmente los escribe presionando la tecla Tab: tanto los cuadernos de Jupyter como VS Code convertirán automáticamente su tecla Tab en cuatro espacios. Permítame mostrarle cómo se definen formalmente los bloques de código mediante el uso de la declaración:

```
si condición:  
    aprobar # Hacer nada
```

La línea que precede al bloque de código siempre termina con dos puntos. Dado que se llega al final del bloque de código cuando ya no se aplica sangría a la línea, debe utilizar la declaración aprobar si desea crear un bloque de código ficticio que no hace nada. En VBA, esto correspondería a lo siguiente:

```
Si condición Luego  
    'Hacer nada  
Terminara si
```

Ahora que sabe cómo definir bloques de código, comencemos a usarlos en la siguiente sección, donde presentaré correctamente las declaraciones si.

## La declaración if y las expresiones condicionales

Para presentar la declaración si, permítanme reproducir el ejemplo de “[Legibilidad y capacidad de mantenimiento](#)” en la página 13 en [Capítulo 1](#), pero esta vez en Python:

```
En [85]: I = 20  
        si I < 5:  
            impresión("i es menor que 5")  
        elif I <= 10:  
            impresión("i está entre 5 y 10")  
        demás:  
            impresión("yo es mayor que 10")
```

yo es mayor que 10

Si hicieras lo mismo que hicimos nosotros en [Capítulo 1](#), es decir, sangra el elif y demás declaraciones, obtendrías una Error de sintaxis. Python no le permitirá sangrar su código de manera diferente a la lógica. En comparación con VBA, las palabras clave están en minúsculas y en lugar de De lo contrario en VBA, Python usa elif. Las declaraciones son una forma fácil de saber si un programador es nuevo en Python o si ya ha adoptado un *Pitónico* estilo: en Python, una simple si declaración no requiere ningún paréntesis alrededor y para probar si un valor es Certo, no es necesario que lo haga explícitamente. Esto es lo que quiero decir con eso:

```
En [86]: es importante = Certo  
        si es importante:  
            impresión("Esto es importante.")  
        demás:  
            impresión("Esto no es importante.")
```

Esto es importante.

Lo mismo funciona si desea verificar si una secuencia como una lista está vacía o no:

```
En [87]: valores = []
si valores:
    impresión("Se proporcionaron los siguientes valores: {valores}")
demás:
    impresión("No se proporcionaron valores.")
```

No se proporcionaron valores.

Los programadores que vienen de otros lenguajes suelen escribir algo como si (es\_importante == Verdadero) o si len (valores)> 0 en lugar de.

*Expresiones condicionales*, también llamado *operadores ternarios*, le permite utilizar un estilo más compacto para si / si no declaraciones:

```
En [88]: es_importante = Falso
impresión("importante") si es_importante más imprimir("no importante")
no importante
```

Con si declaraciones y expresiones condicionales en nuestro bolsillo, dirijamos nuestra atención a por y tiempo bucles en la siguiente sección.

## Los bucles for y while

Si necesita hacer algo repetidamente, como imprimir el valor de diez variables diferentes, se está haciendo un gran favor al no copiar / pegar la declaración de impresión diez veces. En su lugar, use unpor bucle para hacer el trabajo por usted. por los bucles iteran sobre los elementos de una secuencia como una lista, una tupla o una cadena (recuerde, las cadenas son secuencias de caracteres). Como ejemplo introductorio, creemos unpor bucle que toma cada elemento del monedas lista, la asigna a la variable divisa y lo imprime, uno tras otro hasta que no haya más elementos en la lista:

```
En [89]: monedas = ["DÓLAR ESTADOUNIDENSE", "HKD", "AUD"]
```

```
por divisa en monedas:
    impresión(divisa)
```

Dólar estadounidense

HKD

AUD

Como nota al margen, VBA's Para cada declaración está cerca de cómo Python por el bucle funciona. El ejemplo anterior podría escribirse así en VBA:

Oscuro monedas Como Variante  
Oscuro curr Como Variante 'moneda es una palabra reservada en VBA

```
monedas = Formación("DÓLAR ESTADOUNIDENSE", "HKD", "AUD")
```

## Para cada curr En monedas

Depurar.Imprimir curr

próximo

En Python, si necesita una variable de contador en un bucle, el distancia o enumerar Los elementos incorporados pueden ayudarte con eso. Primero veamos distancia, que proporciona una secuencia de números: usted lo llama proporcionando un solo parada argumento o proporcionando un comienzo y parada argumento, con un opcional paso argumento. Como con rebanar,comienzo es inclusivo, parada es exclusivo, y paso determina el tamaño del paso, con 1 siendo el predeterminado:

```
distancia(parada)
distancia(comienzo, parada, paso)
```

distancia evalúa perezosamente, lo que significa que sin pedirlo explícitamente, no verá la secuencia que genera:

En [90]: distancia(5)

Fuera [90]: rango (0, 5)

Convertir el rango en una lista resuelve este problema:

En [91]: lista(distancia(5)) # detener la

discusiónFuera [91]: [0, 1, 2, 3, 4]

En [92]: lista(distancia(2, 5, 2)) # Arrancar, detener, avanzar argumentos

Fuera [92]: [2, 4]

La mayor parte del tiempo, no es necesario envolver distancia con un lista, aunque:

En [93]: por I en distancia(3):

impresión(I)

```
0
1
2
```

Si necesita una variable de contador mientras recorre una secuencia, use enumerar. Devuelve una secuencia de (índice, elemento) tuplas. De forma predeterminada, el índice comienza en cero y se incrementa en uno. Puedes usar enumerate en un bucle como este:

En [94]: por I, divisa en enumerar(monedas):

impresión(I, divisa)

```
0 USD
1 HKD
2 AUD
```

Hacer un bucle sobre tuplas y conjuntos funciona igual que con las listas. Cuando recorre los diccionarios, Python recorre las claves:

```
En [95]: los tipos de cambio = {"EURUSD": 1.1152,  
                                "GBPUSD": 1.2454,  
                                "AUDUSD": 0.6161}  
por currency_pair en los tipos de cambio:  
    impresión(currency_pair)
```

```
EURUSD  
GBPUSD  
AUDUSD
```

Usando el elementos método, obtienes la clave y el valor al mismo tiempo que la tupla:

```
En [96]: por currency_pair, tipo de cambio en los tipos de cambio.elementos():  
    impresión(currency_pair, tipo de cambio)
```

```
EURUSD 1.1152  
GBPUSD 1.2454  
AUDUSD 0.6161
```

Para salir de un bucle, use el rotura declaración:

```
En [97]: por I en distancia(15):  
    si I == 2:  
        rotura  
    demás:  
        impresión(I)
```

```
0  
1
```

Omite el resto de un bucle con el Seguir declaración, lo que significa que la ejecución continúa con un nuevo bucle y el siguiente elemento:

```
En [98]: por I en distancia(4):  
    si I == 2:  
        Seguir  
    demás:  
        impresión(I)
```

```
0  
1  
3
```

Al comparar los bucles for en VBA con Python, hay una diferencia sutil: en VBA, la variable de contador aumenta más allá de su límite superior después de terminar el bucle:

```
Para I = 1 Para 3  
    Depurar.Imprimir i  
próximo I  
Depurar.Imprimir i
```

Esto imprime:

```
1  
2  
3  
4
```

En Python, se comporta como probablemente esperarías:

En [99]: por I en distancia(1, 4):

```
    impresión(I)  
    impresión(I)
```

```
1  
2  
3  
3
```

En lugar de recorrer una secuencia, también puede usar *while bucles* para ejecutar un bucle mientras una determinada condición es verdadera:

En [100]: norte = 0

```
    tiempo norte <= 2:  
        impresión(norte)  
        norte += 1
```

```
0  
1  
2
```



### Asignación aumentada

He usado el *asignación aumentada* notación en el último ejemplo: `n += 1`. Esto es lo mismo que si escribieras `n = n + 1`. También funciona con todos los demás operadores matemáticos que he introducido anteriormente; por ejemplo, por menos podrías escribir `n -= 1`.

Muy a menudo, deberá recopilar ciertos elementos en una lista para su posterior procesamiento. En este caso, Python ofrece una alternativa a los bucles de escritura: lista, diccionario y comprensión de conjuntos.

## Comprendiciones de listas, diccionarios y conjuntos

Las comprensiones de listas, diccionarios y conjuntos son técnicamente una forma de crear la estructura de datos respectiva, pero a menudo reemplazan una por loop, por lo que los presento aquí. Suponga que en la siguiente lista de pares de divisas USD, le gustaría seleccionar aquellas divisas en las que USD se cotiza como segunda divisa. Podrías escribir lo siguiente por círculo:

En [101]: pares\_de\_monedas = ["USDJPY", "USDGBP", "USDCHF",  
 "USDCAD", "AUDUSD", "NZDUSD"]

```
En [102]: usd_quote = []
por par en pares_de_monedas:
    si par[3] == "DÓLAR ESTADOUNIDENSE":
        usd_quote.adjuntar(par[3])
    usd_quote
```

Fuera [102]: ['AUD', 'NZD']

Esto suele ser más fácil de escribir con un *lista de comprensión*. La comprensión de una lista es una forma concisa de crear una lista. Puede obtener su sintaxis de este ejemplo, que hace lo mismo que el anterior por círculo:

```
En [103]: [par[:3] por par en pares_de_monedas si par[3] == "DÓLAR
ESTADOUNIDENSE"] Fuera [103]: ['AUD', 'NZD']
```

Si no tiene ninguna condición que satisfacer, simplemente deje el si parte lejos. Por ejemplo, para invertir todos los pares de divisas de modo que la primera divisa quede en segundo lugar y viceversa, haría lo siguiente:

```
En [104]: [par[3:] + par[:3] por par en pares_de_monedas] Fuera [104]:
['JPYUSD', 'GBPUSD', 'CHFUSD', 'CADUSD', 'USDAUD', 'USDNZD']
```

Con los diccionarios, existen comprensiones de diccionario:

```
En [105]: los tipos de cambio = {"EURUSD": 1.1152,
                    "GBPUSD": 1.2454,
                    "AUDUSD": 0.6161}
{k: v * 100 por (k, v) en los tipos de cambio.elementos()}
```

Salida [105]: {'EURUSD': 111.52, 'GBPUSD': 124.54, 'AUDUSD': 61.61}

Y con conjuntos, hay comprensiones de conjuntos:

```
En [106]: {s + "DÓLAR ESTADOUNIDENSE" por s en ["EUR", "GBP", "EUR", "HKD",
"HKD"]} Fuera [106]: {'EURUSD', 'GBPUSD', 'HKDUSD'}
```

En este punto, ya puede escribir scripts simples, ya que conoce la mayoría de los componentes básicos de Python. En la siguiente sección, aprenderá cómo organizar su código para que se pueda mantener cuando sus scripts comiencen a crecer.

## Organización del código

En esta sección, veremos cómo llevar código a una estructura mantenible: comenzaré presentando funciones con todos los detalles que comúnmente necesitará antes de mostrarle cómo dividir su código en diferentes módulos de Python. El conocimiento sobre los módulos nos permitirá terminar esta sección mirando en el fecha y hora módulo que forma parte de la biblioteca estándar.

## Funciones

Incluso si usa Python solo para scripts simples, seguirá escribiendo funciones con regularidad: son una de las construcciones más importantes de todos los lenguajes de programación y le permiten reutilizar las mismas líneas de código desde cualquier lugar de su programa.

Comenzaremos esta sección definiendo una función antes de ver cómo llamarla.

### Definición de funciones

Para escribir su propia función en Python, debe usar la palabra clave `def`, que significa función *definición*. A diferencia de VBA, Python no diferencia entre una función y un procedimiento Sub. En Python, el equivalente de un procedimiento Sub es simplemente una función que no devuelve nada. Las funciones en Python siguen la sintaxis de los bloques de código, es decir, terminan la primera línea con dos puntos y sangra el cuerpo de la función:

```
def nombre_de_la_función(argumento_requerido, argumento_opcional=valor por defecto, ...):  
    regreso valor1, valor2, ...
```

#### Argumentos requeridos

Los argumentos obligatorios no tienen un valor predeterminado. Los argumentos múltiples están separados por comas.

#### Argumentos opcionales

Hace que un argumento sea opcional proporcionando un valor predeterminado. Ninguno se utiliza a menudo para hacer que un argumento sea opcional si no hay un valor predeterminado significativo.

#### Valor devuelto

los `regreso` declaración define el valor que devuelve la función. Si lo deja a un lado, la función regresa automáticamente `Ninguno`. Python convenientemente le permite devolver múltiples valores separados por comas.

Para poder jugar con una función, definamos una que pueda convertir la temperatura de Fahrenheit o Kelvin a grados Celsius:

```
En [107]: def convert_to_celsius(grados, fuente="fahrenheit"):  
    si fuente.más bajo() == "fahrenheit":  
        regreso (grados-32) * (5/9)  
    elif fuente.más bajo() == "Kelvin":  
        regreso grados - 273.15  
    demás:  
        regreso F"No sé cómo realizar una conversión desde {fuente}"
```

Estoy usando el método de cadena `más bajo`, que transforma las cadenas proporcionadas a minúsculas. Esto nos permite aceptar la fuente cadena con cualquier letra mayúscula mientras que la comparación seguirá funcionando. Con la `convert_to_celsius` función definida, veamos cómo podemos llamarla!

### Funciones de llamada

Como se mencionó brevemente al principio de este capítulo, usted llama a una función agregando paréntesis al nombre de la función, encerrando los argumentos de la función:

```
valor1, valor2, ... = nombre de la función(positional_arg, arg_name=valor, ...)
```

#### Argumentos posicionales

Si proporciona un valor como argumento posicional (positional\_arg), los valores se hacen coincidir con los argumentos de acuerdo con su posición en la definición de la función.

#### Argumentos de palabras clave

Al proporcionar el argumento en la forma arg\_name = valor, estás proporcionando un argumento de palabra clave. Esto tiene la ventaja de que puede proporcionar los argumentos en cualquier orden. También es más explícito para el lector y puede facilitar su comprensión. Por ejemplo, si la función se define como `f(a, b)`, podría llamar a la función así: `f(b=1, a=2)`. Este concepto también existe en VBA, donde puede usar argumentos de palabras clave llamando a una función como esta: `f(b:=1, a:=1)`.

Juguemos con el `convert_to_celsius` función para ver cómo funciona todo esto en la práctica:

```
En [108]: convert_to_celsius(100, "fahrenheit") # Argumentos posicionales
```

```
[108]: 37.77777777777778
```

```
En [109]: convert_to_celsius(50) # Utilizará la fuente predeterminada (fahrenhe
```

```
[109]: 10.0
```

```
En [110]: convert_to_celsius(fuente="Kelvin", grados=0) # Argumentos de palabras clave
```

```
[110]: -273.15
```

Ahora que sabe cómo definir y llamar funciones, veamos cómo organizarlas con la ayuda de módulos.

## Módulos y declaración de importación

Cuando escriba código para proyectos más grandes, tendrá que dividirlo en diferentes archivos en algún momento para poder llevarlo a una estructura mantenible. Como ya hemos visto en el capítulo anterior, los archivos de Python tienen la extensión `.py` normalmente se refiere a su archivo principal como *texto*. Si ahora desea que su secuencia de comandos principal acceda a la funcionalidad de otros archivos, debe *importar* esa funcionalidad primero. En este contexto, los archivos fuente de Python se denominan *módulos*. Para tener una mejor idea de cómo funciona esto y cuáles son las diferentes opciones de importación, eche un vistazo al archivo `temperature.py` en el repositorio complementario abriendolo con VS Code ([Ejemplo 3-1](#)). Si necesita un repaso sobre cómo abrir archivos en VS Code, eche otro vistazo a [Capítulo 2](#).

### Ejemplo 3-1. temperature.py

```
TEMPERATURE_SCALES = ("fahrenheit", "Kelvin", "Celsius")
```

```
def convert_to_celsius(grados, fuente="fahrenheit"):
    si fuente.más bajo() == "fahrenheit":
        regreso (grados-32) * (5/9)elif
    fuente.más bajo() == "Kelvin":
        regreso grados - 273.15
    demás:
        regreso F"No sé cómo realizar una conversión desde {fuente}"
```

```
impresión("Este es el módulo de temperatura".)
```

Para poder importar el temperatura módulo de su cuaderno Jupyter, necesitará el cuaderno Jupyter y el temperatura módulo para estar en el mismo directorio - como es el caso del repositorio complementario. Para importar, solo usa el nombre del módulo, sin el .py finalizando. Después de ejecutar el importar declaración, tendrá acceso a todos los objetos en ese módulo de Python a través de la notación de puntos. Por ejemplo, usar temperature.convert\_to\_celsius () para realizar su conversión:

En [111]: `importar temperatura`Este

es el módulo de temperatura.

En [112]: `temperatura.TEMPERATURE_SCALES`Fuera

[112]: ('fahrenheit', 'kelvin', 'celsius')

En [113]: `temperatura.convert_to_celsius(120, "fahrenheit")` Fuera

[113]: 48.88888888888889

Tenga en cuenta que utilicé letras mayúsculas para TEMPERATURE\_SCALES para expresar que es una constante, diré más sobre eso hacia el final de este capítulo. Cuando ejecutas la celda con temperatura de importación, Python ejecutará el *temperature.py* archivo de arriba a abajo. Puede ver fácilmente que esto sucede, ya que la importación del módulo activará la función de impresión en la parte inferior de *temperature.py*.



#### Los módulos solo se importan una vez

Si ejecuta el temperatura de importación celda nuevamente, notará que ya no imprime nada. Esto se debe a que los módulos de Python solo se importan una vez por sesión. Si cambia el código en un módulo que importa, debe reiniciar su intérprete de Python para recoger todos los cambios, es decir, en un cuaderno de Jupyter, tendrá que hacer clic en Kernel> Reiniciar.

En realidad, normalmente no imprime nada en módulos. Esto fue solo para mostrarle el efecto de importar un módulo más de una vez. Más comúnmente, pones funciones y clases en tus módulos (para más información sobre clases, consulta [Apéndice C](#)). Si no quieres escribir temperatura cada vez que utilice un objeto del módulo de temperatura, cambie la importación declaración como esta:

En [114]: `importar temperatura como tp` En [115]:

```
tp.TEMPERATURE_SCALES
Fuera [115]: ('fahrenheit',
'kelvin', 'celsius')
```

Asignar un alias corto tp a su módulo puede hacer que sea más fácil de usar mientras siempre está claro de dónde proviene un objeto. Muchos paquetes de terceros sugieren una convención específica cuando se usa un alias. Por ejemplo, pandas está usando importar pandas como pd. Hay una opción más para importar objetos de otro módulo:

En [116]: `de temperatura importar TEMPERATURE_SCALES, convert_to_celsius en [117]`

```
]: TEMPERATURE_SCALES
```

```
Fuera [117]: ('fahrenheit', 'kelvin', 'celsius')
```



### La carpeta `_pycache_`

Cuando importa el módulo temperatura, verá que Python crea una carpeta llamada `_pycache_` con archivos que tienen la extensión .pyc. Estos son archivos compilados por código de bytes que crea el intérprete de Python cuando importa un módulo. Para nuestros propósitos, podemos simplemente ignorar esta carpeta, ya que es un detalle técnico de cómo Python ejecuta su código.

Al usar el comando `from` para importar y sintaxis, solo importa objetos específicos. Al hacer esto, los estás importando directamente a la *espacio de nombres* de su guion principal: es decir, sin mirar las declaraciones de importación, no podrá saber si los objetos importados se definieron en su secuencia de comandos de Python actual o cuaderno de Jupyter o si provienen de otro módulo. Esto podría causar conflictos: si su script principal tiene una función llamada `convert_to_celsius`, anularía la que está importando desde el módulo temperatura. Sin embargo, si usa uno de los dos métodos anteriores, su función local y la del módulo importado podrían vivir a continuación. el uno al otro como `convert_to_celsius` y `temperature.convert_to_celsius`.



### No nombre sus scripts como paquetes existentes

Una fuente común de errores es nombrar su archivo de Python de la misma manera que un paquete o módulo de Python existente. Si crea un archivo para probar algunas funciones de pandas, no llame a ese archivo `pandas.py`, ya que esto puede causar conflictos.

Ahora que sabe cómo funciona el mecanismo de importación, usémoslo de inmediato para importar el fecha y hora ¡módulo! Esto también le permitirá aprender algunas cosas más sobre objetos y clases.

## La clase datetime

Trabajar con fecha y hora es una operación común en Excel, pero viene con limitaciones: por ejemplo, el formato de celda de Excel para la hora no admite unidades más pequeñas que milisegundos y las zonas horarias no son compatibles en absoluto. En Excel, la fecha y la hora se almacenan como un flotador simple llamado *fecha número de serie*. A continuación, se formatea la celda de Excel para mostrarla como fecha y / o hora. Por ejemplo, el 1 de enero de 1900 tiene el número de serie de fecha 1, lo que significa que también es la fecha más temprana con la que puede trabajar en Excel. El tiempo se traduce a la parte decimal del flotador, por ejemplo, 01/01/1900 10:10:00 está representado por 1.423611111.

En Python, para trabajar con fecha y hora, importa el fecha y hora módulo, que forma parte de la biblioteca estándar. Los fecha y hora módulo contiene una clase con el mismo nombre que nos permite crear fecha y hora objetos. Dado que tener el mismo nombre para el módulo y la clase puede resultar confuso, utilizaré la siguiente convención de importación a lo largo de este libro: importar fecha y hora como dt. Esto facilita la diferenciación entre el módulo (dt) y la clase fecha y hora).

Hasta este punto, usábamos la mayor parte del tiempo *literales* para crear objetos como listas o diccionarios. Los literales se refieren a la sintaxis que Python reconoce como un tipo de objeto específico; en el caso de una lista, sería algo como [1, 2, 3]. Sin embargo, la mayoría de los objetos deben crearse llamando a su clase: este proceso se llama *instanciación*, y los objetos, por lo tanto, también se denominan *instancias de clase*. Llamar a una clase funciona de la misma manera que llamar a una función, es decir, agrega paréntesis al nombre de la clase y proporciona los argumentos de la misma manera que lo hicimos con las funciones. Para instanciar una fecha y hora objeto, debe llamar a la clase de esta manera:

```
importar fecha y hora como dt
dt.fecha y hora(año, mes, día, hora, minuto, segundo, microsegundo, zona horaria)
```

Repasemos un par de ejemplos para ver cómo trabaja con fecha y hora objetos en Python. Para el propósito de esta introducción, ignoremos las zonas horarias y trabajemos con time-zone-naive fecha y hora objetos:

```
En [118]: # Importar el módulo de fecha y hora como "dt"
importar fecha y hora como dt
```

```
En [119]: # Cree una instancia de un objeto de fecha y hora llamado "marca de tiempo"
marca de tiempo = dt.fecha y hora(2020, 1, 31, 14, 30)marca
de tiempo
```

```
Fuera [119]: datetime.datetime (2020, 1, 31, 14, 30)
```

En [120]: # Los objetos de fecha y hora ofrecen varios atributos, por ejemplo, obtener el día  
marca de tiempo.día

Fuera [120]: 31

En [121]: # La diferencia de dos objetos de fecha y hora devuelve un objeto timedelta  
marca de tiempo - dt.fecha y hora(2020, 1, 14, 12, 0)

Fuera [121]: datetime.timedelta (días = 17, segundos = 9000)

En [122]: # En consecuencia, también puede trabajar con objetos timedelta  
marca de tiempo + dt.timedelta(dias=1, horas=4, minutos=11)

Fuera [122]: datetime.datetime (2020, 2, 1, 18, 41)

Para *formato* fecha y hora objetos en cadenas, utilice el strftime método, y para *analizar gramaticalmente* una cadena y convertirla en una fecha y hora objeto, use el strptime función (puede encontrar una descripción general de los códigos de formato aceptados en la [documentos de fecha y hora](#)):

En [123]: # Dar formato a un objeto de fecha y hora de una manera específica  
# También puede usar una cadena f: f "{marca de tiempo: % d / % m / % Y % H: % M}" "marca de  
tiempo.strftime("%D/%metro/% Y% H:%METRO")"

Salida [123]: '31 / 01/2020 14:30 '

En [124]: # Analizar una cadena en un objeto de fecha y hora  
dt.fecha y hora.strptime("12.1.2020", "%D.%metro.% Y")

Fuera [124]: datetime.datetime (2020, 1, 12, 0, 0)

Después de esta breve introducción a la fecha y hora módulo, pasemos al último tema de este capítulo, que trata sobre formatear su código correctamente.

## PEP 8: Guía de estilo para código Python

Es posible que se haya preguntado por qué a veces usaba nombres de variables con guiones bajos o en mayúsculas. Esta sección explicará mis opciones de formato al presentarle la guía de estilo oficial de Python. Python utiliza las llamadas propuestas de mejora de Python (PEP) para discutir la introducción de nuevas características del lenguaje. Uno de ellos, la Guía de estilo para código Python, generalmente se conoce por su número: PEP 8. PEP 8 es un conjunto de recomendaciones de estilo para la comunidad Python; si todos los que trabajan con el mismo código se adhieren a la misma guía de estilo, el código se vuelve mucho más legible. Esto es especialmente importante en el mundo del código abierto, donde muchos programadores trabajan en el mismo proyecto, a menudo sin conocerse personalmente. [Ejemplo 3-2](#) muestra un archivo corto de Python que presenta las convenciones más importantes.

### Ejemplo 3-2. pep8\_sample.py

```
'''Este script muestra algunas reglas de PEP 8.''' ❶
```

```

importar fecha y hora como dt ②

TEMPERATURE_SCALES = ("fahrenheit", "Kelvin",
                      "Celsius") ③
④

clase Convertidor de temperatura: ⑤
    aprobar # No hace nada por el momento ⑥

def convert_to_celsius(grados, fuente="fahrenheit"):
    """ Esta función convierte grados Farenheit o Kelvin en
    grados Celsius. ⑧
    """
    si fuente.más bajo() == "fahrenheit": ⑨
        regreso (grados-32) * (5/9)elif ⑩
    fuente.más bajo() == "Kelvin":
        regreso grados - 273.15
    demás:
        regreso F"No sé cómo realizar una conversión desde {fuente}" ⑪

Celsius = convert_to_celsius(44, fuente="fahrenheit")
escalas_no_celsius = TEMPERATURE_SCALES[:-1] ⑫

impresión("Tiempo actual: " + dt.fecha y hora.ahora().isoformat())
impresión(F"La temperatura en grados Celsius es: {celsius}")

```

- ① Explique lo que hace el script / módulo con un *docstring* en la cima. Una cadena de documentos es un tipo especial de cadena, entre comillas triples. Además de servir como una cadena para documentar su código, una cadena de documentos también facilita la escritura de cadenas en varias líneas y es útil si su texto contiene muchas comillas dobles o comillas simples, ya que no necesitará escapar de ellas. También son útiles para escribir consultas SQL multilínea, como veremos en[Capítulo 11](#).
- ② Todas las importaciones están en la parte superior del archivo, una por línea. Enumere primero las importaciones de la biblioteca estándar, luego las de paquetes de terceros y finalmente las de sus propios módulos. Esta muestra solo hace uso de la biblioteca estándar.
- ③ Utilice letras mayúsculas con guiones bajos para las constantes. Utilice una longitud de línea máxima de 79 caracteres. Si es posible, aproveche los paréntesis, los corchetes o las llaves para los saltos de línea implícitos.
- ④ Separe las clases y funciones con dos líneas vacías del resto del código.

- ⑤ A pesar de que muchas clases como fecha y hora están todas en minúsculas, sus propias clases deben usar Palabras en mayúsculas como nombres. Para obtener más información sobre las clases, consulte [Apéndice C](#).
- ⑥ Los comentarios en línea deben estar separados por al menos dos espacios del código. Los bloques de código deben tener una sangría de cuatro espacios.
- ⑦ Las funciones y los argumentos de las funciones deben usar nombres en minúsculas con guiones bajos si mejoran la legibilidad. No use espacios entre el nombre del argumento y su valor predeterminado.
- ⑧ La cadena de documentación de una función también debe enumerar y explicar los argumentos de la función. No he hecho esto aquí para que la muestra sea breve, pero encontrará documentos completos en `excel.py` archivo que se incluye en el repositorio complementario y que nos encontraremos en [Capítulo 8](#).
- ⑨ No use espacios alrededor del colon.
- ⑩ Utilice espacios alrededor de los operadores matemáticos. Si se utilizan operadores con diferentes prioridades, puede considerar agregar espacios alrededor de aquellos con la prioridad más baja solamente. Dado que la multiplicación en este ejemplo tiene la prioridad más baja, agregué espacios a su alrededor.
- ⑪ Utilice nombres en minúsculas para las variables. Utilice guiones bajos si mejoran la legibilidad. Al asignar un nombre de variable, use espacios alrededor del signo igual. Sin embargo, al llamar a una función, no use espacios alrededor del signo igual que se usa con argumentos de palabras clave.
- ⑫ Con la indexación y el corte, no use espacios alrededor de los corchetes.

Este es un resumen simplificado de PEP 8, por lo que es una buena idea echar un vistazo al original. [PEP 8](#) una vez que empieces a ponerte más serio con Python. PEP 8 establece claramente que es una recomendación y que sus propias guías de estilo tendrán prioridad. Después de todo, la coherencia es el factor más importante. Si está interesado en otras pautas disponibles públicamente, es posible que desee echar un vistazo a [Guía de estilo de Google para Python](#), que está razonablemente cerca de PEP 8. En la práctica, la mayoría de los programadores de Python se adhieren vagamente a PEP 8, e ignorar la longitud máxima de línea de 79 caracteres es probablemente el pecado más común.

Dado que puede ser difícil formatear su código correctamente mientras lo escribe, puede hacer que su estilo sea verificado automáticamente. La siguiente sección le muestra cómo funciona esto con VS Code.

## Código PEP 8 y VS

Al trabajar con VS Code, hay una manera fácil de asegurarse de que su código se adhiera a PEP 8: use un *linter*. Un linter verifica su código fuente en busca de errores de estilo y sintaxis.

Encienda la paleta de comandos (Ctrl + Shift + P en Windows o Comando-Shift-P en macOS) y busque Python: seleccione Linter. Una opción popular es *escama8*, un paquete que viene preinstalado con Anaconda. Si está habilitado, VS Code subrayará los problemas con líneas onduladas cada vez que guarde su archivo. Pasar el cursor sobre una línea tan ondulada le dará una explicación en una descripción emergente. Para volver a desactivar un linter, busque "Python: Enable Linting" en la paleta de comandos y seleccione "Disable Linting". Si lo prefiere, también puede ejecutar *escama8* en un indicador Anaconda para que se imprima un informe (el comando solo imprime algo si hay una violación de PEP 8, por lo que ejecutarlo en *pep8\_sample.py* no imprimirá nada a menos que introduzca una infracción):

```
(base)> cd C:\ Usuarios \nombre de usuario\python-para-
excel(base)> flake8 pep8_sample.py
```

Python recientemente ha llevado el análisis de código estático un paso más allá al agregar soporte para *sugerencias de tipo*. La siguiente sección explica cómo funcionan.

### Sugerencias de tipo

En VBA, a menudo ve un código que antepone a cada variable una abreviatura para el tipo de datos, como `strEmployeeName` o `wbWorkbookName`. Si bien nadie le impedirá hacer esto en Python, no se hace comúnmente. Tampoco encontrará un equivalente a los VBA *Opción explícita* o *Oscuro declaración* para declarar el tipo de una variable. En cambio, Python 3.5 introdujo una característica llamada *sugerencias de tipo*. Las sugerencias de tipo también se denominan *escribir anotaciones* y le permite declarar el tipo de datos de una variable. Son completamente opcionales y no tienen ningún efecto sobre cómo el intérprete de Python ejecuta el código (sin embargo, existen paquetes de terceros como *pirantico* que puede imponer sugerencias de tipo en tiempo de ejecución). El propósito principal de las sugerencias de tipo es permitir que los editores de texto como VS Code detecten más errores antes de ejecutar el código, pero también pueden mejorar la autocompletación del código de VS Code y otros editores. El verificador de tipos más popular para el código anotado es *mypy*, que VS Code ofrece como linter. Para tener una idea de cómo funcionan las anotaciones de tipo en Python, aquí hay una pequeña muestra sin sugerencias de tipo:

```
X = 1

def Hola(nombre):
    regreso F"¡Hola, {nombre}!"
```

Y nuevamente con sugerencias de tipo:

X: En t = 1

```
def Hola(nombre: str) -> str:  
    regreso F"¡Hola, {nombre}!"
```

Como las sugerencias de tipo generalmente tienen más sentido en bases de código más grandes, no las usaré en el resto de este libro.

## Conclusión

Este capítulo fue una introducción completa a Python. Conocimos los bloques de construcción más importantes del lenguaje, incluidas estructuras de datos, funciones y módulos. También tocamos algunas de las particularidades de Python, como los espacios en blanco significativos y las pautas de formato de código, mejor conocido como PEP 8. Para continuar con este libro, no necesitará conocer todos los detalles: como principiante, solo conozca las listas y diccionarios, indexación y segmentación, así como cómo trabajar con funciones, módulos, por bucles y si las declaraciones ya te llevarán lejos.

Comparado con VBA, encuentro Python más consistente y poderoso pero al mismo tiempo más fácil de aprender. Si eres un fan acérrimo de VBA y este capítulo no te ha convencido todavía, estoy bastante seguro de que la siguiente parte lo hará: allí, te daré una introducción a los cálculos basados en matrices antes de comenzar nuestro viaje de análisis de datos con el biblioteca de pandas.

Comencemos con [Parte II](#) aprendiendo algunos conceptos básicos sobre NumPy!

---

**PARTE II**

---

## **Introducción a los pandas**



# Fundaciones NumPy

Como puede recordar de [Capítulo 1](#), NumPy es el paquete central para la computación científica en Python, que proporciona soporte para cálculos basados en matrices y álgebra lineal. Como NumPy es la columna vertebral de los pandas, voy a presentar sus conceptos básicos en este capítulo: después de explicar qué es una matriz NumPy, veremos la vectorización y la difusión, dos conceptos importantes que le permiten escribir código matemático conciso y que lo encontrarás de nuevo en pandas. Después de eso, veremos por qué NumPy ofrece funciones especiales llamadas funciones universales antes de terminar este capítulo aprendiendo cómo obtener y establecer valores de una matriz y explicando la diferencia entre una vista y una copia de una matriz NumPy. Incluso si apenas usaremos NumPy directamente en este libro, conocer sus conceptos básicos facilitará el aprendizaje de los pandas en el próximo capítulo.

## Empezando con NumPy

En esta sección, aprenderemos sobre matrices NumPy unidimensionales y bidimensionales y lo que hay detrás de los términos técnicos. *vectorización, radiodifusión, y función universal*.

### Matriz NumPy

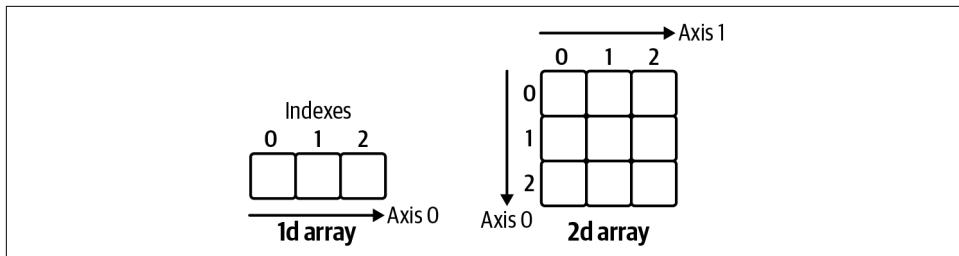
Para realizar cálculos basados en matrices con listas anidadas, como las conocimos en el último capítulo, tendría que escribir algún tipo de bucle. Por ejemplo, para agregar un número a cada elemento en una lista anidada, puede usar la siguiente comprensión de lista anidada:

```
En [1]: matriz = [[1, 2, 3],  
                  [4, 5, 6],  
                  [7, 8, 9]]
```

```
En [2]: [[I + 1 por I en hilera] por hilera en matriz]
```

```
Fuera [2]: [[2, 3, 4], [5, 6, 7], [8, 9, 10]]
```

Esto no es muy legible y, lo que es más importante, si lo hace con matrices grandes, recorrer cada elemento se vuelve muy lento. Dependiendo de su caso de uso y del tamaño de las matrices, calcular con matrices NumPy en lugar de listas de Python puede hacer que sus cálculos sean desde un par de veces hasta alrededor de cien veces más rápido. NumPy logra este rendimiento mediante el uso de código escrito en C o Fortran; estos son lenguajes de programación compilados que son mucho más rápidos que Python. Una matriz NumPy es una matriz N-dimensional para *datos homogéneos*. Homogéneo significa que todos los elementos de la matriz deben ser del mismo tipo de datos. Más comúnmente, se trata de matrices de flotadores unidimensionales y bidimensionales como se muestra esquemáticamente en [Figura 4-1](#).



*Figura 4-1. Una matriz NumPy unidimensional y bidimensional*

Creamos una matriz unidimensional y bidimensional para trabajar a lo largo de este capítulo:

En [3]: # Primero, importemos NumPy

```
importar numpy como notario público
```

En [4]: # La construcción de una matriz con una lista simple da como resultado una matriz 1d

```
array1 = notario público.formación([10, 100, 1000.])
```

En [5]: # La construcción de una matriz con una lista anidada da como resultado una matriz 2d

```
array2 = notario público.formación([[1, 2, 3],  
[4., 5., 6.]])
```



#### Dimensión de matriz

Es importante tener en cuenta la diferencia entre una matriz unidimensional y bidimensional: una matriz unidimensional tiene solo un eje y, por lo tanto, no tiene una orientación explícita de columna o fila. Si bien esto se comporta como matrices en VBA, es posible que tenga que acostumbrarse si proviene de un lenguaje como MATLAB, donde las matrices unidimensionales siempre tienen una orientación de columna o fila.

Incluso si array1 consta de números enteros excepto por el último elemento (que es un flotante), la homogeneidad de las matrices NumPy obliga a que el tipo de datos de la matriz sea float64, que es capaz de acomodar todos los elementos. Para conocer el tipo de datos de una matriz, acceda a su dtype atributo:

```
En [6]: array1.dtypeFuera
```

```
[6]: dtype ('float64')
```

Ya que `dtype` te devuelve `float64` en lugar de flotador que conocimos en el último capítulo, es posible que haya adivinado que NumPy usa sus propios tipos de datos numéricos, que son más granulares que los tipos de datos de Python. Sin embargo, esto generalmente no es un problema, ya que la mayoría de las veces, la conversión entre los diferentes tipos de datos en Python y NumPy ocurre automáticamente. Si alguna vez necesita convertir explícitamente un tipo de datos NumPy en uno de los tipos de datos básicos de Python, simplemente use el constructor correspondiente (diré más sobre cómo acceder a un elemento desde una matriz en breve):

```
En [7]: flotador(array1[0])
```

```
Fuera [7]: 10.0
```

Para obtener una lista completa de los tipos de datos de NumPy, consulte la [Documentos de NumPy](#). Con las matrices NumPy, puede escribir código simple para realizar cálculos basados en matrices, como veremos a continuación.

## Vectorización y difusión

Si construye la suma de un escalar y una matriz NumPy, NumPy realizará una operación de elemento, lo que significa que no tiene que recorrer los elementos usted mismo. La comunidad de NumPy se refiere a esto como *vectorización*. Te permite escribir código conciso, representando prácticamente la notación matemática:

```
En [8]: array2 + 1
```

```
Salida [8]: matriz ([[2., 3., 4.],  
[5., 6., 7.]])
```



Escalar

*Escalar* se refiere a un tipo de datos básico de Python como un flotante o una cadena. Esto es para diferenciarlos de estructuras de datos con múltiples elementos como listas y diccionarios o matrices NumPy unidimensionales y bidimensionales.

El mismo principio se aplica cuando trabaja con dos matrices: NumPy realiza la operación por elementos:

```
En [9]: array2 * array2
```

```
Fuera [9]: matriz ([[1., 4., 9.],  
[16., 25., 36.]])
```

Si usa dos matrices con diferentes formas en una operación aritmética, NumPy extiende, si es posible, la matriz más pequeña automáticamente a través de la matriz más grande para que sus formas sean compatibles. Se llama *radiodifusión*:

En [10]: `array2 * array1`

Fuera [10]: matriz ([[ 10., 200., 3000.],  
[ 40., 500., 6000.]])

Para realizar multiplicaciones de matrices o productos escalares, use el operador @:<sup>1</sup>

En [11]: `array2 @ array2.T # array2.T es un atajo para array2.transpose ()`

Fuera [11]: matriz ([[14., 32.],  
[32., 77.]])

¡No se deje intimidar por la terminología que he introducido en esta sección, como escalar, vectorización o difusión! Si alguna vez ha trabajado con matrices en Excel, todo esto debería sentirse muy natural como se muestra en Figura 4-2. La captura de pantalla está tomada de `array_calculations.xlsx`, que encontrarás en el *SG* directorio del repositorio complementario.

	A	B	C	D	E	F	G	H	I	J
1	array1									
2		10	100	1000						
3					array2 + 1	2	3	4		
4	array2					5	6	7		
5		1	2	3						
6		4	5	6	array2 * array2	1	4	9		
7						16	25	36		
8										
9										
10					array2 * array1	10	200	3000		
11						40	500	6000		
12										
13					array2 @ array2.T	14	32			
14						32	77			
15								=MMULT(A5:C6, TRANSPOSE(A5:C6))		

Figura 4-2. Cálculos basados en matrices en Excel

Ahora sabe que las matrices realizan operaciones aritméticas por elementos, pero ¿cómo se puede aplicar una función a cada elemento de una matriz? Para eso están las funciones universales.

## Funciones universales (ufunc)

Funciones universales (ufunc) funcionan en todos los elementos de una matriz NumPy. Por ejemplo, si usa la función de raíz cuadrada estándar de Python de la Matemáticas módulo en una matriz NumPy, obtendrá un error:

En [12]: `importar Matemáticas`

En [13]: `Matemáticas.sqrt(array2) # Esto generará un error`

<sup>1</sup> Si ha pasado un tiempo desde su última clase de álgebra lineal, puede omitir este ejemplo; la multiplicación de matrices no es algo sobre lo que se basa este libro.

```
-----  
TypeError Traceback (última llamada más reciente) <ipython-input-13-5c37e8f41094>  
en <module>  
----> 1 math.sqrt (array2) # Esto generará un error
```

TypeError: solo las matrices de tamaño 1 se pueden convertir a escalares de Python

Por supuesto, podría escribir un bucle anidado para obtener la raíz cuadrada de cada elemento, luego construir una matriz NumPy nuevamente a partir del resultado:

```
En [14]: notario público.formación([[Matemáticas.sqrt(I) por I en hilera] por hilera en array2])
```

```
Salida [14]: matriz ([[1. , 1.41421356, 1.73205081],  
[2. , 2.23606798, 2.44948974]])
```

Esto funcionará en los casos en que NumPy no ofrece ufunc y la matriz es lo suficientemente pequeña. Sin embargo, si NumPy tiene ufunc, utilícelo, ya que será mucho más rápido con arreglos grandes, además de ser más fácil de escribir y leer:

```
En [15]: notario público.sqrt(array2)
```

```
Salida [15]: matriz ([[1. , 1.41421356, 1.73205081],  
[2. , 2.23606798, 2.44948974]])
```

Algunos de los ufuncs de NumPy, como suma, también están disponibles como métodos de matriz: si desea la suma de cada columna, haga lo siguiente:

```
En [diecisésis]: array2.suma(eje=0) # Devuelve una matriz 1d
```

```
Salida [16]: matriz ([5., 7., 9.])
```

El argumento eje = 0 se refiere al eje a lo largo de las filas mientras eje = 1 se refiere al eje a lo largo de las columnas, como se muestra en [Figura 4-1](#). Dejando el eje argumento de distancia resume toda la matriz:

```
En [17]: array2.suma()
```

```
Fuera [17]: 21.0
```

Encontrará más ufuncs de NumPy a lo largo de este libro, ya que se pueden usar con pandas DataFrames.

Hasta ahora, siempre hemos trabajado con toda la matriz. La siguiente sección le muestra cómo manipular partes de una matriz e introduce algunos constructores de matriz útiles.

## Creación y manipulación de matrices

Comenzaré esta sección obteniendo y configurando elementos específicos de una matriz antes de introducir algunos constructores de matriz útiles, incluido uno para crear números pseudoaleatorios que podría usar para una simulación de Monte Carlo. Concluiré esta sección explicando la diferencia entre una vista y una copia de una matriz.

## Obtención y configuración de elementos de matriz

En el último capítulo, le mostré cómo indexar y dividir listas para obtener acceso a elementos específicos. Cuando trabaja con listas anidadas comomatriz del primer ejemplo de este capítulo, puede utilizar *indexación encadenada*: matriz [0] [0] obtendrá el primer elemento de la primera fila. Sin embargo, con las matrices NumPy, proporciona los argumentos de índice y corte para ambas dimensiones en un solo par de corchetes:

```
numpy_array[row_selection, column_selection]
```

Para matrices unidimensionales, esto se simplifica a numpy\_array [selección]. Cuando selecciona un solo elemento, obtendrá un escalar; de lo contrario, obtendrá una matriz unidimensional o bidimensional. Recuerde que la notación de corte usa un índice inicial (incluido) y un índice final (excluido) con dos puntos en el medio, como en inicio fin. Al omitir el índice de inicio y finalización, se queda con dos puntos, que por lo tanto representa todas las filas o todas las columnas en una matriz bidimensional. He visualizado algunos ejemplos en [Figura 4-3](#), pero es posible que también desee dar [Figura 4-1](#) otra mirada, ya que los índices y ejes están etiquetados allí. Recuerde, al cortar una columna o fila de una matriz bidimensional, terminará con una matriz unidimensional, no con una columna bidimensional o un vector de fila.

array1[2]	array2[0, 0]	array2[:, 1:]	array2[:, 1]	array2[1, :2]

Figura 4-3. Seleccionar elementos de una matriz NumPy

Juega con los ejemplos que se muestran en [Figura 4-3](#) ejecutando el siguiente código:

```
En [18]: array1[2] # Devuelve un escalar
```

```
Fuera [18]: 1000.0
```

```
En [19]: array2[0, 0] # Devuelve un escalar
```

```
Fuera [19]: 1.0
```

```
En [20]: array2[:, 1:] # Devuelve una matriz 2d
```

```
Fuera [20]: matriz ([[2., 3.],  
[5., 6.]])
```

```
En [21]: array2[:, 1] # Devuelve una matriz 1d
```

```
Fuera [21]: matriz ([2., 5.])
```

```
En [22]: array2[1,:2] # Devuelve una matriz 1d
```

```
Fuera [22]: matriz ([4., 5.])
```

Hasta ahora, he construido las matrices de muestra a mano, es decir, proporcionando números en una lista. Pero NumPy también ofrece algunas funciones útiles para construir matrices.

## Constructores de matrices útiles

NumPy ofrece algunas formas de construir matrices que también serán útiles para crear pandas DataFrames, como veremos en [Capítulo 5](#). Una forma de crear matrices con facilidad es utilizar el `arange` función. Esto significará *rango de matriz* y es similar al `incorporado distancia` que conocimos en el capítulo anterior, con la diferencia de que `arange` devuelve una matriz NumPy. Combinándolo con `remodelar` nos permite generar rápidamente una matriz con las dimensiones deseadas:

```
En [23]: notario público.arange(2 * 5).remodelar(2, 5) # 2 filas, 5 columnas
```

```
Fuera [23]: matriz ([[0, 1, 2, 3, 4],  
[5, 6, 7, 8, 9]])
```

Otra necesidad común, por ejemplo para las simulaciones de Monte Carlo, es generar matrices de números pseudoaleatorios normalmente distribuidos. NumPy lo hace fácil:

```
En [24]: notario público.aleatorio.randn(2, 3) # 2 filas, 3 columnas
```

```
Salida [24]: matriz ([[ -0.30047275, -1.19614685, -0.13652283],  
[ 1.05769357, 0.03347978, -1.2153504]])
```

Otros constructores útiles que vale la pena explorar son `np.ones` y `np.zeros` para crear matrices con unos y ceros, respectivamente, y `np. ojo` para crear una matriz de identidad. Nos encontraremos con algunos de estos constructores nuevamente en el próximo capítulo, pero por ahora, aprendamos sobre la diferencia entre una vista y una copia de una matriz NumPy.

## Ver vs. Copiar

Regreso de matrices NumPy *puntos de vista* cuando los cortas. Esto significa que está trabajando con un subconjunto de la matriz original sin copiar los datos. Por lo tanto, establecer un valor en una vista también cambiará la matriz original:

```
En [25]: array2
```

```
Fuera [25]: matriz ([[1., 2., 3.],  
[4., 5., 6.]])
```

```
En [26]: subconjunto = array2[:, :2]
```

subconjunto

```
Fuera [26]: matriz ([[1., 2.],  
[4., 5.]])
```

```
En [27]: subconjunto[0, 0] = 1000 En [
```

```
28]: subconjunto
```

```
Fuera [28]: matriz ([[1000., 2.],  
[4., 5.]])
```

```
En [29]: array2
```

```
Fuera [29]: matriz ([[1000.,      2.,      3.],
                   [4.,       5.,       6.]])
```

Si eso no es lo que quieras, tendrías que cambiar En [26] como sigue:

```
subconjunto = array2[:, :2].Copiar()
```

Trabajar en una copia dejará la matriz original sin cambios.

## Conclusión

En este capítulo, le mostré cómo trabajar con matrices NumPy y qué hay detrás de expresiones como la vectorización y la transmisión. Dejando a un lado estos términos técnicos, trabajar con matrices debería resultar bastante intuitivo dado que siguen muy de cerca la notación matemática. Si bien NumPy es una biblioteca increíblemente poderosa, existen dos problemas principales cuando desea usarla para el análisis de datos:

- Toda la matriz NumPy debe ser del mismo tipo de datos. Esto, por ejemplo, significa que no puede realizar ninguna de las operaciones aritméticas que hicimos en este capítulo cuando su matriz contiene una combinación de texto y números. Tan pronto como el texto esté involucrado, la matriz tendrá el tipo de datos objeto, que no permitirá operaciones matemáticas.
- El uso de matrices NumPy para el análisis de datos dificulta saber a qué se refiere cada columna o fila porque normalmente selecciona columnas a través de su posición, como en matriz2[:, 1].

pandas ha resuelto estos problemas al proporcionar estructuras de datos más inteligentes sobre las matrices NumPy. Qué son y cómo funcionan es el tema del próximo capítulo.

## Análisis de datos con pandas

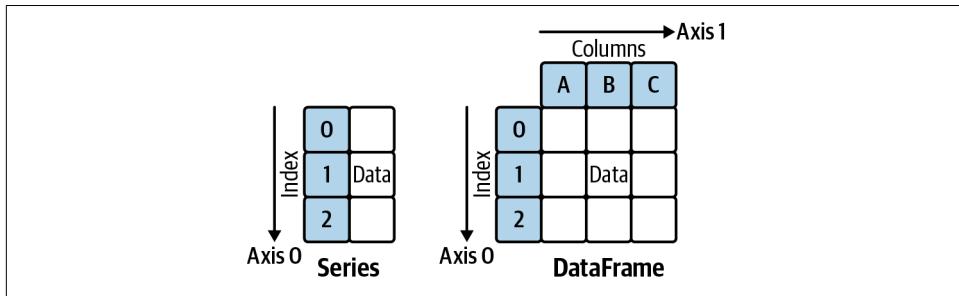
Este capítulo le presentará a los pandas, los *Biblioteca de análisis de datos de Python o, como me gusta decirlo, la hoja de cálculo basada en Python con superpoderes*. Es tan poderoso que algunas de las empresas con las que trabajé han logrado deshacerse de Excel por completo reemplazándolo con una combinación de cuadernos de Jupyter y pandas. Sin embargo, como lector de este libro, supongo que mantendrá Excel, en cuyo caso los pandas servirán como una interfaz para obtener datos dentro y fuera de las hojas de cálculo. pandas hace que las tareas que son particularmente dolorosas en Excel sean más fáciles, rápidas y menos propensas a errores. Algunas de estas tareas incluyen obtener grandes conjuntos de datos de fuentes externas y trabajar con estadísticas, series de tiempo y gráficos interactivos. Los superpoderes más importantes de los pandas son la vectorización y la alineación de datos. Como ya hemos visto en el capítulo anterior con matrices NumPy, la vectorización le permite escribir conciso,

Este capítulo cubre todo el viaje del análisis de datos: comienza con la limpieza y preparación de los datos antes de mostrarle cómo encontrar sentido a conjuntos de datos más grandes a través de la agregación, las estadísticas descriptivas y la visualización. Al final del capítulo, veremos cómo podemos importar y exportar datos con pandas. Pero lo primero es lo primero: comenzemos con una introducción a las estructuras de datos principales de los pandas: DataFrame y Series.

### DataFrame y Series

DataFrame y Series son las estructuras de datos centrales en pandas. En esta sección, los presento con un enfoque en los componentes principales de un DataFrame: índice, columnas y datos. *Marco de datos* es similar a una matriz NumPy bidimensional, pero viene con etiquetas de columna y fila y cada columna puede contener diferentes tipos de datos. Al extraer una sola columna o fila de un DataFrame, obtiene una Serie unidimensional. De nuevo, un *Serie* es similar a una matriz NumPy unidimensional con etiquetas. Cuando

miras la estructura de un DataFrame en [Figura 5-1](#), no se necesitará mucha imaginación para ver que DataFrames serán sus hojas de cálculo basadas en Python.



*Figura 5-1. Una serie pandas y DataFrame*

Para mostrarle lo fácil que es la transición de una hoja de cálculo a un DataFrame, considere la siguiente tabla de Excel en [Figura 5-2](#), que muestra a los participantes de un curso en línea con su puntuación. Encontrarás el archivo correspondiente *course\_participants.xlsx* en el SGcarpeta del repositorio complementario.

	A	B	C	D	E	F
1	user_id	name	age	country	score	continent
2	1001	Mark		55 Italy	4.5	Europe
3	1000	John		33 USA	6.7	America
4	1002	Tim		41 USA	3.9	America
5	1003	Jenny		12 Germany	9	Europe

*Figura 5-2. course\_participants.xlsx*

Para que esta tabla de Excel esté disponible en Python, comience importando pandas, luego use su `read_excel` función, que devuelve un DataFrame:

En [1]: [importar pandas como pd](#)

En [2]: [pd.read\\_excel\("xl / course\\_participants.xlsx"\)](#) Fuerza

```
[2]:      user_id    nombre edad país puntuación continente
        0     1001   marca   55  Italia     4.5  Europa
        1     1000   John    33  Estados Unidos  6,7  America
        2     1002   Tim     41  Estados Unidos  3.9  America
        3     1003  Jenny    12  Alemania     9.0  Europa
```



#### La función `read_excel` con Python 3.9

Si estás corriendo `pd.read_excel` con Python 3.9 o superior, asegúrese de usar al menos pandas 1.2 o obtendrá un error al leer `xlsx` archivos.

Si ejecuta esto en un cuaderno de Jupyter, el DataFrame tendrá un formato agradable como una tabla HTML, lo que lo hace aún más parecido al aspecto de la tabla en Excel. Gastaré la totalidad de Capítulo 7 sobre leer y escribir archivos de Excel con pandas, por lo que este fue solo un ejemplo introductorio para mostrarle que las hojas de cálculo y los DataFrames son, de hecho, muy similares. Ahora vamos a volver a crear este DataFrame desde cero sin leerlo del archivo de Excel: una forma de crear un DataFrame es proporcionar los datos como una lista anidada, junto con los valores paracolumnas y índice:

```
En [3]: datos=[["Marcos", 55, "Italia", 4.5, "Europa"],  
           ["John", 33, "ESTADOS UNIDOS", 6.7, "America"], [  
           "Tim", 41, "ESTADOS UNIDOS", 3.9, "America"], [  
           "Jenny", 12, "Alemania", 9.0, "Europa"]]  
df = pd.Marco de datos(datos=datos,  
                      columnas=["nombre", "la edad", "país",  
                                "puntaje", "continente"],  
                      índice=[1001, 1000, 1002, 1003])  
df
```

Fuera [3]:

	nombre	edad	país	puntuación	continente
1001	marca	55	Italia	4.5	Europa
1000	John	33	Estados Unidos	6.7	America
1002	Tim	41	Estados Unidos	3.9	America
1003	Jenny	12	Alemania	9.0	Europa

Llamando al info método, obtendrá información básica, lo más importante, el número de puntos de datos y los tipos de datos para cada columna:

```
En [4]: df.info()  
  
<class 'pandas.core.frame.DataFrame'>  
Int64Index: 4 entradas, 1001 a 1003  
columnas de datos (5 columnas en total):  
 # Columna      Recuento no nulo   Dtype  
-----  
 0  nombre          4 no nulo     objeto  
 1  la edad         4 no nulo     int64  
 2  país            4 no nulo     objeto  
 3  puntaje         4 no nulo     float64  
 4  continente      4 no nulo     objeto  
dtypes: float64 (1), int64 (1), object (3) uso de  
memoria: 192.0+ bytes
```

Si solo está interesado en el tipo de datos de sus columnas, ejecute df.dtypes en lugar de. Las columnas con cadenas o tipos de datos mixtos tendrán el tipo de datos objeto.<sup>1</sup> Veamos ahora más de cerca el índice y las columnas de un DataFrame.

---

<sup>1</sup> pandas 1.0.0 introdujo un dedicado cuadro tipo de datos para hacer algunas operaciones más fáciles y consistentes con texto. Como todavía es experimental, no voy a utilizarlo en este libro.

## Índice

Las etiquetas de fila de un DataFrame se llaman *índice*. Si no tiene un índice significativo, déjelo a un lado al construir un DataFrame. Entonces, los pandas crearán automáticamente un índice entero a partir de cero. Vimos esto en el primer ejemplo cuando leímos el DataFrame del archivo de Excel. Un índice permitirá a los pandas buscar datos más rápido y es esencial para muchas operaciones comunes, por ejemplo, combinar dos DataFrames. Accede al objeto de índice de la siguiente manera:

En [5]: df.índice

Fuera [5]: Int64Index ([1001, 1000, 1002, 1003], dtype = 'int64')

Si tiene sentido, asigne un nombre al índice. Sigamos la tabla en Excel y le damos la nombre user\_id:

En [6]: df.índice.nombre = "user\_id"  
df

Fuera [6]: nombre edad país puntuación continente

user_id		nombre	edad	país	puntuación	continente
1001	Marcos	55	Italia	4.5	Europa	
1000	John	33	Estados Unidos	6,7	America	
1002	Tim	41	Estados Unidos	3.9	America	
1003	Jenny	12	Alemania	9.0	Europa	

A diferencia de la clave principal de una base de datos, un índice de DataFrame puede tener duplicados, pero la búsqueda de valores puede ser más lenta en ese caso. Para convertir un índice en una columna normal, use reset\_index, y para establecer un nuevo uso de índice set\_index. Si no desea perder su índice existente al configurar uno nuevo, asegúrese de restablecerlo primero:

En [7]: # "reset\_index" convierte el índice en una columna, reemplazando el  
# index con el índice predeterminado. Esto corresponde al DataFrame  
# desde el principio que cargamos desde Excel.df.  
reset\_index()

Fuera [7]: user\_id nombre edad país puntuación continente

0	1001	marca	55	Italia	4.5	Europa
1	1000	John	33	Estados Unidos	6,7	America
2	1002	Tim	41	Estados Unidos	3.9	America
3	1003	Jenny	12	Alemania	9.0	Europa

En [8]: # "reset\_index" convierte "user\_id" en una columna normal y  
# "set\_index" convierte la columna "nombre" en el índice df.  
reset\_index().set\_index("nombre")

Fuera [8]: user\_id edad país puntuación continente

nombre	user_id	edad	país	puntuación	continente
Marcos	1001	55	Italia	4.5	Europa
John	1000	33	Estados Unidos	6,7	America
Tim	1002	41	Estados Unidos	3.9	America
Jenny	1003	12	Alemania	9.0	Europa

Haciendo df.reset\_index().set\_index("nombre"), Tu estas usando *encadenamiento de métodos*: ya que reset\_index() devuelve un DataFrame, puede llamar directamente a otro método DataFrame sin tener que escribir primero el resultado intermedio.



#### Los métodos DataFrame devuelven copias

Siempre que llame a un método en un DataFrame en el formulario df.method\_name(), obtendrá una copia del DataFrame con ese método aplicado, dejando intacto el DataFrame original. Lo acabamos de hacer llamando df.reset\_index(). Si quisiera cambiar el DataFrame original, tendría que asignar el valor de retorno a la variable original de la siguiente manera:

```
df = df.reset_index()
```

Como no estamos haciendo esto, significa que nuestra variable df todavía conserva sus datos originales. Los siguientes ejemplos también llaman a métodos DataFrame, es decir, no cambien el DataFrame original.

Para cambiar el índice, use el reindexar método:

En [9]: df.reindex([999, 1000, 1001, 1004])

Fuera [9]:

	nombre	edad	país	puntaje	continente
user_id					
999	Yaya	Yaya	Yaya	Yaya	Yaya
1000	John	33,0		6,7	America
1001	Marcos	55,0	Italia	4,5	Europa
1004	Yaya	Yaya	Yaya	Yaya	Yaya

Este es un primer ejemplo de alineación de datos en funcionamiento: reindexar asumirá todas las filas que coincidan con el nuevo índice e introducirá filas con valores faltantes (Yaya) donde no existe información. Los elementos de índice que deje fuera se eliminarán. Voy a presentar Yaya correctamente un poco más adelante en este capítulo. Finalmente, para ordenar un índice, use el sort\_index método:

En [10]: df.sort\_index()

Fuera [10]:

	nombre	edad	país	puntuación	continente
user_id					
1000	John	33		6.7	América
1001	Marcos	55	Italia	4.5	Europa
1002	Tim	41		3.9	America
1003	Jenny	12	Alemania	9.0	Europa

Si, en cambio, desea ordenar las filas por una o más columnas, utilice sort\_values:

En [11]: df.sort\_values(["continente", "la edad"])

Fuera [11]:

	nombre	edad	país	puntuación	continente
user_id					
1000	John	33		6.7	América

```

1002      Tim  41      Estados Unidos 3.9    America
1003    Jenny  12      Alemania     9.0    Europa
1001    Marcos  55      Italia      4.5    Europa

```

El ejemplo muestra cómo ordenar primero por continente, entonces por la edad. Si desea ordenar solo por una columna, también puede proporcionar el nombre de la columna como una cadena:

```
df.sort_values ("continente")
```

Esto ha cubierto los conceptos básicos de cómo funcionan los índices. ¡Ahora dirijamos nuestra atención a su equivalente horizontal, las columnas DataFrame!

## Columnas

Para obtener información sobre las columnas de un DataFrame, ejecute el siguiente código:

```
En [12]: df.columns
```

```
Fuera [12]: Índice (['nombre', 'edad', 'país', 'puntuación', 'continente'], dtype = 'objeto')
```

Si no proporciona ningún nombre de columna al construir un DataFrame, los pandas numerarán las columnas con números enteros que comienzan en cero. Sin embargo, con las columnas, esto casi nunca es una buena idea, ya que las columnas representan variables y, por lo tanto, son fáciles de nombrar. Asigna un nombre a los encabezados de las columnas de la misma manera que lo hicimos con el índice:

```
En [13]: df.columnas.nombre = "propiedades"
df
```

```
Fuera [13]: propiedades      nombre edad país puntuación continente
           user_id
1001      Marcos  55  Italia      4.5 Europa
1000      John   33  Estados Unidos 6.7 America
1002      Tim    41  Estados Unidos 3.9 America
1003    Jenny  12  Alemania     9.0 Europa
```

Si no le gustan los nombres de las columnas, cámbielos el nombre:

```
En [14]: df.rebautizar(columnas={"nombre": "Primer nombre", "la edad": "La edad"})
```

```
Fuera [14]: propiedades Nombre Edad país puntuación continente
           user_id
1001      Marcos  55  Italia      4.5 Europa
1000      John   33  Estados Unidos 6.7 America
1002      Tim    41  Estados Unidos 3.9 America
1003    Jenny  12  Alemania     9.0 Europa
```

Si desea eliminar columnas, utilice la siguiente sintaxis (el ejemplo le muestra cómo eliminar columnas e índices al mismo tiempo):

```
En [15]: df.soltar(columnas=["nombre", "país"],
                    índice=[1000, 1003])
```

```
Fuera [15]: propiedades
   user_id    la edad  puntaje continente
      1001        55     4.5    Europa
      1002        41     3.9    America
```

Las columnas y el índice de un DataFrame están ambos representados por un Índice objeto, para que pueda cambiar sus columnas en filas y viceversa transponiendo su DataFrame:

En [dieciséis]: `df.T # Atajo para df.transpose ()`

```
Fuera [16]: user_id      1001      1000      1002      1003
   propiedades
      nombre      Marcos      John      Tim      Jenny
      la edad       55          33        41        12
      país         Italia      Estados Unidos      Estados Unidos      Alemania
      puntaje      4.5          6,7       3.9          9
      continente    Europa      America      America      Europa
```

Vale la pena recordar aquí que nuestro DataFrame df todavía no ha cambiado, ya que nunca hemos reasignado el DataFrame que regresa de las llamadas al método al original df variable. Si desea reordenar las columnas de un DataFrame, puede usar el reindexar método que usamos con el índice, pero seleccionar las columnas en el orden deseado suele ser más intuitivo:

En [17]: `df.loc[:, ["continente", "país", "nombre", "la edad", "puntaje"]]`

```
Fuera [17]: propiedades continente      país      nombre      la edad      puntaje
   user_id
      1001        Europa      Italia      Marcos      55      4.5
      1000        America      Estados Unidos      John      33      6,7
      1002        America      Estados Unidos      Tim      41      3.9
      1003        Europa      Alemania      Jenny      12      9.0
```

Este último ejemplo necesita bastantes explicaciones: todo sobre loc y cómo funciona la selección de datos es el tema de la siguiente sección.

## Manipulación de datos

Los datos del mundo real apenas se sirven en bandeja de plata, por lo que antes de trabajar con ellos, debe limpiarlos y convertirlos en una forma digerible. Comenzaremos esta sección viendo cómo seleccionar datos de un DataFrame, cómo cambiarlos y cómo lidiar con los datos faltantes y duplicados. Luego, realizaremos algunos cálculos con DataFrames y veremos cómo trabaja con datos de texto. Para concluir esta sección, descubriremos cuándo los pandas devuelven una vista frente a una copia de los datos. Muchos conceptos de esta sección están relacionados con lo que ya hemos visto con las matrices NumPy en el último capítulo.

## Seleccionar datos

Comencemos accediendo a los datos por etiqueta y posición antes de mirar otros métodos, incluida la indexación booleana y la selección de datos mediante el uso de MultiIndex.

### Seleccionar por etiqueta

La forma más común de acceder a los datos de un DataFrame es haciendo referencia a sus etiquetas. Usa el atributo loc, lo que significa *localización*, para especificar qué filas y columnas desea recuperar:

```
df.loc[row_selection, column_selection]
```

loc admite la notación de corte y, por lo tanto, acepta dos puntos para seleccionar todas las filas o columnas, respectivamente. Además, puede proporcionar listas con etiquetas, así como un solo nombre de columna o fila. Mira esto [Tabla 5-1](#) para ver algunos ejemplos de cómo seleccionar diferentes partes de nuestro DataFrame de muestra df.

Tabla 5-1. Selección de datos por etiqueta

Selección	Tipo de datos devueltos	Ejemplo
Valor único	Escalar	df.loc[1000, "país"] df.loc["país"]
Una columna (1d)	Serie	[:, "país"]
Una columna (2d)	Marco de datos	df.loc[:, ["país"]]
Varias columnas	Marco de datos	df.loc[:, ["país", "edad"]]
Rango de columnas	Marco de datos	"nombre": "país"] df.loc[1000,:]
Una fila (1d)	Serie	
Una fila (2d)	Marco de datos	df.loc[[1000],:]
Varias filas	Marco de datos	df.loc[[1003, 1000],:]
Rango de filas	Marco de datos	df.loc[1000: 1002,:]



El corte de etiquetas tiene intervalos cerrados

El uso de la notación de corte con etiquetas es inconsistente con respecto a cómo funciona todo lo demás en Python y pandas: *incluir* el extremo superior.

Aplicando nuestro conocimiento de [Tabla 5-1](#), usemos loc para seleccionar escalares, Series y DataFrames:

```
En [18]: # El uso de escalares para la selección de filas y columnas devuelve un escalar  
df.loc[1001, "nombre"]
```

Fuera [18]: 'Mark'

```
En [19]: # El uso de un escalar en la selección de fila o columna devuelve una serie  
df.loc[[1001, 1002], "la edad"]
```

```
Fuera [19]: user_id
 1001    55
 1002    41
Nombre: edad, tipo: int64
```

En [20]: # Seleccionar varias filas y columnas devuelve un DataFrame  
`df.loc[:1002, ["nombre", "país"]]`

```
Fuera [20]: propiedades    nombre del país
      user_id
 1001      Marcos    Italia
 1000      John    Estados Unidos
 1002      Tim    Estados Unidos
```

Es importante que comprenda la diferencia entre un DataFrame con una o más columnas y una Serie: incluso con una sola columna, los DataFrames son bidimensionales, mientras que las Series son unidimensionales. Tanto DataFrame como Series tienen un índice, pero solo el DataFrame tiene encabezados de columna. Cuando selecciona una columna como Serie, el encabezado de la columna se convierte en el nombre de la Serie. Muchas funciones o métodos funcionarán tanto en Series como en DataFrame, pero cuando realiza cálculos aritméticos, el comportamiento es diferente: con DataFrames, pandas alinea los datos de acuerdo con los encabezados de las columnas; más sobre esto un poco más adelante en este capítulo.



#### Acceso directo para la selección de columnas

Dado que la selección de columnas es una operación tan común, pandas ofrece un atajo. En lugar de:

`df.loc[:, column_selection]`

puedes escribir:

`df[column_selection]`

Por ejemplo, `df[["país"]]` devuelve una serie de nuestro DataFrame de muestra y `df[["nombre", "país"]]` devuelve un DataFrame con dos columnas.

#### Seleccionar por posición

Seleccionar un subconjunto de un DataFrame por posición corresponde a lo que hicimos al principio de este capítulo con las matrices NumPy. Con DataFrames, sin embargo, debe usar el `iloc` atributo, que significa *ubicación entera*:

`df.iloc[row_selection, column_selection]`

Cuando usa rebanadas, se ocupa de los intervalos estándar semiabiertos. [Tabla 5-2](#) le ofrece los mismos casos que analizamos anteriormente en [Tabla 5-1](#).

Tabla 5-2. Selección de datos por posición

Selección	Tipo de datos devueltos	Ejemplo
Valor único	Escalar	df.iloc [1, 2]
Una columna (1d)	Serie	df.iloc[:, 2]
Una columna (2d)	Marco de datos	df.iloc[:, [2]]
Varias columnas	Marco de datos	df.iloc[:, [2, 1]]
Rango de columnas	Marco de datos	df.iloc[:, :3]
Una fila (1d)	Serie	df.iloc [1,:]
Una fila (2d)	Marco de datos	df.iloc [[1],:]
Varias filas	Marco de datos	df.iloc [[3, 1],:]
Rango de filas	Marco de datos	df.iloc [1: 3,:]

Así es como se usa iloc—de nuevo con las mismas muestras que usamos con loc antes de:

```
En [21]: df.iloc[0, 0] # Devuelve un escalar
Fuera
[21]: 'Mark'

En [22]: df.iloc[[0, 2], 1]           # Devuelve una serie
Fuera [22]: user_id
      1001    55
      1002    41
      Nombre: edad, tipo: int64

En [23]: df.iloc[:3, [0, 2]]          # Devuelve un DataFrame
Fuera [23]: propiedades    nombre del país
      user_id
      1001      Marcos    Italia
      1000      John     Estados Unidos
      1002      Tim      Estados Unidos
```

Seleccionar datos por etiqueta o posición no es el único medio para acceder a un subconjunto de su DataFrame. Otra forma importante es utilizar la indexación booleana; ¡Vamos a ver cómo funciona!

#### Seleccionar por indexación booleana

La indexación booleana se refiere a la selección de subconjuntos de un DataFrame con la ayuda de una Serie o un DataFrame cuyos datos consisten solo en Cierto o Falso. Las series booleanas se utilizan para seleccionar columnas y filas específicas de un DataFrame, mientras que los DataFrames booleanos se utilizan para seleccionar valores específicos en todo un DataFrame. Por lo general, utilizará la indexación booleana para filtrar las filas de un DataFrame. Piense en ello como la funcionalidad de Autofiltro en Excel. Por ejemplo, así es como filtra su DataFrame para que solo muestre a las personas que viven en los EE. UU. Y tienen más de 40 años:

```
En [24]: tf = (df["la edad"] > 40) Y (df["país"] == "ESTADOS UNIDOS")
tf # Esta es una serie con solo Verdadero / Falso
```

Fuera [24]: user\_id

```
1001    Falso
1000    Falso
1002    Cierto
1003    Falso
dtype: bool
```

```
En [25]: df.loc[tf,:]
```

```
Out [25]: nombre de la propiedad      edad país      puntaje continente
          user_id
          1002      Tim   41      Estados Unidos  3.9 América
```

Hay dos cosas que necesito explicar aquí. Primero, debido a limitaciones técnicas, no puede usar los operadores booleanos de Python desde Capítulo 3 con DataFrames. En su lugar, debe utilizar los símbolos como se muestra en Tabla 5-3.

Tabla 5-3. operadores booleanos

Tipos de datos básicos de Python	DataFrames y Series
y	Y
o	
no	~

En segundo lugar, si tiene más de una condición, asegúrese de poner cada expresión booleana entre paréntesis para que la precedencia de los operadores no se interponga en su camino: por ejemplo, & tiene una precedencia de operadores mayor que ==. Por tanto, sin paréntesis, la expresión de la muestra se interpretaría como:

```
df["la edad"] > (40 Y df["país"]) == "ESTADOS UNIDOS"
```

Si desea filtrar el índice, puede referirse a él como df.index:

```
En [26]: df.loc[df.índice > 1001,:]
```

```
Fuera [26]: propiedades      nombre edad país puntuación continente
           user_id
           1002      Tim   41      Estados Unidos  3.9      America
           1003      Jenny  12      Alemania     9.0      Europa
```

Para lo que usarías el en operador con estructuras de datos básicas de Python como listas, use es en con una serie. Así es como filtra su DataFrame a participantes de Italia y Alemania:

```
En [27]: df.loc[df["país"].es en(["Italia", "Alemania"]),:]
```

```
Fuera [27]: propiedades nombre edad país puntuación continente
           user_id
```

1001	Marcos	55	Italia	4.5	Europa
1003	Jenny	12	Alemania	9.0	Europa

Mientras usa loc para proporcionar una serie booleana, los DataFrames ofrecen una sintaxis especial sin loc para seleccionar valores dado el DataFrame completo de booleanos:

`df[boolean_df]`

Esto es especialmente útil si tiene DataFrames que constan solo de números. Proporcionar un DataFrame de booleanos devuelve el DataFrame con Yaya donde sea que esté el DataFrame booleano Falso. Una vez más, una discusión más detallada de Yaya seguirá en breve. Comencemos creando un nuevo DataFrame de muestra llamado lluvia que consta solo de números:

En [28]: # Esta podría ser la precipitación anual en milímetros.

```
Iluvia = pd.Marco de datos(datos={"Ciudad 1": [300,1, 100,2],
                                    "Ciudad 2": [400,3, 300,4],
                                    "Ciudad 3": [1000,5, 1100,6]})
```

Iluvia

Fuera [28]:	Ciudad 1	Ciudad 2	Ciudad 3
0	300,1	400,3	1000,5
1	100,2	300,4	1100,6

En [29]: Iluvia < 400 Fuera

[29]:	Ciudad 1	Ciudad 2	Ciudad 3
0	Cierto	Falso	Falso
1	Cierto	Cierto	Falso

En [30]: Iluvia[Iluvia < 400] Fuera [30]:

	Ciudad 1	Ciudad 2	Ciudad 3
0	300,1	Yaya	Yaya
1	100,2	300,4	Yaya

Tenga en cuenta que en este ejemplo, he usado un diccionario para construir un nuevo DataFrame. Esto a menudo es conveniente si los datos ya existen en ese formulario. Trabajar con valores booleanos de esta manera se usa más comúnmente para filtrar valores específicos como valores atípicos.

Para concluir la parte de selección de datos, presentaré un tipo especial de índice llamado MultiIndex.

#### Seleccionar usando un MultiIndex

A *MultiIndex* es un índice con más de un nivel. Le permite agrupar jerárquicamente sus datos y le brinda fácil acceso a subconjuntos. Por ejemplo, si establece el índice de nuestro DataFrame de muestradf a una combinación de continente y país, puede seleccionar fácilmente todas las filas con un continente determinado:

En [31]: # Un MultiIndex necesita ser ordenado

```
df_multi = df.reset_index().set_index(["continente", "país"])
```

```
df_multi = df_multi.sort_index()  
df_multi
```

Fuera [31]: propiedades  
país continente

		user_id	nombre	la edad	puntaje
America	Estados Unidos	1000	John	33	6,7
	Estados Unidos	1002	Tim	41	3.9
Europa	Alemania	1003	Jenny	12	9.0
	Italia	1001	Marcos	55	4.5

En [32]: df\_multi.loc["Europa",:]

Fuera [32]: propiedades  
país

	user_id	nombre	la edad	puntaje
Alemania	1003	Jenny	12	9.0
Italia	1001	Marcos	55	4.5

Tenga en cuenta que pandas embellece la salida de un MultiIndex al no repetir el nivel de índice más a la izquierda (los continentes) para cada fila. En cambio, solo imprime el continente cuando cambia. La selección de varios niveles de índice se realiza proporcionando una tupla:

En [33]: df\_multi.loc[("Europa", "Italia"),:]

Fuera [33]: propiedades  
país continente

	user_id	nombre	edad	puntuación
Europa Italia	Marca 1001		55	4.5

Si desea restablecer selectivamente parte de un MultiIndex, proporcione el nivel como argumento.

Zero es la primera columna de la izquierda:

En [34]: df\_multi.reset\_index(nivel=0)

Fuera [34]: propiedades continente  
país

	continente	user_id	nombre	la edad	puntaje
Estados Unidos	America	1000	John	33	6,7
Estados Unidos	America	1002	Tim	41	3.9
Alemania	Europa	1003	Jenny	12	9.0
Italia	Europa	1001	Marcos	55	4.5

Si bien no crearemos manualmente un MultiIndex en este libro, existen ciertas operaciones como agrupar por, lo que hará que los pandas devuelvan un DataFrame con un MultiIndex, por lo que es bueno saber qué es. Nos reuniremos agrupar por más adelante en este capítulo.

Ahora que conoces varias formas de *Seleccione* datos, es hora de aprender cómo *cambio* datos.

## Configuración de datos

La forma más fácil de cambiar los datos de un DataFrame es asignando valores a ciertos elementos usando el loc o iloc atributos. Este es el punto de partida de esta sección antes de pasar a otras formas de manipular DataFrames existentes: reemplazar valores y agregar nuevas columnas.

#### Configuración de datos por etiqueta o posición

Como se señaló anteriormente en este capítulo, cuando llama a métodos DataFrame como df.reset\_index(), el método siempre se aplicará a una copia, dejando intacto el DataFrame original. Sin embargo, la asignación de valores a través del loc y iloc atributos cambia el DataFrame original. Como quiero dejar nuestro DataFrame df intacto, estoy trabajando con una copia aquí que estoy llamando df2. Si desea cambiar un solo valor, haga lo siguiente:

En [35]: # Copie el DataFrame primero para dejar el original intacto  
df2 = df.Copiar()

En [36]: df2.loc[1000, "nombre"] = "JOHN"  
df2

Fuera [36]: propiedades nombre edad país puntuación continente

user_id					
1001	Marcos	55	Italia	4.5	Europa
1000	JOHN	33	Estados Unidos	6.7	America
1002	Tim	41	Estados Unidos	3.9	America
1003	Jenny	12	Alemania	9.0	Europa

También puede cambiar varios valores al mismo tiempo. Una forma de cambiar la puntuación de los usuarios con ID 1000 y 1001 es usar una lista:

En [37]: df2.loc[[1000, 1001], "puntaje"] = [3, 4]  
df2

Fuera [37]: propiedades nombre edad país puntuación continente

user_id					
1001	Marcos	55	Italia	4.0	Europa
1000	JOHN	33	Estados Unidos	3.0	America
1002	Tim	41	Estados Unidos	3.9	America
1003	Jenny	12	Alemania	9.0	Europa

Cambio de datos por posición a través de iloc funciona de la misma manera. Pasemos ahora para ver cómo cambia los datos usando la indexación booleana.

#### Configuración de datos mediante indexación booleana

La indexación booleana, que usamos para filtrar filas, también se puede usar para asignar valores en un DataFrame. Imagine que necesita anonimizar todos los nombres de personas menores de 20 años o de EE. UU.:

En [38]: tf = (df2["la edad"] < 20) | (df2["país"] == "ESTADOS UNIDOS")  
df2.loc[tf, "nombre"] = "xxx"  
df2

Fuera [38]: propiedades nombre edad país puntuación continente

user_id					
1001	Marcos	55	Italia	4.0	Europa
1000	XXX	33	Estados Unidos	3.0	América

```

1002      xxx  41   Estados Unidos  3.9  America
1003      xxx  12   Alemania       9.0  Europa

```

A veces, tiene un conjunto de datos en el que necesita reemplazar ciertos valores en todos los ámbitos, es decir, no específicos de ciertas columnas. En ese caso, haga uso de la sintaxis especial nuevamente y proporcione todo el DataFrame con valores booleanos como este (la muestra hace uso nuevamente de lluvia Marco de datos):

En [39]: # Copie el DataFrame primero para dejar el original intacto  
`lluvia2 = lluvia.Copiar()lluvia2`

Fuera [39]: Ciudad 1 Ciudad 2 Ciudad 3  
0 300,1 400,3 1000,5  
1 100,2 300,4 1100,6

En [40]: # Establezca los valores en 0 siempre que estén por debajo de 400  
`lluvia2[lluvia2 < 400] = 0lluvia2`

Fuera [40]: Ciudad 1 Ciudad 2 Ciudad 3  
0 0,0 400,3 1000,5  
1 0,0 0,0 1100,6

Si solo desea reemplazar un valor por otro, hay una manera más fácil de hacerlo, como le mostraré a continuación.

#### Configurar datos reemplazando valores

Si desea reemplazar un cierto valor en todo su DataFrame o columnas seleccionadas, use el reemplazar método:

En [41]: df2.reemplazar("ESTADOS UNIDOS", "NOSOTROS")

Fuera [41]: propiedades nombre edad país puntuación continente  
user\_id  
1001 Marcos 55 Italia 4.0 Europa  
1000 xxx 33 nosotros 3,0 America  
1002 xxx 41 nosotros 3.9 America  
1003 xxx 12 Alemania 9.0 Europa

Si, en cambio, solo quisiera actuar en el país columna, puede usar esta sintaxis en su lugar:

`df2.reemplazar({"país": {"ESTADOS UNIDOS": "NOSOTROS"}})`

En este caso, desde Estados Unidos solo aparece en el país columna, produce el mismo resultado que la muestra anterior. Para concluir esta sección, veamos cómo puede agregar columnas adicionales a un DataFrame.

#### Configurar datos agregando una nueva columna

Para agregar una nueva columna a un DataFrame, asigne valores a un nuevo nombre de columna. Por ejemplo, puede agregar una nueva columna a un DataFrame usando un escalar o una lista:

```
En [42]: df2.loc[:, "descuento"] = 0  
df2.loc[:, "precio"] = [49,9, 49,9, 99,9, 99,9]df2
```

Fuera [42]: nombre de la propiedad edad país puntuación continente precio de descuento

user_id	nombre	edad	país	puntuación	continente	precio	descuento
1001	Marcos	55	Italia	4.0	Europa	0	49,9
1000	xxx	33	Estados Unidos	3,0	America	0	49,9
1002	xxx	41	Estados Unidos	3.9	America	0	99,9
1003	xxx	12	Alemania	9.0	Europa	0	99,9

Agregar una nueva columna a menudo implica cálculos vectorizados:

```
En [43]: df2 = df.Copiar() # Comencemos con una copia nueva  
df2.loc[:, "Año de nacimiento"] = 2021 - df2["la edad"]df2
```

Fuera [43]: propiedades nombre edad país puntuación continente año de nacimiento

user_id	nombre	edad	país	puntuación	continente	año de nacimiento
1001	Marcos	55	Italia	4.5	Europa	1966
1000	John	33	Estados Unidos	6,7	America	1988
1002	Tim	41	Estados Unidos	3.9	America	1980
1003	Jenny	12	Alemania	9.0	Europa	2009

Te mostraré más sobre el cálculo con DataFrames en un momento, pero antes de llegar allí, ¿recuerdas que he usado Yaya algunas veces ya? La siguiente sección finalmente le dará más contexto sobre el tema de los datos faltantes.

#### Datos perdidos

La falta de datos puede ser un problema, ya que tiene el potencial de sesgar los resultados de su análisis de datos, lo que hace que sus conclusiones sean menos sólidas. Sin embargo, es muy común tener lagunas en sus conjuntos de datos con las que tendrá que lidiar. En Excel, generalmente tiene que lidiar con celdas vacías o #N / A errores, pero pandas usa NumPy np.nan para datos faltantes, mostrado como Yaya. Yaya es el estándar de coma flotante para *No un número*. Para marcas de tiempo, pd.NaT se usa en su lugar, y para el texto, pandas usa Ninguno. Utilizando Ninguno o np.nan, puede introducir valores perdidos:

```
En [44]: df2 = df.Copiar() # Comencemos con una copia nueva  
df2.loc[1000, "puntaje"] = Ninguno  
df2.loc[1003,:] = Ningunodf2
```

Fuera [44]: propiedades nombre edad país puntaje continente

user_id	nombre	edad	país	puntaje	continente
1001	Marcos	55,0	Italia	4.5	Europa
1000	John	33,0	Estados Unidos	Yaya	America

```

1002      Tim 41,0 Estados Unidos 3.9 America
1003 Ninguno Yaya Ninguno Yaya Ninguno

```

Para limpiar un DataFrame, a menudo desea eliminar filas con datos faltantes. Esto es tan simple como:

[En \[45\]: df2.dropna\(\)](#)

```

Fuera [45]: propiedades nombre edad país puntaje continente
user_id
1001 Marcos 55,0 Italia 4.5 Europa
1002 Tim 41,0 Estados Unidos 3.9 America

```

Sin embargo, si solo desea eliminar filas donde *todos* faltan valores, use el cómo parámetro:

[En \[46\]: df2.dropna\(cómo="todos"\)](#)

```

Fuera [46]: propiedades nombre edad país puntaje continente
user_id
1001 Marcos 55,0 Italia 4.5 Europa
1000 John 33,0 Estados Unidos Yaya America
1002 Tim 41,0 Estados Unidos 3.9 America

```

Para obtener un DataFrame booleano o una serie dependiendo de si hay Yaya o no, usa isna:

[En \[47\]: df2.isna\(\)](#)

```

Fuera [47]: propiedades nombre edad país puntaje continente
user_id
1001 Falso Falso Falso Falso Falso
1000 Falso Falso Falso Cierto Falso
1002 Falso Falso Falso Falso Falso
1003 Cierto Cierto Cierto Cierto Cierto

```

Para completar los valores faltantes, use fillna. Por ejemplo, para reemplazar Yaya en la columna de puntuación con su media (introduciré estadísticas descriptivas como significar dentro de poco):

[En \[48\]: df2.fillna\({"puntaje": df2\["puntaje"\].significar\(\)}\)](#)

```

Fuera [48]: propiedades nombre edad país puntaje continente
user_id
1001 Marcos 55,0 Italia 4.5 Europa
1000 John 33,0 Estados Unidos 4.2 America
1002 Tim 41,0 Estados Unidos 3.9 America
1003 Ninguno Yaya Ninguno 4.2 Ninguno

```

La falta de datos no es la única condición que nos obliga a limpiar nuestro conjunto de datos. Lo mismo ocurre con los datos duplicados, ¡así que veamos cuáles son nuestras opciones!

## Datos duplicados

Al igual que los datos faltantes, los duplicados impactan negativamente en la confiabilidad de su análisis. Para deshacerse de filas duplicadas, use el drop\_duplicates método. Opcionalmente, puede proporcionar un subconjunto de columnas como argumento:

En [49]: df.drop\_duplicates(["país", "continente"])

Fuera [49]: propiedades nombre edad país puntuación continente  
user\_id  
1001 Marcos 55 Italia 4.5 Europa  
1000 John 33 Estados Unidos 6.7 America  
1003 Jenny 12 Alemania 9.0 Europa

De forma predeterminada, esto dejará la primera aparición. Para saber si una determinada columna contiene duplicados o para obtener sus valores únicos, use los siguientes dos comandos (use df.index en lugar de df["país"] si desea ejecutar esto en el índice en su lugar):

En [50]: df["país"].nunique()

Fuera [50]: Falso

En [51]: df["país"].nunique()

Fuera [51]: matriz ([['Italia', 'EE. UU.', 'Alemania']], dtype = objeto)

Y finalmente, para comprender qué filas están duplicadas, use la duplicated método, que devuelve una serie booleana: de forma predeterminada, utiliza el parámetro mantener = "primero", que mantiene la primera aparición y marca solo duplicados con Verdadero. Configurando el parámetro mantener = Falso, volverá Cierto para todas las filas, incluida su primera aparición, lo que facilita la obtención de un DataFrame con todas las filas duplicadas. En el siguiente ejemplo, observamos el país columna para duplicados, pero en realidad, a menudo mira el índice o filas enteras. En este caso, tendrías que usar df.index.duplicated () o df.duplicated () en lugar de:

En [52]: # De forma predeterminada, solo marca los duplicados como Verdadero, es decir  
# sin la primera aparición df["país"]  
.duplicated()

Fuera [52]: user\_id  
1001 Falso  
1000 Falso  
1002 Cierto  
1003 Falso  
Nombre: país, tipo: bool

En [53]: # Para obtener todas las filas donde "país" está duplicado, use  
# keep = False  
df.loc[df["país"].duplicated(guardar=False),:]

Fuera [53]: propiedades nombre edad país puntuación continente  
user\_id

```
1000      John   33      Estados Unidos  6,7    America
1002      Tim    41      Estados Unidos  3,9    America
```

Una vez que haya limpiado sus DataFrames eliminando los datos faltantes y duplicados, es posible que desee realizar algunas operaciones aritméticas; la siguiente sección le brinda una introducción a cómo funciona esto.

## Operaciones aritméticas

Al igual que las matrices NumPy, DataFrames y Series utilizan la vectorización. Por ejemplo, para agregar un número a cada valor en el DataFrame de lluvia, simplemente haga lo siguiente:

En [54]: lluviaFuera

```
[54]:      Ciudad 1  Ciudad 2  Ciudad 3
          0    300,1    400,3  1000,5
          1    100,2    300,4  1100,6
```

En [55]: lluvia + 100Fuera

```
[55]:      Ciudad 1  Ciudad 2  Ciudad 3
          0    400,1    500,3  1100,5
          1    200,2    400,4  1200,6
```

Sin embargo, el verdadero poder de los pandas es su automático. *alineación de datos* mecanismo: cuando usa operadores aritméticos con más de un DataFrame, los pandas los alinean automáticamente por sus columnas e índices de fila. Creemos un segundo DataFrame con algunas de las mismas etiquetas de fila y columna. Luego construimos la suma:

```
En [56]: more_rainfall = pd.DataFrame(datos=[[100, 200], [300, 400]],
                                         índice=[1, 2],
                                         columnas=["Ciudad 1", "Ciudad 4"])
more_rainfall
```

```
Fuera [56]:      Ciudad 1  Ciudad 4
              1        100     200
              2        300     400
```

En [57]: lluvia + more\_rainfall

```
Fuera [57]: Ciudad 1 Ciudad 2 Ciudad 3      Ciudad 4
              0      NaN  NaN  NaN      Yaya
              1    200,2  NaN  NaN      Yaya
              2      NaN  NaN  NaN      Yaya
```

El índice y las columnas del DataFrame resultante son la unión de los índices y las columnas de los dos DataFrames: los campos que tienen un valor en ambos DataFrames muestran la suma, mientras que el resto del DataFrame muestra Yaya. Esto puede ser algo a lo que tenga que acostumbrarse si viene de Excel, donde las celdas vacías se convierten automáticamente en ceros cuando las usa en operaciones aritméticas. Para obtener el mismo comportamiento que en Excel, use el método `fill_value` para reemplazar Yaya valores con ceros:

En [58]: lluvia.agregar(more\_rainfall, fill\_value=0)

```
Fuera [58]:    Ciudad 1    Ciudad 2    Ciudad 3    Ciudad 4
              0   300,1     400,3   1000,5      Yaya
              1   200,2     300,4  1100,6     200,0
              2   300,0      Yaya     Yaya   400,0
```

Esto funciona en consecuencia para los otros operadores aritméticos como se muestra en [Tabla 5-4](#).

*Tabla 5-4. Operadores aritméticos*

Operador	Método
*	mul
+	agregar
-	sub
/	div
**	pow

Cuando tiene un DataFrame y una Serie en su cálculo, de manera predeterminada, la Serie se transmite a lo largo del índice:

En [59]: # Una serie tomada de una fila  
`Iluvia.loc[1,:]`

```
Fuera [59]: Ciudad 1      100,2
            Ciudad 2     300,4
            Ciudad 3    1100,6
Nombre: 1, dtype: float64
```

En [60]: Iluvia + Iluvia.loc[1,:] Fuera [60]:

```
          Ciudad 1    Ciudad 2    Ciudad 3
0    400,3     700,7   2101,1
1    200,4     600,8   2201,2
```

Por lo tanto, para agregar una serie en columnas, debe usar el agregar método con un explícito eje argumento:

En [61]: # Una serie tomada de una columna  
`Iluvia.loc[:, "Ciudad 2"]`

```
Fuera [61]: 0      400,3
            1      300,4
Nombre: Ciudad 2, dtype: float64
```

En [62]: Iluvia.agregar(Iluvia.loc[:, "Ciudad 2"], eje=0) Fuera [62]:

```
          Ciudad 1    Ciudad 2    Ciudad 3
0    700,4     800,6   1400,8
1    400,6     600,8   1401,0
```

Si bien esta sección trata sobre DataFrames con números y cómo se comportan en operaciones aritméticas, la siguiente sección muestra sus opciones cuando se trata de manipular texto en DataFrames.

## Trabajar con columnas de texto

Como hemos visto al principio de este capítulo, las columnas con texto o tipos de datos mixtos tienen el tipo de datos objeto. Para realizar operaciones en columnas con cadenas de texto, utilice el str atributo que le da acceso a los métodos de cadena de Python. Ya hemos conocido algunos métodos de cadena en [Capítulo 3](#), pero no estará de más echar un vistazo a los métodos disponibles en el [Documentos de Python](#). Por ejemplo, para eliminar los espacios en blanco iniciales y finales, utilice la banda método; para poner todas las primeras letras en mayúscula, existe la capitalizar método. Encadenarlos juntos limpiará las columnas de texto desordenadas que a menudo son el resultado de la entrada manual de datos:

En [63]: # Creamos un nuevo DataFrame

```
usuarios = pd.Marco de datos(datos=[" Marcos ", "JOHN      ", "Tim", "Jenny"],  
                           columns=["nombre"])
```

usuarios

Fuera [63]:

	nombre
0	Marcos
1	JOHN
2	Tim
3	Jenny

En [64]: users\_cleaned = usuarios.loc[:, "nombre"].str.banda().str.capitalizar()  
users\_cleaned

Fuera [64]:

	nombre
0	Marcos
1	John
2	Tim
3	Jenny

Nombre: nombre, dtype: objeto

O, para buscar todos los nombres que comienzan con una "J":

En [sesenta y cinco]: users\_cleaned.str.comienza con("J")

Fuera [65]:

0	Falso
1	Cierto
2	Falso
3	Cierto

Nombre: nombre, dtype: bool

Los métodos de cadena son fáciles de usar, pero a veces es posible que deba manipular un DataFrame de una manera que no está incorporada. En ese caso, cree su propia función y aplíquela a su DataFrame, como muestra la siguiente sección.

## Aplicar una función

Los DataFrames ofrecen aplicar mapa método, que aplicará una función a cada elemento individual, algo que es útil si no hay ufuncs NumPy disponibles. Por ejemplo, no hay ufuncs para el formato de cadenas, por lo que podemos formatear cada elemento de un DataFrame así:

En [66]: lluviaFuera

```
[66]:      Ciudad 1  Ciudad 2  Ciudad 3  
0    300,1    400,3  1000,5  
1    100,2    300,4  1100,6
```

En [67]: `def format_string(X):  
 regreso F"{{x:., 2f}}"`

En [68]: # Tenga en cuenta que pasamos la función sin llamarla,  
# es decir, format\_string y no format\_string ()!lluvia.  
aplicar mapa(format\_string)

```
Fuera [68]:  Ciudad 1 Ciudad 2 Ciudad 3  
0300,10 400,30 1000,50  
1100,20 300,40 1,100,60
```

Para desglosar esto: la siguiente cadena f devuelve X como una cadena: f "{{x}}". Para agregar formato, agregue dos puntos a la variable seguidos de la cadena de formato,.2f. La coma es el separador de miles y.2f medio *notación de coma fija con dos dígitos después de la coma decimal*. Para obtener más detalles sobre cómo formatear cadenas, consulte la **Especificación de formato Mini-idioma**, que forma parte de la documentación de Python.

Para este tipo de caso de uso, *expresiones lambda* (ver barra lateral) son ampliamente utilizados ya que le permiten escribir lo mismo en una sola línea sin tener que definir una función separada. Con expresiones lambda, podemos reescribir el ejemplo anterior como sigue:

En [69]: lluvia.aplicar mapa(lambda X: F"{{x:., 2f}}")

```
Fuera [69]: Ciudad 1 Ciudad 2 Ciudad 3  
0300,10 400,30 1.000,50  
1100,20 300,40 1.100,60
```

## Expresiones lambda

Python le permite definir una función en una sola línea a través de *expresiones lambda*. Las expresiones lambda son funciones anónimas, lo que significa que es una función sin nombre. Considere esta función:

```
def nombre de la función(arg1, arg2, ...):  
    regreso return_value
```

Esta función se puede reescribir como una expresión lambda como esta:

```
lambda arg1, arg2, ...: return_value
```

En esencia, reemplazas def con lambda, dejar de lado el regreso palabra clave y el nombre de la función, y poner todo en una línea. Como vimos con el aplicar mapa , esto puede ser realmente conveniente en este caso, ya que no necesitamos definir una función para algo que solo se está usando una sola vez.

Ahora he mencionado todos los métodos importantes de manipulación de datos, pero antes de continuar, es importante comprender cuándo los pandas usan una vista de un DataFrame y cuándo usan una copia.

## Ver vs. Copiar

Puede recordar del capítulo anterior que cortar matrices NumPy devuelve una vista. Con DataFrames, desafortunadamente es más complicado: no siempre es fácil predecir si loc y iloc devolver vistas o copias, lo que lo convierte en uno de los temas más confusos. Dado que es una gran diferencia si está cambiando la vista o una copia de un DataFrame, pandas genera la siguiente advertencia regularmente cuando cree que está configurando los datos de una manera no deseada: SettingWithCopyWarning. Para eludir esta advertencia bastante críptica, aquí hay algunos consejos:

- Establecer valores en el DataFrame original, no en un DataFrame que ha sido cortado de otro DataFrame
- Si desea tener un DataFrame independiente después de cortar, haga una copia explícita:

```
selección = df.loc[:, ["país", "continente"]].Copiar()
```

Si bien las cosas se complican con loc y iloc, vale la pena recordar que *todos* Métodos de DataFrame como df.dropna () o df.sort\_values ("nombre\_columna") *siempre* devolver una copia.

Hasta ahora, hemos trabajado principalmente con un DataFrame a la vez. La siguiente sección le muestra varias formas de combinar múltiples DataFrames en uno, una tarea muy común para la cual pandas ofrece herramientas poderosas.

## Combinando DataFrames

Combinar diferentes conjuntos de datos en Excel puede ser una tarea engorrosa y, por lo general, implica una gran cantidad de BUSCARV fórmulas. Afortunadamente, la combinación de DataFrames es una de las características principales de Pandas, donde sus capacidades de alineación de datos harán su vida realmente fácil, reduciendo así en gran medida la posibilidad de introducir errores. La combinación y fusión de DataFrames se puede realizar de varias formas; esta sección analiza solo los casos más comunes usando concat, unirse, y unir. Si bien se superponen, cada función hace que una tarea específica sea muy simple. Voy a empezar con el concat función, luego explique las diferentes opciones con entrar, y concluya presentando unir, la función más genérica de las tres.

## Concatenar

Para simplemente pegar varios DataFrames juntos, el concat La función es tu mejor amiga. Como puede ver por el nombre de la función, este proceso tiene el nombre técnico *con-catenación*. Por defecto, concat pega DataFrames a lo largo de las filas y alinea las columnas automáticamente. En el siguiente ejemplo, creo otro DataFrame, more\_users, y adjuntarlo a la parte inferior de nuestro DataFrame de muestra df:

```
En [70]: datos=[[15, "Francia", 4.1, "Becky"],
           [44, "Canadá", 6.1, "Leanne"]]
more_users = pd.Marco de datos(datos=datos,
                               columnas=["la edad", "país", "puntaje", "nombre"],
                               índice=[1000, 1011])
more_users
Fuera [70]:      edad país     puntaje     nombre
1000    15 Francia      4.1     Becky
1011    44 Canadá       6.1     Leanne
```

```
En [71]: pd.concat([df, more_users], eje=0)
```

```
Fuera [71]: nombre edad país puntuación continente
1001   Marcos    55 Italia    4.5 Europa
1000   John      33 Estados Unidos 6.7 America
1002   Tim        41 Estados Unidos 3.9 America
1003   Jenny     12 Alemania    9.0 Europa
1000   Becky     15 Francia      4.1 Yaya
1011 Leanne     44 Canadá       6.1 Yaya
```

Tenga en cuenta que ahora tiene elementos de índice duplicados, como concat pega los datos en el eje indicado (filas) y solo alinea los datos en el otro (columnas), haciendo coincidir los nombres de las columnas automáticamente, ¡incluso si no están en el mismo orden en los dos DataFrames! Si desea pegar dos DataFrames juntos a lo largo de las columnas, establezca eje = 1:

```
En [72]: datos=[[3, 4],
               [5, 6]]
more_categories = pd.Marco de datos(datos=datos,
                                      columnas=["cuestionarios", "inicios de
                                                    sesión"], índice=[1000, 2000])
more_categories
Fuera [72]:      cuestionarios     inicios de sesión
1000            3                  4
2000            5                  6
```

```
En [73]: pd.concat([df, more_categories], eje=1) Fuera
```

```
[73]:      nombre     la edad     país     puntaje continente  cuestionarios     inicios de sesión
1000  John    33,0      Estados Unidos 6.7 América      3,0      4.0
1001 Marcos   55,0      Italia 4,5 Europa      Yaya      Yaya
1002  Tim     41,0      Estados Unidos 3,9 América      Yaya      Yaya
```

```
1003 Jenny 12,0 Alemania  
2000 NaN NaN NaN
```

```
9.0     Europa     Yaya     Yaya  
Yaya     Yaya     5,0      6.0
```

La característica especial y muy útil de concat es que acepta más de dos marcos de datos. Usaremos esto en el próximo capítulo para hacer un solo DataFrame a partir de múltiples archivos CSV:

```
pd.concat([df1, df2, df3, ...])
```

Por otra parte, entrar y unir solo funciona con dos DataFrames, como veremos a continuación.

## Unirse y fusionarse

Cuando usted *entrar* dos DataFrames, combina las columnas de cada DataFrame en un nuevo DataFrame mientras decide qué sucede con las filas basándose en la teoría de conjuntos. Si ha trabajado con bases de datos relacionales antes, es el mismo concepto que el ENTRAR cláusula en consultas SQL. [Figura 5-3](#) muestra cómo funcionan los cuatro tipos de combinación (es decir, la combinación interna, izquierda, derecha y externa) mediante el uso de dos DataFrames de muestra, df1 y df2.

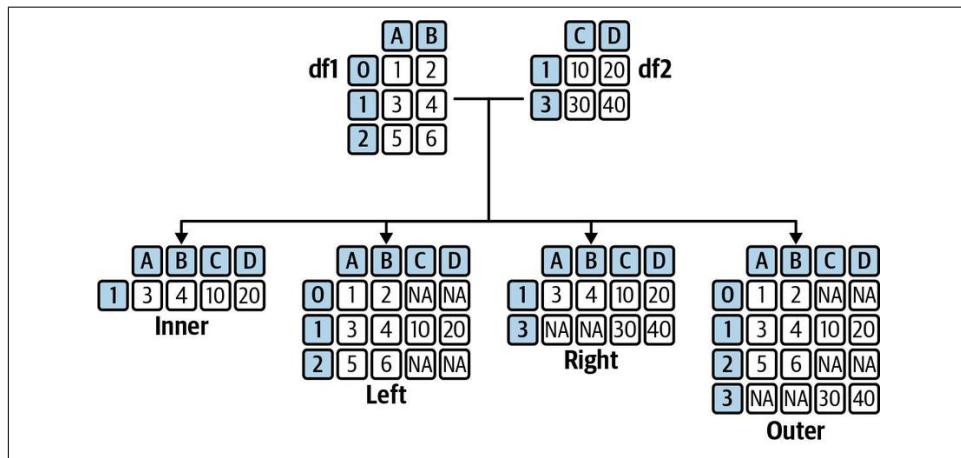


Figura 5-3. Tipos de unión

Con entrar, pandas usa los índices de ambos DataFrames para alinear las filas. Un *unir internamente* devuelve un DataFrame con solo aquellas filas donde los índices se superponen. Aunirse a la izquierda toma todas las filas del DataFrame izquierdo df1 y coincide con las filas del DataFrame correcto df2 en el índice. Dónde df2 no tiene una fila coincidente, los pandas completarán Yaya. La unión de la izquierda corresponde a la BUSCARV caso en Excel. Los *unirse a la derecha* toma todas las filas de la tabla de la derecha df2 y los empareja con filas de df1 en el índice. Y finalmente, el *unión externa*, que es la abreviatura de *unión externa completa*, toma la unión de índices de ambos DataFrames y coincide con los valores donde puede. [Tabla 5-5](#) es el equivalente de [Figura 5-3](#) en forma de texto.

Tabla 5-5. Tipos de unión

Escribe	Descripción
interno	Solo filas cuyo índice existe en ambos DataFrames
izquierda	Todas las filas del DataFrame izquierdo, filas coincidentes del DataFrame derecho Todas
Derecha	las filas del DataFrame derecho, filas coincidentes del DataFrame izquierdo La unión de
exterior	índices de fila de ambos DataFrames

Veamos cómo funciona esto en la práctica, trayendo los ejemplos de [Figura 5-3](#) a la vida:

En [74]: df1 = pd.Marco de datos(datos=[[1, 2], [3, 4], [5, 6]], columnas=["A", "B"])

df1

Fuera [74]:

	A	B
0	1	2
1	3	4
2	5	6

En [75]: df2 = pd.Marco de datos(datos=[[10, 20], [30, 40]], columnas=["C", "D"], índice=[1, 3])

df2

Fuera [75]:

	C	D
1	10	20
3	30	40

En [76]: df1.entrar(df2, cómo="interno")

Fuera [76]: ABCD

1	3	4	10	20
---	---	---	----	----

En [77]: df1.entrar(df2, cómo="izquierda")

Fuera [77]:

	A	B	C	D
0	1	2	Yaya	Yaya
1	3	4	10,0	20,0
2	5	6	NaN	NaN

En [78]: df1.entrar(df2, cómo="Derecha")

Fuera [78]: ABCD

1	3,0	4,0	10	20
3	NaN	NaN	30	40

En [79]: df1.entrar(df2, cómo="exterior")

Fuera [79]:

	A	B	C	D
0	1,0	2,0	Yaya	Yaya
1	3,0	4,0	10,0	20,0
2	5,0	6,0	NaN	NaN
			3	NaN
			NaN	30,0
				40,0

Si desea unirse a una o más columnas de DataFrame en lugar de depender del índice, use `unir` en lugar de `entrar`. `unir` acepta el sobre argumento para proporcionar uno o

más columnas como el *condición de unión*: estas columnas, que deben existir en ambos marcos de datos, se utilizan para hacer coincidir las filas:

```
En [80]: # Agregue una columna llamada "categoría" a ambos DataFrames  
df1["categoría"] = ["a", "B", "C"] df2[  
    "categoría"] = ["C", "B"]
```

En [81]: df1

```
Fuera [81]:      A  Categoría B  
0   1   2           a  
1   3   4           B  
2   5   6           C
```

En [82]: df2

```
Fuera [82]:      C  Categoría D  
1   10  20          C  
3   30  40          B
```

En [83]: df1.unir(df2, cómo="interno", sobre=["categoría"]) Fuera

```
[83]:      A  Categoría B      C  D  
0   3   4           B  30  40  
1   5   6           C  10  20
```

En [84]: df1.unir(df2, cómo="izquierda", sobre=["categoría"]) Fuera

```
[84]:      A  Categoría B      C  D  
0   1   2           a  Yaya  Yaya  
1   3   4           B  30,0  40,0  
2   5   6           C  10,0  20,0
```

Ya que entrar y unir aceptar bastantes argumentos opcionales para adaptarse a escenarios más complejos, los invito a echar un vistazo a la [documentación oficial](#) para aprender más sobre ellos.

Ahora sabe cómo manipular uno o más DataFrames, lo que nos lleva al siguiente paso en nuestro viaje de análisis de datos: dar sentido a los datos.

## Estadística descriptiva y agregación de datos

Una forma de dar sentido a los grandes conjuntos de datos es calcular una estadística descriptiva como la suma o la media en todo el conjunto de datos o en subconjuntos significativos. Esta sección comienza observando cómo funciona esto con los pandas antes de presentar dos formas de agregar datos en subconjuntos: el agrupar por método y el tabla dinámica función.

### Estadísticas descriptivas

*Estadísticas descriptivas* le permite resumir conjuntos de datos mediante el uso de medidas cuantitativas. Por ejemplo, el número de puntos de datos es una estadística descriptiva simple. Los promedios como la media, la mediana o la moda son otros ejemplos populares. DataFrames y Series permiten

acceder a estadísticas descriptivas cómodamente a través de métodos como suma, media, y contar, por nombrar unos cuantos. Conocerá a muchos de ellos a lo largo de este libro, y la lista completa está disponible a través del [documentación de pandas](#). De forma predeterminada, devuelven una serie junto eje = 0, lo que significa que obtienes la estadística de las columnas:

En [85]: lluvia.Fuera

```
[85]:      Ciudad 1    Ciudad 2    Ciudad 3  
0     300,1     400,3   1000,5  
1     100,2     300,4   1100,6
```

En [86]: lluvia.significar()

```
Fuera [86]: Ciudad 1      200.15  
             Ciudad 2      350.35  
             Ciudad 3      1050.55  
             dtype: float64
```

Si desea la estadística por fila, proporcione la eje argumento:

En [87]: lluvia.significar(eje=1)

```
Fuera [87]: 0      566.966667  
             1      500.400000  
             dtype: float64
```

De forma predeterminada, los valores perdidos no se incluyen en estadísticas descriptivas como suma o significar. Esto está en línea con la forma en que Excel trata las celdas vacías, por lo que el uso de Excel PROMEDIO fórmula en un rango con celdas vacías le dará el mismo resultado que el significar método aplicado en una Serie con los mismos números y Yaya valores en lugar de celdas vacías.

Obtener una estadística en todas las filas de un DataFrame a veces no es lo suficientemente bueno y necesita información más granular, por ejemplo, la media por categoría. ¡Veamos cómo se hace!

## Agrupamiento

Usando nuestro DataFrame de muestra df de nuevo, averigüemos la puntuación media por continente. Para hacer esto, primero agrupa las filas por continente y luego aplica el significar método, que calculará la media *por grupo*. Todas las columnas no numéricas se excluyen automáticamente:

En [88]: df.agrupar por(["continente"]).significar()

```
Fuera [88]: propiedades      la edad      puntaje  
            continente  
            America       37,0      5.30  
            Europa        33,5      6,75
```

Si incluye más de una columna, el DataFrame resultante tendrá un índice jerárquico, el MultiIndex que conocimos anteriormente:

En [89]: df.agrupar por(["continente", "país"]).significar()

Fuera [89]: propiedades

	país	continente	la edad	puntaje
America	Estados Unidos	America	37	5.3
Europa	Alemania	Europa	12	9.0
	Italia		55	4.5

En lugar de significar, puede utilizar la mayoría de las estadísticas descriptivas que ofrece pandas y si desea utilizar su propia función, utilice el agg método. Por ejemplo, así es como obtiene la diferencia entre el valor máximo y mínimo por grupo:

En [90]: df.agrupar por(["continente"]).agg(lambda X: X.max() - X.min())

Fuera [90]: propiedades

	continente	la edad	puntaje
America	America	8	2.8
Europa	Europa	43	4.5

Una forma popular de obtener estadísticas por grupo en Excel es utilizar tablas dinámicas. Introducen una segunda dimensión y son excelentes para ver sus datos desde diferentes perspectivas. pandas también tiene una funcionalidad de tabla dinámica, como veremos a continuación.

## Girar y derretir

Si está utilizando tablas dinámicas en Excel, no tendrá problemas para aplicar pandas ' tabla dinámica función, ya que funciona en gran parte de la misma manera. Los datos en el siguiente DataFrame están organizados de manera similar a cómo se almacenan típicamente los registros en una base de datos; cada fila muestra una transacción de venta para una fruta específica en una determinada región:

En [91]: datos = [[{"Fruta": "Naranjas", "Región": "Norte", "Ingresos": 12.3}, {"Fruta": "Manzanas", "Región": "Sur", "Ingresos": 10.55}, {"Fruta": "Naranjas", "Región": "Sur", "Ingresos": 22.0}, {"Fruta": "Plátanos", "Región": "Sur", "Ingresos": 5.9}, {"Fruta": "Plátanos", "Región": "Norte", "Ingresos": 31.3}, {"Fruta": "Naranjas", "Región": "Norte", "Ingresos": 13.1}]]

```
Ventas = pd.Marco de datos(datos=datos,
                           columnas=["Fruta", "Región", "Ingresos"])
Ventas
```

Fuera [91]:

	Región	Frutal	Ingresos
0	norte	Naranjas	12.3
1	Sur	Manzanas	10.55
2	Sur	Naranjas	22.0
3	Sur	Plátanos	5.9
4	norte	Plátanos	31.3
5	norte	Naranjas	13.1

Para crear una tabla dinámica, proporcione el DataFrame como primer argumento al tabla dinámica función. índice y columnas defina qué columna del DataFrame se convertirá en las etiquetas de fila y columna de la tabla dinámica, respectivamente. valores se agregarán a la parte de datos del DataFrame resultante mediante el uso de aggfunc, a

función que se puede proporcionar como una cadena o NumPy ufunc. Y finalmente, márgenes corresponden a las Gran total en Excel, es decir, si te vas márgenes y margins\_name lejos, el Total la columna y la fila no se mostrarán:

```
En [92]: pivot = pd.tabla dinámica(Ventas,  
                                     índice="Fruta", columnas="Región", valores=  
                                     ="Ingresos", aggfunc="suma", márgenes=  
                                     Cierto, margins_name="Total")  
pivot
```

```
Fuera [92]: Región      norte   Sur    Total  
Fruta  
Manzanas      Yaya  10,55  10,55  
Plátanos     31,3   5,90  37,20  
Naranjas      25,4   22.00 47,40  
Total         56,7   38,45 95.15
```

En resumen, pivotar sus datos significa tomar los valores únicos de una columna (Región en nuestro caso) y convertirlos en los encabezados de columna de la tabla dinámica, agregando así los valores de otra columna. Esto facilita la lectura de información resumida en las dimensiones de interés. En nuestra tabla dinámica, verá instantáneamente que no se vendieron manzanas en la región norte y que en la región sur, la mayoría de los ingresos provienen de las naranjas. Si desea ir al revés y convertir los encabezados de columna en los valores de una sola columna, use derretir. En ese sentido, derretir es lo opuesto al tabla dinámica función:

```
En [93]: pd.derretir(pivot.iloc[:-1,:-1].reset_index(),  
                     id_vars="Fruta",  
                     value_vars=["Norte", "Sur"], value_name="Ingresos")
```

```
Fuera [93]:      Región Frutal    Ingresos  
0   Manzanas    norte      Yaya  
1   Plátanos    norte     31.30  
2   Naranjas    norte     25.40  
3   Manzanas    Sur       10,55  
4   Plátanos    Sur       5,90  
5   Naranjas    Sur       22.00
```

Aquí, proporciono nuestra tabla dinámica como entrada, pero estoy usando iloc para deshacerse del total de filas y columnas. También restablezco el índice para que toda la información esté disponible como columnas regulares. Entonces proporciono id\_vars para indicar los identificadores y value\_vars para definir qué columnas quiero "desvincular". La fusión puede ser útil si desea preparar los datos para que puedan almacenarse en una base de datos que los espera en este formato.

Trabajar con estadísticas agregadas lo ayuda a comprender sus datos, pero a nadie le gusta leer una página llena de números. Para hacer que la información sea fácilmente comprensible, nada funciona mejor que crear visualizaciones, que es nuestro siguiente tema. Mientras que Excel usa el término *gráficos*, los pandas generalmente se refieren a ellos como *parcelas*. Utilizaré estos términos indistintamente en este libro.

## Graficado

El trazado le permite visualizar los resultados de su análisis de datos y bien puede ser el paso más importante de todo el proceso. Para el trazado, vamos a utilizar dos bibliotecas: comenzamos mirando Matplotlib, la biblioteca de trazado predeterminada de los pandas, antes de centrarnos en Plotly, una biblioteca de trazado moderna que nos brinda una experiencia más interactiva en los cuadernos de Jupyter.

### Matplotlib

Matplotlib es un paquete de trazado que existe desde hace mucho tiempo y se incluye en la distribución de Anaconda. Con él, puede generar gráficos en una variedad de formatos, incluidos gráficos vectoriales para una impresión de alta calidad. Cuando llamas al trama método de un DataFrame, los pandas producirán un gráfico Matplotlib de forma predeterminada.

Para usar Matplotlib en un cuaderno de Jupyter, primero debe ejecutar uno de los dos comandos mágicos (consulte la barra lateral "Comandos mágicos" en la página 116):%matplotlib en línea o Cuaderno% matplotlib. Configuran el portátil para que los trazados se puedan mostrar en el propio portátil. El último comando agrega un poco más de interactividad, lo que le permite cambiar el tamaño o el factor de zoom del gráfico. Comencemos y creemos una primera trama con pandas y Matplotlib (ver Figura 5-4):

```
En [94]: importar numpy como notario público  
%matplotlib en línea  
# O% cuaderno matplotlib  
  
En [95]: datos = pd.Marco de datos(datos=notario público.aleatorio.rand(4, 4) * 100000,  
índice=[ "Q1", "Q2", "Q3", "Q4"], columnas=[ "Este",  
"Oeste", "Norte", "Sur"] )  
datos.índice.nombre = "Cuartelos"  
datos.columnas.nombre = "Región"  
datos
```

	Este	Oeste	Norte	Sur
Cuartelos				
Q1	23254.220271	96398.309860	16845.951895	41671.684909
Q2	87316.022433	45183.397951	15460.819455	50951.465770
Tercer trimestre	51458.760432	3821.139360	77793.393899	98915.952421
Cuarto trimestre	64933.848496	7600.277035	55001.831706	86248.512650

```
En [96]: datos.trama() # Atajo para data.plot.line()
```

```
Fuera [96]: <AxesSubplot: xlabel = 'Quarters'>
```

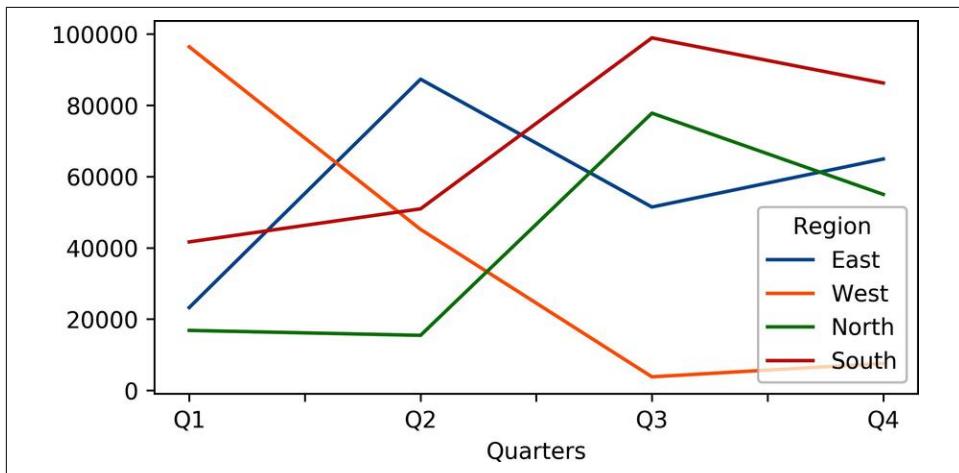


Figura 5-4. Gráfico de Matplotlib

Tenga en cuenta que en este ejemplo, he usado una matriz NumPy para construir un marco de datos pandas. Proporcionar matrices NumPy le permite aprovechar los constructores de NumPy que conocemos en el capítulo anterior; aquí, usamos NumPy para generar un DataFrame de pandas basado en números pseudoaleatorios. Por lo tanto, cuando ejecute la muestra en su extremo, obtendrá valores diferentes.

### Comandos mágicos

Los %matplotlib en línea El comando que usamos para hacer que Matplotlib funcione correctamente con los cuadernos de Jupyter es un *comando mágico*. Los comandos mágicos son un conjunto de comandos simples que hacen que la celda de un cuaderno de Jupyter se comporte de cierta manera o hacen que algunas tareas engorrosas sean tan fáciles que casi se siente como magia. Escribe comandos mágicos en celdas como el código Python, pero comienzan con %% o %. Los comandos que afectan a toda la celda comienzan con %%, y los comandos que solo afectan a una sola línea en una celda comienzan con %.

Veremos más comandos mágicos en los próximos capítulos, pero si desea enumerar todos los comandos mágicos disponibles actualmente, ejecute%lsmagic, y para una descripción detallada, ejecute%magia.

Incluso si usa el comando mágico%cuaderno matplotlib, probablemente notará que Matplotlib fue diseñado originalmente para gráficos estáticos en lugar de para una experiencia interactiva en una página web. Es por eso que a continuación usaremos Plotly, una biblioteca de trazado diseñada para la web.

## Plotly

Plotly es una biblioteca basada en JavaScript y puede, desde la versión 4.8.0, usarse como un backend de trazado de pandas con gran interactividad: puede acercar fácilmente, hacer clic en la leyenda para seleccionar o anular la selección de una categoría y obtener información sobre herramientas con más información sobre el punto de datos sobre el que se desplaza. Plotly no está incluido en la instalación de Anaconda, por lo que si aún no lo ha instalado, hágalo ahora ejecutando el siguiente comando:

```
(base)> conda instalar plotly
```

Una vez que ejecute la siguiente celda, el backend de trazado de todo el cuaderno se establecerá en Plotly y si vuelve a ejecutar la celda anterior, también se representará como un gráfico de Plotly. Para Plotly, en lugar de ejecutar un comando mágico, solo necesita configurarlo como backend antes de poder trazar Figuras 5-5 y 5-6:

```
En [97]: # Establecer el backend de trazado en Plotly  
pd.opciones.Graficado.backend = "trama"
```

```
En [98]: datos.trama()
```

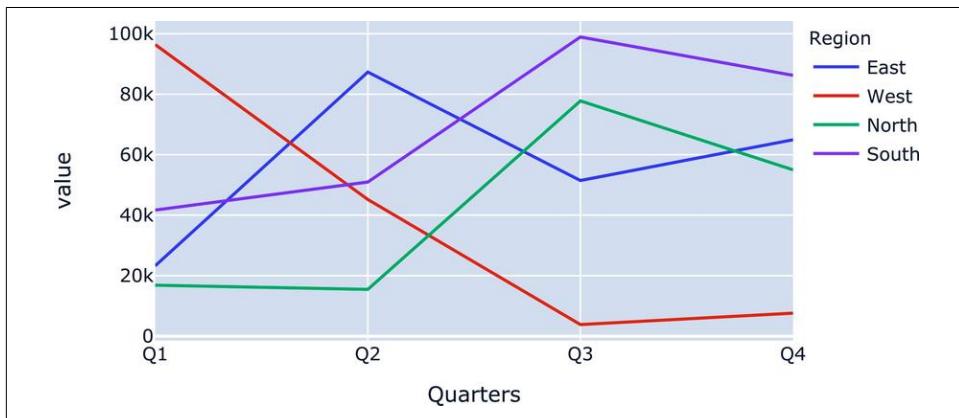


Figura 5-5. Trazar trazado de líneas

```
En [99]: # Mostrar los mismos datos que el diagrama de barras  
datos.trama.bar(modo de bar="grupo")
```

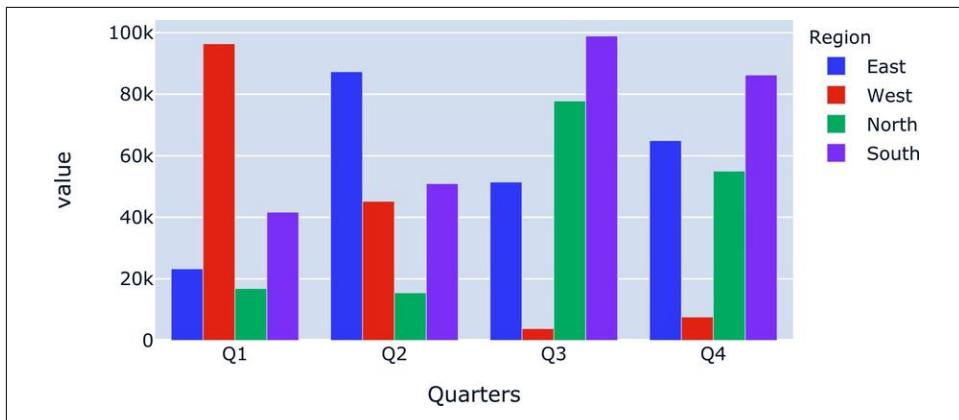


Figura 5-6. Gráfico de barras de trazado



#### Diferencias en el trazado de backends

Si usa Plotly como backend de trazado, deberá verificar los argumentos aceptados de los métodos de trazado directamente en los documentos de Plotly. Por ejemplo, puede echar un vistazo al argumento `group` en [Documentación de gráficos de barras de Plotly](#).

pandas y las bibliotecas de trazado subyacentes ofrecen una gran cantidad de tipos de gráficos y opciones para dar formato a los gráficos en casi cualquier forma deseada. También es posible organizar varias parcelas en una serie de subtramas. Como resumen, [Tabla 5-6](#) muestra los tipos de parcelas disponibles.

Tabla 5-6. tipos de parcelas de pandas

Escribe	Descripción
línea	Gráfico de líneas, predeterminado cuando se ejecuta <code>df.plot()</code>
bar	Gráfico de barras verticales
barh	Gráfico de barras horizontales
hist	Histograma
caja	Diagrama de caja
kde	Gráfico de densidad, también se puede utilizar a través de <code>densidad</code>
zona	Gráfico de área
dispersión	Gráfico de dispersión
hexbin	Parcelas de contenedores hexagonales
tarta	Gráfico circular

Además de eso, pandas ofrece algunas herramientas y técnicas de trazado de nivel superior que se componen de múltiples componentes individuales. Para obtener más detalles, consulte la [documentación de visualización de pandas](#).

### Otras bibliotecas de trazado

El panorama de visualización científica en Python es muy activo, y además de Matplotlib y Plotly, hay muchas otras opciones de alta calidad para elegir que pueden ser la mejor opción para ciertos casos de uso:

#### *Seaborn*

[Seaborn](#) está construido sobre Matplotlib. Mejora el estilo predeterminado y agrega gráficos adicionales como mapas de calor, que a menudo simplifican su trabajo: puede crear gráficos estadísticos avanzados con solo unas pocas líneas de código.

#### *Bokeh*

[Bokeh](#) es similar a Plotly en tecnología y funcionalidad: está basado en JavaScript y, por lo tanto, también funciona muy bien para gráficos interactivos en cuadernos de Jupyter. Bokeh está incluido en Anaconda.

#### *Altair*

[Altair](#) es una biblioteca para visualizaciones estadísticas basadas en el [Proyecto Vega](#). Altair también está basado en JavaScript y ofrece cierta interactividad como hacer zoom.

#### *HoloViews*

[HoloViews](#) es otro paquete basado en JavaScript que se centra en facilitar el análisis y la visualización de datos. Con unas pocas líneas de código, puede lograr gráficos estadísticos complejos.

Crearemos más gráficos en el próximo capítulo para analizar series de tiempo, pero antes de llegar allí, terminemos este capítulo aprendiendo cómo podemos importar y exportar datos con pandas.

## Importación y exportación de marcos de datos

Hasta ahora, construimos DataFrames desde cero usando listas anidadas, diccionarios o matrices NumPy. Es importante conocer estas técnicas, pero por lo general, los datos ya están disponibles y simplemente necesita convertirlos en un DataFrame. Para hacer esto, pandas ofrece varias funciones de lectura. Pero incluso si necesita acceder a un sistema propietario para el que pandas no ofrece un lector incorporado, a menudo tiene un paquete de Python para conectarse a ese sistema, y una vez que tiene los datos, es bastante fácil convertirlo en un Marco de datos. En Excel, la importación de datos es el tipo de trabajo que normalmente manejaría con Power Query.

Después de analizar y cambiar su conjunto de datos, es posible que desee enviar los resultados nuevamente a una base de datos o exportarlos a un archivo CSV o, dado el título del libro, presentarlos en un libro de Excel a su gerente. Para exportar Pandas DataFrames, utilice uno de los métodos de exportación que ofrece DataFrames. **Tabla 5-7** muestra una descripción general de los métodos de importación y exportación más comunes.

*Tabla 5-7. Importar y exportar DataFrames*

Formato / sistema de datos	Importar: función pandas (pd)	Exportar: método DataFrame (df)
Archivos CSV	pd.read_csv	df.to_csv
JSON	pd.read_json	df.to_json
HTML	pd.read_html	df.to_html
Portapapeles	pd.read_clipboard	df.to_clipboard
Archivos de Excel	pd.read_excel	df.to_excel
Bases de datos SQL	pd.read_sql	df.to_sql

Nos reuniremos pd.read\_sql y pd.to\_sql en [Capítulo 11](#), donde los usaremos como parte de un estudio de caso. Y ya que te voy a dedicar todo [Capítulo 7](#) al tema de leer y escribir archivos de Excel con pandas, me enfocaré en importar y exportar archivos CSV en esta sección. ¡Comencemos por exportar un DataFrame existente!

#### Exportación de archivos CSV

Si necesita pasar un DataFrame a un colega que podría no usar Python o pandas, pasarlo en forma de archivo CSV suele ser una buena idea: casi todos los programas saben cómo importarlos. Para exportar nuestro DataFrame de muestra df a un archivo CSV, utilice el to\_csv método:

En [100]: `df.to_csv("course_participants.csv")`

Si desea almacenar el archivo en un directorio diferente, proporcione la ruta completa como una cadena, por ejemplo, r "C:\ ruta \ a \ ubicación \ deseada \ msft.csv".



#### Usar cadenas sin formato para rutas de archivo en Windows

En cadenas, la barra invertida se usa para escapar de ciertos caracteres. Es por eso que para trabajar con rutas de archivo en Windows, debe usar barras diagonales inversas dobles (C:\\ ruta \\ archivo.csv) o prefijar la cadena con un r para convertirlo en un *cuerda cruda* que interpreta a los personajes literalmente. Esto no es un problema en macOS o Linux, ya que usan barras diagonales en las rutas.

Al proporcionar solo el nombre del archivo como lo hago, producirá el archivo *course\_participants.csv* en el mismo directorio que el cuaderno con el siguiente contenido:

```
user_id, nombre, edad, país, puntuación, continente  
1001, Mark, 55, Italia, 4.5, Europa  
1000, John, 33, EE. UU., 6.7, America 1002,  
Tim, 41, EE. UU., 3.9, America 1003, Jenny,  
12, Alemania, 9.0, Europa
```

Ahora que sabe cómo utilizar el `df.to_csv` método, ¡veamos cómo funciona la importación de un archivo CSV!

### Importación de archivos CSV

Importar un archivo CSV local es tan fácil como proporcionar su ruta al `read_csv` función.

`MSFT.csv` es un archivo CSV que descargué de Yahoo! Finance y contiene los precios históricos diarios de las acciones de Microsoft; lo encontrará en el repositorio complementario, en la `csv` carpeta:

```
En [101]: msft = pd.read_csv("csv / MSFT.csv")
```

A menudo, necesitará proporcionar algunos parámetros más para `read_csv` que solo el nombre del archivo. Por ejemplo, `sep` le permite decirle a los pandas qué separador o delimitador usa el archivo CSV en caso de que no sea la coma predeterminada. Usaremos algunos parámetros más en el próximo capítulo, pero para una descripción completa, eche un vistazo a la [documentación de pandas](#).

Ahora que estamos tratando con grandes DataFrames con muchos miles de filas, normalmente lo primero es ejecutar el `info` método para obtener un resumen del DataFrame. A continuación, es posible que desee echar un vistazo a la primera y última fila del DataFrame usando el `cabeza` y `cola` métodos. Estos métodos devuelven cinco filas de forma predeterminada, pero esto se puede cambiar proporcionando el número deseado de filas como argumento. También puede ejecutar el `describir` método para obtener algunas estadísticas básicas:

```
En [102]: msft.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 8622 entradas, 0 a 8621  
columnas de datos (total 7 columnas):  
 # Columna      Recuento no nulo   Dtype  
-----  
 0  Fecha        8622 no nulo     objeto  
 1  Abierto       8622 no nulo     float64  
 2  Elevado       8622 no nulo     float64  
 3  Bajo          8622 no nulo     float64  
 4  Cerrar         8622 no nulo     float64  
 5  Adj Cerrar    8622 no nulo     float64  
 6  Volumen        8622 no nulo     int64  
dtypes: float64 (5), int64 (1), object (1) uso de  
memoria: 471.6+ KB
```

```
En [103]: # Estoy seleccionando algunas columnas debido a problemas de espacio.  
          # También puede ejecutar: msft.head()  
          msft.loc[:, ["Fecha", "Adj cerrar", "Volumen"]].cabeza()
```

```
Fuera [103]:          Fecha   Adj Cerrar      Volumen
0 1986-03-13    0.062205  1031788800
1 14 de marzo de 1986  0.064427  308160000
2 1986-03-17    0.065537  133171200
3 1986-03-18    0.063871  67766400
4 1986-03-19    0.062760  47894400
```

En [104]: msft.loc[:, ["Fecha", "Adj cerrar", "Volumen"]].cola(2)

Salida [104]: Volumen de cierre de ajuste de fecha  
8620 2020-05-26 181.570007 36073600  
8621 2020-05-27 181.809998 39492600

En [105]: msft.loc[:, ["Adj cerrar", "Volumen"]].describir() Fuera

```
[105]:          Adj Cerrar      Volumen
contar    8622.000000  8.622000e + 03
significar  24,921952  6.030722e + 07
std       31.838096  3.877805e + 07
min        0.057762  2.304000e + 06
25%       2.247503  3.651632e + 07
50%       18.454313  5.350380e + 07
75%       25.699224  7.397560e + 07
max      187.663330  1.031789e + 09
```

Adj Cerrar representa *precio de cierre ajustado* y corrige el precio de las acciones por acciones corporativas como la división de acciones. Volumen es el número de acciones que se negociaron. He resumido los diversos métodos de exploración de DataFrame que hemos visto en este capítulo en **Tabla 5-8**.

*Tabla 5-8. Métodos y atributos de exploración de DataFrame*

Método / atributo DataFrame (df)	Descripción
df.info ()	Proporciona el número de puntos de datos, tipo de índice, tipo d y uso de memoria.
df.describe ()	Proporciona estadísticas básicas que incluyen recuento, media, estándar, mínimo, máximo y percentiles. Devuelve el primeronorte filas del DataFrame. Devuelve el últimonorte filas del DataFrame. Devuelve el dtype de cada columna.
gl. cola (n = 5)	
df.dtypes	

los read\_csv La función también acepta una URL en lugar de un archivo CSV local. Así es como lee el archivo CSV directamente desde el repositorio complementario:

En [106]: # El salto de línea en la URL es solo para que encaje en la página  
url = ("https://raw.githubusercontent.com/fzumstein/  
"python-para-excel / 1st-edition / csv / MSFT.csv")  
msft = pd.read\_csv(url)

En [107]: msft.loc[:, ["Fecha", "Adj cerrar", "Volumen"]].cabeza(2) Fuera

```
[107]:          Fecha   Adj Cerrar      Volumen
0 1986-03-13    0.062205  1031788800
1 14 de marzo de 1986  0.064427  308160000
```

Continuaremos con este conjunto de datos y el `read_csv` función en el próximo capítulo sobre series de tiempo, donde giraremos la Fecha columna en una `DatetimeIndex`.

## Conclusión

Este capítulo estaba repleto de nuevos conceptos y herramientas para analizar conjuntos de datos en pandas. Aprendimos cómo cargar archivos CSV, cómo lidar con datos faltantes o duplicados y cómo hacer uso de estadísticas descriptivas. También vimos lo fácil que es convertir sus `DataFrames` en gráficos interactivos. Si bien puede tomar un tiempo digerir todo, probablemente no pasará mucho tiempo antes de que comprenda el inmenso poder que está obteniendo al agregar pandas a su cinturón de herramientas. En el camino, comparamos pandas con la siguiente funcionalidad de Excel:

### *Funcionalidad de autofiltro*

Ver "[Selección mediante indexación booleana](#)" en la página 94.

### *Fórmula VLOOKUP*

Ver "[Unión y fusión](#)" en la página 109.

### *Tabla dinámica*

Ver "[Girar y fundir](#)" en la página 113.

### *Consulta de energía*

Esta es una combinación de "[Importación y exportación de marcos de datos](#)" en la página 119, "[Manipulación de datos](#)" en la página 91, y "[Combinación de DataFrames](#)" en la página 107.

El siguiente capítulo trata sobre el análisis de series de tiempo, la funcionalidad que llevó a una amplia adopción de pandas por parte de la industria financiera. ¡Veamos por qué esta parte de los pandas tiene tanta ventaja sobre Excel!



## Análisis de series temporales con pandas

A *series de tiempo* es una serie de puntos de datos a lo largo de un eje basado en el tiempo que juega un papel central en muchos escenarios diferentes: mientras que los comerciantes utilizan los precios históricos de las acciones para calcular las medidas de riesgo, el pronóstico del tiempo se basa en series de tiempo generadas por sensores que miden la temperatura, la humedad y presión de aire. Y el departamento de marketing digital se basa en series de tiempo generadas por las páginas web, por ejemplo, la fuente y el número de páginas vistas por hora, y las utilizará para sacar conclusiones con respecto a sus campañas de marketing.

El análisis de series de tiempo es una de las principales fuerzas impulsoras por las que los científicos y analistas de datos han comenzado a buscar una mejor alternativa a Excel. Los siguientes puntos resumen algunas de las razones detrás de este movimiento:

### *Grandes conjuntos de datos*

Las series de tiempo pueden crecer rápidamente más allá del límite de Excel de aproximadamente un millón de filas por hoja. Por ejemplo, si trabaja con precios de acciones intradía en un nivel de datos de ticks, a menudo se trata de cientos de miles de registros, ¡por acción y día!

### *Fecha y hora*

Como hemos visto en [Capítulo 3](#), Excel tiene varias limitaciones cuando se trata de manejar la fecha y la hora, la columna vertebral de las series de tiempo. La falta de soporte para las zonas horarias y un formato de número que se limita a milisegundos son algunos de ellos. pandas admite zonas horarias y utiliza NumPydatetime64 [ns] tipo de datos, que ofrece una resolución de hasta nanosegundos.

### *Falta funcionalidad*

Excel echa de menos incluso las herramientas básicas para poder trabajar con datos de series de tiempo de una manera decente. Por ejemplo, si desea convertir una serie de tiempo diaria en una serie de tiempo mensual, no hay una manera fácil de hacerlo a pesar de ser una tarea muy común.

Los DataFrames le permiten trabajar con varios índices basados en el tiempo: DatetimeIndex es el más común y representa un índice con marcas de tiempo. Otros tipos de índices, como PeriodIndex, se basan en intervalos de tiempo como horas o meses. En este capítulo, sin embargo, solo estamos analizando DatetimeIndex, que presentaré ahora con más detalle.

## DatetimeIndex

En esta sección, aprenderemos a construir un DatetimeIndex, cómo filtrar dicho índice en un intervalo de tiempo específico y cómo trabajar con zonas horarias.

### Creando un DatetimeIndex

Para construir un DatetimeIndex, pandas ofrece la función de rango de fechas. Acepta una fecha de inicio, una frecuencia y el número de períodos o la fecha de finalización:

En [1]: # Comencemos importando los paquetes que usamos en este capítulo.

```
# y estableciendo el backend de trazado en Plotlyimportar pandas  
como pdimportar numpy como notario público
```

```
pd.opciones.Graficado.backend = "trama"
```

En [2]: # Esto crea un DatetimeIndex basado en una marca de tiempo de inicio,

```
# número de períodos y frecuencia ("D" = diario).daily_index = pd.rango de  
fechas("2020-02-28", períodos=4, frecuencia="D")daily_index
```

Fuera [2]: DatetimeIndex(['2020-02-28', '2020-02-29', '2020-03-01', '2020-03-02'],  
dtype = 'datetime64 [ns]', freq = 'D')

En [3]: # Esto crea un DatetimeIndex basado en la marca de tiempo de inicio / finalización.

```
# La frecuencia se establece en "semanalmente los domingos" ("W-SUN").índice_semanal =  
pd.rango de fechas("2020-01-01", "2020-01-31", frecuencia="W-SUN")índice_semanal
```

Fuera [3]: DatetimeIndex(['2020-01-05', '2020-01-12', '2020-01-19', '2020-01-26'],  
dtype = 'datetime64 [ns]', freq = 'W-SUN')

En [4]: # Construya un DataFrame basado en el índice\_semanal. Esto podría ser

```
# el recuento de visitantes de un museo que solo abre los domingos.pd  
.Marco de datos(datos=[21, 15, 33, 34],  
columnas=["visitantes"], índice=índice_semanal)
```

Fuera [4]: visitantes

2020-01-05	21
2020-01-12	15
2020-01-19	33
2020-01-26	34

Volvamos ahora a la serie temporal de acciones de Microsoft del último capítulo. Cuando observe más de cerca los tipos de datos de las columnas, notará que el Fecha

la columna tiene el tipo objeto, lo que significa que pandas ha interpretado las marcas de tiempo como cadenas:

```
En [5]: msft = pd.read_csv("csv / MSFT.csv")En [6]
]: msft.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8622 entradas, 0 a 8621
columnas de datos (total 7 columnas):
 # Columna      Recuento no nulo   Dtype
---  -----
 0  Fecha        8622 no nulo     objeto
 1  Abierto      8622 no nulo     float64
 2  Elevado      8622 no nulo     float64
 3  Bajo          8622 no nulo     float64
 4  Cerrar        8622 no nulo     float64
 5  Adj Cerrar    8622 no nulo     float64
 6  Volumen       8622 no nulo     int64
dtypes: float64 (5), int64 (1), object (1) uso de
memoria: 471.6+ KB
```

Hay dos formas de solucionar este problema y convertirlo en un fecha y hora tipo de datos. El primero es ejecutar elto\_datetime función en esa columna. Asegúrese de volver a asignar la columna transformada al DataFrame original si desea cambiarlo en la fuente:

```
En [7]: msft.loc[:, "Fecha"] = pd.to_datetime(msft["Fecha"])En [8]:
msft.dtipos

Salida [8]: fecha           datetime64 [ns]
            Abierto         float64
            Elevado         float64
            Bajo             float64
            Cerrar           float64
            Adj Cerrar       float64
            Volumen          int64
dtype: objeto
```

La otra posibilidad es contar read\_csv sobre las columnas que contienen marcas de tiempo mediante el uso de parse\_dates argumento. parse\_dates espera una lista de nombres de columnas o índices.

Además, casi siempre desea convertir las marcas de tiempo en el índice del marco de datos, ya que esto le permitirá filtrar los datos fácilmente, como veremos en un momento. Para ahorrarte un extra set\_index llamar, proporcione la columna que le gustaría usar como índice a través de la index\_col argumento, nuevamente como nombre de columna o índice:

```
En [9]: msft = pd.read_csv("csv / MSFT.csv",
                           index_col="Fecha", parse_dates=["Fecha"])
```

```
En [10]: msft.info()
```

```
<class 'pandas.core.frame.DataFrame'> DatetimeIndex: 8622
entradas, 1986-03-13 a 2020-05-27 Columnas de datos (total 6
columnas):
```

```

# Columna      Recuento no nulo   Dtype
-----      -----
0    Abierto      8622 no nulo     float64
1    Elevado      8622 no nulo     float64
2    Bajo         8622 no nulo     float64
3    Cerrar       8622 no nulo     float64
4    Adj Cerrar   8622 no nulo     float64
5    Volumen      8622 no nulo     int64
dtypes: float64 (5), int64 (1) uso
de memoria: 471,5 KB

```

Como info revela, ahora se trata de un DataFrame que tiene un DatetimeIndex. Si necesita cambiar otro tipo de datos (digamos que desea Volumen ser un flotador en lugar de un Entero), de nuevo tienes dos opciones: proporcionar `dtype = {"Volumen": float}` como argumento para el `read_csv` función, o aplicar la `astipo` método de la siguiente manera:

```
En [11]: msft.loc[:, "Volumen"] = msft["Volumen"].astipo("float")
msft["Volumen"].dtype
```

Fuera [11]: `dtype ('float64')`

Con las series de tiempo, siempre es una buena idea asegurarse de que el índice esté ordenado correctamente antes de comenzar su análisis:

```
En [12]: msft = msft.sort_index()
```

Y finalmente, si necesita acceder solo a partes de un DatetimeIndex, como la parte de la fecha sin la hora, acceda al fecha atributo como este:

```
En [13]: msft.index.fecha
```

```
Out [13]: array ([datetime.date (1986, 3, 13), datetime.date (1986, 3, 14),
                  datetime.date (1986, 3, 17), ..., datetime.date (2020, 5, 22),
                  datetime.date (2020, 5, 26), datetime.date (2020, 5, 27)], dtype =
objeto)
```

En lugar de fecha, también puedes usar partes de una fecha como año mes dia, etc. Para acceder a la misma funcionalidad en una columna regular con el tipo de datos fecha y hora, tendrás que utilizar el `dt` atributo, por ejemplo, `df["nombre_columna"].dt.date`.

Con un ordenado DatetimeIndex, ¡Veamos cómo podemos filtrar el DataFrame a ciertos períodos de tiempo!

## Filtrar un índice de fecha y hora

Si su DataFrame tiene un DatetimeIndex, hay una manera fácil de seleccionar filas de un período de tiempo específico usando `loc` con una cadena en el formato AAAA-MM-DD HH:MM:SS. Los pandas convertirán esta cuerda en una rebanada para que cubra todo el período. Por ejemplo, para seleccionar todas las filas de 2019, proporcione el año como *cuerda*, no un número:

En [14]: `msft.loc["2019", "Adj cerrar"]`

Fuera [14]: fecha

2019-01-02	99.099190
2019-01-03	95.453529
2019-01-04	99.893005
2019-01-07	100.020401
2019-01-08	100.745613
	...
2019-12-24	156.515396
2019-12-26	157.798309
2019-12-27	158.086731
2019-12-30	156.724243
2019-12-31	156.833633

Nombre: Adj Close, Longitud: 252, dtipo: float64

Llevemos esto un paso más allá y grafiquemos los datos entre junio de 2019 y mayo de 2020 (consulte Figura 6-1):

En [15]: `msft.loc["2019-06":"2020-05", "Adj cerrar"].trama()`

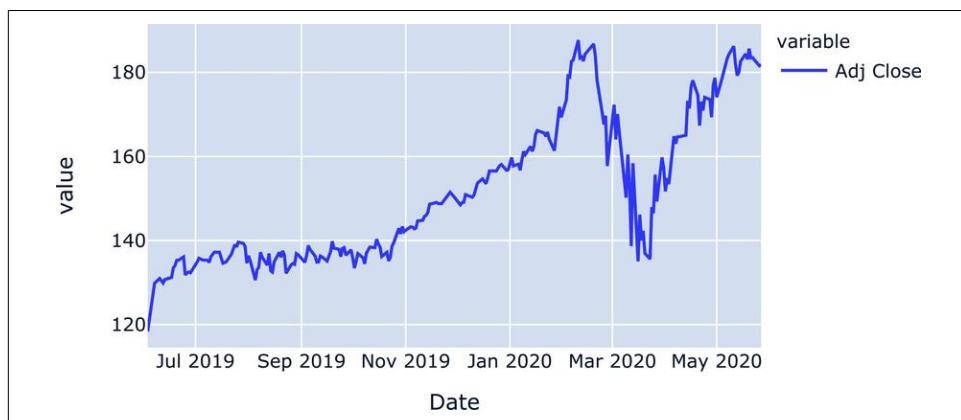


Figura 6-1. Precio de cierre ajustado para MSFT

Desplácese sobre el gráfico Plotly para leer el valor como información sobre herramientas y acerque dibujando un rectángulo con el mouse. Haga doble clic en el gráfico para volver a la vista predeterminada.

Usaremos el precio de cierre ajustado en la siguiente sección para aprender sobre el manejo de la zona horaria.

## Trabajar con zonas horarias

Microsoft cotiza en la bolsa de valores Nasdaq. El Nasdaq está en Nueva York y los mercados cierran a las 4:00 pm Para agregar esta información adicional al índice del DataFrame, primero agregue la hora de cierre a la fecha a través de `deDateOffset`, luego adjunte el correcto

zona horaria a las marcas de tiempo a través de tz\_localize. Dado que la hora de cierre solo es aplicable al precio de cierre, creamos un nuevo DataFrame con él:

```
En [dieciséis]: # Agregue la información de la hora a la fecha
msft_close = msft.loc[:, ["Adj cerrar"]].Copiar()
msft_close.índice = msft_close.índice + pd.DateOffset(horas=dieciséis)
msft_close.cabeza(2)
```

Fuera [16]:

	Adj Cerrar
Fecha	
13 de marzo de 1986 16:00:00	0.062205
14/03/1986 16:00:00	0.064427

```
En [17]: # Hacer que las marcas de tiempo tengan en cuenta la zona horaria
msft_close = msft_close.tz_localize("America / New_York")
msft_close.cabeza(2)
```

Fuera [17]:

	Adj Cerrar
Fecha	
1986-03-13 16: 00: 00-05: 00	0.062205
1986-03-14 16: 00: 00-05: 00	0.064427

Si desea convertir las marcas de tiempo a la zona horaria UTC, use el método DataFrame tz\_convert. UTC significa *Hora universal coordinada* y es el sucesor de Greenwich Mean Time (GMT). Observe cómo cambian las horas de cierre en UTC dependiendo de si el horario de verano (DST) está en vigor o no en Nueva York:

```
En [18]: msft_close = msft_close.tz_convert("UTC")
msft_close.loc["2020-01-02", "Adj cerrar"] # 21:00 sin horario de verano
```

Fuera [18]: Fecha  
2020-01-02 21: 00: 00 + 00: 00 159.737595  
Nombre: Adj Close, dtype: float64

```
En [19]: msft_close.loc["2020-05-01", "Adj cerrar"] # 20:00 con horario de verano
```

Fuera [19]: Fecha  
2020-05-01 20: 00: 00 + 00: 00 174.085175  
Nombre: Adj Close, dtype: float64

La preparación de series de tiempo como esta le permitirá comparar los precios de cierre de las bolsas de valores en diferentes zonas horarias, incluso si falta la información de la hora o se indica en la zona horaria local.

Ahora que sabes lo que es DatetimeIndex Es decir, probemos algunas manipulaciones comunes de series de tiempo en la siguiente sección calculando y comparando el desempeño de las acciones.

## Manipulaciones comunes de series de tiempo

En esta sección, le mostraré cómo realizar tareas comunes de análisis de series de tiempo, como calcular los rendimientos de las acciones, graficar el rendimiento de varias acciones y visualizar la correlación de sus rendimientos en un mapa de calor. También veremos cómo cambiar la frecuencia de las series de tiempo y cómo calcular las estadísticas continuas.

### Cambios de cambio y porcentaje

En finanzas, el *devoluciones de registros* de las poblaciones se supone a menudo que se distribuyen normalmente. Por rendimientos logarítmicos, me refiero al logaritmo natural de la relación entre el precio actual y el anterior. Para tener una idea de la distribución de los retornos de registros diarios, tracemos un histograma. Sin embargo, primero debemos calcular los retornos del registro. En Excel, normalmente se hace con una fórmula que involucra celdas de dos filas, como se muestra en Figura 6-2.

	A	B	C
1	Date	Adj Close	
2	3/13/1986	0.062205	
3	3/14/1986	0.064427	=LN(B3/B2)
4	3/17/1986	0.065537	0.017082

Figura 6-2. Cálculo de devoluciones de registros en Excel



### Logaritmos en Excel y Python

Excel utiliza LN para denotar el logaritmo natural y INICIAR SESIÓN para el logaritmo con base 10. El módulo matemático de Python y NumPy, sin embargo, use Iniciar sesión para el logaritmo natural y log10 para el logaritmo con base 10.

Con pandas, en lugar de tener una fórmula que acceda a dos filas diferentes, usa el cambio método para desplazar los valores una fila hacia abajo. Esto le permite operar en una sola fila para que sus cálculos puedan hacer uso de la vectorización. El cambio acepta un número entero positivo o negativo que desplaza la serie de tiempo hacia arriba o hacia abajo en el número respectivo de filas. Veamos primero como cambio obras:

En [20]: msft\_close.cabeza()

Fuera [20]:	Adj Cerrar
Fecha	
1986-03-13 21: 00: 00 + 00: 00	0.062205
1986-03-14 21: 00: 00 + 00: 00	0.064427
1986-03-17 21: 00: 00 + 00: 00	0.065537

```
1986-03-18 21: 00: 00 + 00: 00      0.063871
1986-03-19 21: 00: 00 + 00: 00      0.062760
```

En [21]: `msft_close.cambio(1).cabeza()`

Fuera [21]: Adj Cerrar

Fecha	Adj Cerrar
1986-03-13 21: 00: 00 + 00: 00	Yaya
1986-03-14 21: 00: 00 + 00: 00	0.062205
1986-03-17 21: 00: 00 + 00: 00	0.064427
1986-03-18 21:00 : 00 + 00: 00	0.065537
1986-03-19 21: 00: 00 + 00: 00	0.063871

Ahora puede escribir una fórmula única basada en vectores que sea fácil de leer y comprender.

Para obtener el logaritmo natural, use NumPyIniciar sesión ufunc, que se aplica a cada elemento. Entonces podemos trazar un histograma (verFigura 6-3):

```
En [22]: devoluciones = notario público.Iniciar sesión(msft_close / msft_close.cambio(1))
devoluciones = devoluciones.rebautizar(columnas={"Adj cerrar": "devoluciones"})
devoluciones.cabeza()
```

Fuera [22]: devoluciones

Fecha	devoluciones
1986-03-13 21: 00: 00 + 00: 00	1986-03-Yaya
21: 00: 00 + 00: 00	1986-03-17 2100035097
00: 00 1986-03-18 21:00 : 00 + 00: 00	0000017082
-0.025749 1986-03-19 21: 00: 00 + 00: 00	-0.025749
-0.017547	-0.017547

```
En [23]: # Trazar un histograma con los retornos de registros diarios
devoluciones.trama.hist()
```

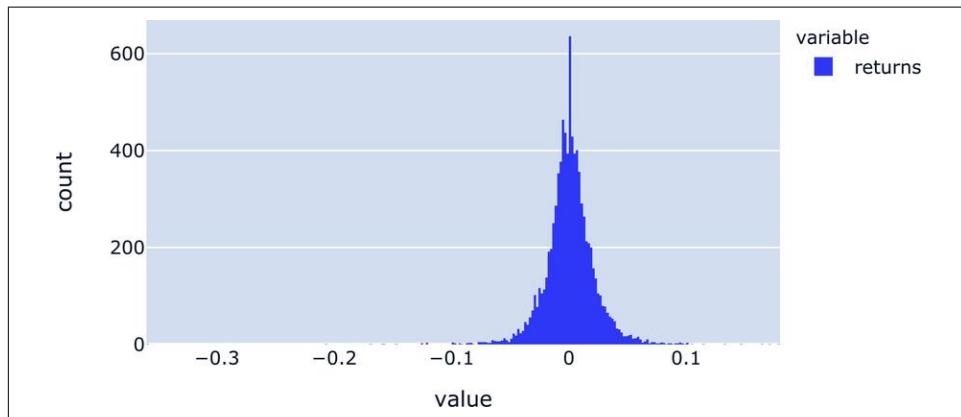


Figura 6-3. Gráfico de histograma

Llegar `devoluciones simples` en su lugar, use el sistema integrado de pandas `pct_change` método. De forma predeterminada, calcula el cambio porcentual de la fila anterior, que también es la definición de rendimientos simples:

```
En [24]: simple_rets = msft_close.pct_change()  
simple_rets = simple_rets.rebautizar(columnas={"Adj cerrar": "rets simples"})  
simple_rets.cabeza()
```

Fuera [24]: rets simples

Fecha	Yaya
1986-03-13 21: 00: 00 + 00: 00	0.035721
1986-03-14 21: 00: 00 + 00: 00	0.017229
1986-03-17 21: 00: 00 + 00: 00	- 0.025421
1986-03-18 21:00 : 00 + 00: 00	- 0.017394

Hasta ahora, hemos analizado solo las acciones de Microsoft. En la siguiente sección, vamos a cargar más series de tiempo para que podamos echar un vistazo a otros métodos de DataFrame que requieren múltiples series de tiempo.

## Rebasamiento y correlación

Las cosas se ponen un poco más interesantes cuando trabajamos con más de una serie temporal. Carguemos algunos precios de cierre ajustados adicionales para Amazon (AMZN), Google (GOOGL) y Apple (AAPL), también descargado de Yahoo! Finanzas:

```
En [25]: partes = [] # Lista para recopilar DataFrames individuales  
por corazón en ["AAPL", "AMZN", "GOOGL", "MSFT"]:  
    # "usecols" nos permite leer solo en la fecha y cierre adj.adj_close =  
    pd.read_csv(F"csv / {ticker} .csv",  
                index_col="Fecha", parse_dates=["Fecha"],  
                usecols=["Fecha", "Adj cerrar"])  
    # Cambie el nombre de la columna al símbolo de cotización  
    adj_close = adj_close.rebautizar(columnas={"Adj cerrar": corazón})  
    # Agregue el DataFrame de la acción a la lista de piezas  
    partes.adjuntar(adj_close)
```

```
En [26]: # Combine los 4 DataFrames en un solo DataFrame  
adj_close = pd.concat(partes, eje=1)  
adj_close
```

Fuera [26]: AAPL AMZN GOOGL MSFT

Fecha	AAPL	AMZN	GOOGL	MSFT
1980-12-12	0.405683	Yaya	Yaya	Yaya
1980-12-15	0.384517	Yaya	Yaya	Yaya
1980-12-16	0.356296	Yaya	Yaya	Yaya
1980-12-17	0.365115	Yaya	Yaya	Yaya
1980-12-18	0.375698	Yaya	Yaya	Yaya
...	...	...	...	...
2020-05-22	318.890015	2436.879883	1413.239990	183.509995
2020-05-26	316.730011	2421.860107	1421.369995	181.570007
2020-05-27	318.109985	2410.389893	1420.280029	181.809998
2020-05-28	318.250000	2401.100098	2020	1418.23
				2433.5907
				Yaya
				Yaya

[9950 filas x 4 columnas]

¿Viste el poder de concat? pandas ha alineado automáticamente las series de tiempo individuales a lo largo de las fechas. Es por eso que obtienes Yaya valores para aquellas acciones que no se remontan tan lejos como Apple. Y desde MSFT tiene Yaya valores en las fechas más recientes, es posible que haya adivinado que descargué *MSFT.csv* dos días antes que los demás. Alinear series de tiempo por fecha es una operación típica que es muy engorrosa de hacer con Excel y, por lo tanto, también muy propensa a errores. Eliminar todas las filas que contienen valores faltantes asegurará que todas las acciones tengan la misma cantidad de puntos de datos:

En [27]: `adj_close = adj_close.dropna()  
adj_close.info()`

```
<class 'pandas.core.frame.DataFrame'> DatetimeIndex: 3970  
entradas, 2004-08-19 a 2020-05-27 Columnas de datos (total 4  
columnas):  
# Columna Tipo de recuento no nulo  
-----0  
AAPL    3970 no nulo    float64  
1   AMZN    3970 no nulo    float64  
2   GOOGL    3970 no nulo    float64  
3   MSFT    3970 no nulo    float64  
dtipos: float64 (4)  
uso de memoria: 155,1 KB
```

Ahora reemplazemos los precios para que todas las series de tiempo comiencen en 100. Esto nos permite comparar su desempeño relativo en un gráfico; ver [Figura 6-4](#). Para reajustar una serie de tiempo, divida cada valor por su valor inicial y multiplique por 100, la nueva base. Si hiciera esto en Excel, normalmente escribiría una fórmula con una combinación de referencias de celda absolutas y relativas, luego copiaría la fórmula para cada fila y cada serie de tiempo. En pandas, gracias a la vectorización y la difusión, se trata de una única fórmula:

En [28]: # Utilice una muestra de junio de 2019 a mayo de 2020  
`adj_close_sample = adj_close.loc["2019-06":"2020-05",:]  
rebased_prices =  
adj_close_sample / adj_close_sample.iloc[0,:] * 100`[rebased\\_prices.cabeza\(2\)](#)

Fuera [28]:	AAPL	AMZN	GOOGL	MSFT
Fecha				
2019-06-03	100.000000	100.000000	100.000000	100.000000
2019-06-04	103.658406	102.178197	101.51626	102.770372

En [29]: `rebased_prices.trauma()`



Figura 6-4. Serie de tiempo reformulada

Para ver cuán independientes son los rendimientos de las diferentes acciones, eche un vistazo a sus correlaciones usando el corr método. Desafortunadamente, pandas no proporciona un tipo de gráfico incorporado para visualizar la matriz de correlación como un mapa de calor, por lo que debemos usar Plotly directamente a través de suplotly.express interfaz (ver Figura 6-5):

```
En [30]: # Correlación de devoluciones de registros diarios
devoluciones = notario.público.Iniciar sesión(adj_close / adj_close.cambio(1))
devoluciones.corr()
```

```
Fuera [30]:   AAPL      AMZN      GOOGL      MSFT
AAPL  1.000000 0.424910 0.503497 0.486065
AMZN  0.424910 1.000000 0.486690 0.485725
GOOGL 0.503497 0.486690 1.000000 0.525645
MSFT  0.486065 0.485725 0.525645 1.000000
```

En [31]: importar plotly.express como px

```
En [32]: higo = px.imshow(devoluciones.corr(),
                           X=adj_close.columnas,y=
                           adj_close.columnas,
                           color_continuous_scale=lista(
                               invertido(px.colores.secuencial.RdBu)),
                           zmin=-1, zmax=1)
higo.show()
```

Si quieres entender como imshow funciona en detalle, eche un vistazo a la [Documentos de la API Plotly Express](#).

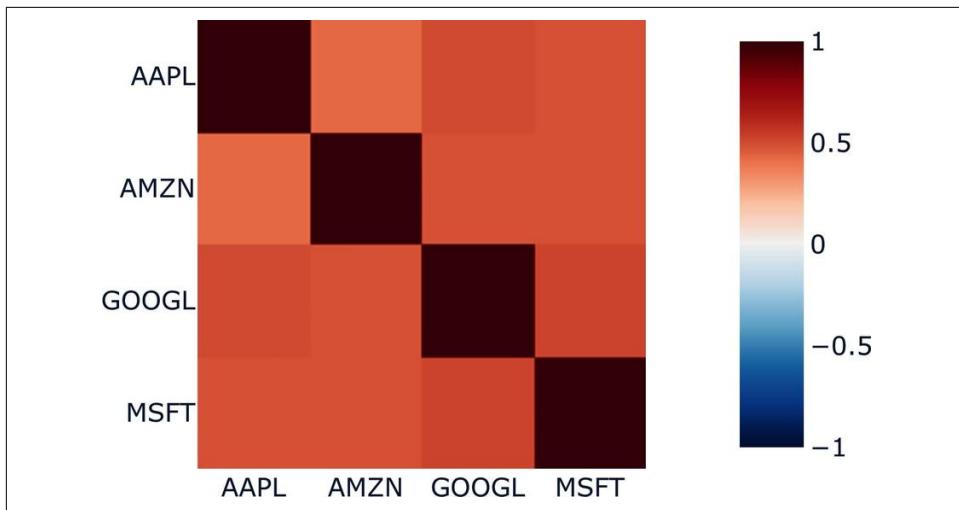


Figura 6-5. Mapa de calor de correlación

En este punto, ya hemos aprendido bastantes cosas sobre las series de tiempo, incluido cómo combinarlas y limpiarlas y cómo calcular retornos y correlaciones. Pero, ¿qué pasa si decide que los rendimientos diarios no son una buena base para su análisis y desea rendimientos mensuales? Cómo se cambia la frecuencia de los datos de series de tiempo es el tema de la siguiente sección.

## Remuestreo

Una tarea regular con series de tiempo es *hasta-* y *submuestreo*. Submuestreo significa que la serie de tiempo se convierte en una con una frecuencia más alta, y un submuestreo significa que se convierte en una con una frecuencia más baja. En las hojas de datos financieros, a menudo muestra el rendimiento mensual o trimestral, por ejemplo. Para convertir la serie de tiempo diaria en una mensual, use el remuestrear método que acepta una cadena de frecuencia como METRO por *fin de mes calendario* o BM por *fin de mes laborable*. Puede encontrar una lista de todas las cadenas de frecuencia en el [pandas docs](#). Similar a como agrupar por funciona, luego encadena un método que define *cómo* estás remuestreando. estoy usando último para tomar siempre la última observación de ese mes:

```
En [33]: fin_de_mes = adj_close.remuestrear("METRO").ultimo()
          fin_de_mes.cabeza()
```

Fuera [33]:	AAPL	AMZN	GOOGL	MSFT
Fecha				
2004-08-31	2.132708	38.139999	51.236237	17.673630
2004-09-30	2.396127	40.860001	64.864868	17.900215
2004-10-31	3.240182	34.130001	95.415413	18.107374
2004-11-30	4.146072	39.680000	91.081078	19.344421
2004-12-31	449.29982.2			

En lugar de último, puede elegir cualquier otro método que funcione en agrupar por, igual que suma o significar. También hay ohlc, que devuelve convenientemente los valores de apertura, máximo, mínimo y cierre durante ese período. Esto puede servir como fuente para crear los gráficos de velas típicos que se utilizan a menudo con los precios de las acciones.

Si esa serie de tiempo de fin de mes fuera todo lo que tiene y necesita producir una serie de tiempo semanal a partir de ella, debe aumentar su serie de tiempo. Mediante el usoasfreq, le está diciendo a los pandas que no apliquen ninguna transformación y, por lo tanto, verá la mayoría de los valores que se muestran Yaya. Si quieras *relleno hacia adelante* el último valor conocido en su lugar, utilice el llenar método:

En [34]: fin de mes.remuestrear("D").asfreq().cabeza() # Sin transformación

Fuera [34]: AAPL AMZN GOOGL MSFT  
Fecha  
2004-08-31 2.132708 38.139999 51.236237 17.67363  
2004-09-01 Yaya Yaya Yaya Yaya  
2004-09-02 Yaya Yaya Yaya Yaya  
2004-09-03 Yaya Yaya Yaya Yaya  
2004-09-04 Yaya Yaya Yaya Yaya

En [35]: fin de mes.remuestrear("W-VI").llenar().cabeza() # Relleno hacia adelante

Fuera [35]: AAPL AMZN GOOGL MSFT  
Fecha  
2004-09-03 2.132708 38.139999 51.236237 17.673630  
2004-09-10 2.132708 38.139999 51.236237 17.673630  
2004-09-17 2.132708 38.139999 51.236237 17.673630  
2004-09-24 2.132708 38.139999 51.236237 17.673630  
2004-10-01 2.396007.82 40.81548

La reducción de la resolución de los datos es una forma de suavizar una serie de tiempo. Calcular estadísticas en una ventana móvil es otra forma, como veremos a continuación.

## Ventanas enrollables

Cuando calcula estadísticas de series de tiempo, a menudo desea una estadística móvil como la *media móvil*. El promedio móvil mira un subconjunto de la serie de tiempo (digamos 25 días) y toma la media de este subconjunto antes de adelantar la ventana un día. Esto dará como resultado una nueva serie de tiempo que es más suave y menos propensa a valores atípicos. Si le gusta el comercio algorítmico, puede estar mirando la intersección del promedio móvil con el precio de las acciones y tomar esto (o alguna variación de él) como una señal comercial. Los DataFrames tienen unlaminación método, que acepta el número de observaciones como argumento. Luego lo encadena con el método estadístico que desea utilizar - en el caso de la media móvil, es el significar. Mirando Figura 6-6, puede comparar fácilmente la serie de tiempo original con la media móvil suavizada:

En [36]: # Trace la media móvil de MSFT con datos de 2019  
msft19 = msft.loc["2019", ["Adj cerrar"]].Copiar()

```
# Agregue el promedio móvil de 25 días como una nueva columna al DataFrame msft19.loc  
[, "Promedio de 25 días"] = msft19["Adj cerrar"].laminación(25).significar()msft19.trama()
```



Figura 6-6. Gráfico de media móvil

En lugar de significar, puede utilizar muchas otras medidas estadísticas, incluidas recuento, suma, mediana, mínimo, máximo, estándar (desviación estándar), o var (diferencia).

En este punto, hemos visto la funcionalidad más importante de los pandas. Sin embargo, es igualmente importante comprender dónde los pandas tienen sus límites, aunque todavía puedan estar muy lejos en este momento.

## Limitaciones con pandas

Cuando sus DataFrames comienzan a crecer, es una buena idea conocer el límite superior de lo que puede contener un DataFrame. A diferencia de Excel, donde tiene un límite estricto de aproximadamente un millón de filas y 12.000 columnas por hoja, pandas solo tiene un límite flexible: todos los datos deben caber en la memoria disponible de su máquina. Si ese no es el caso, puede haber algunas soluciones fáciles: solo cargue las columnas de su conjunto de datos que necesite o elimine los resultados intermedios para liberar algo de memoria. Si eso no ayuda, hay bastantes proyectos que les resultarán familiares a los usuarios de pandas, pero que funcionan con big data. Uno de los proyectos, [Dask](#), funciona sobre NumPy y pandas y le permite trabajar con grandes conjuntos de datos dividiéndolos en múltiples Pandas DataFrames y distribuyendo la carga de trabajo en múltiples núcleos de CPU o máquinas. Otros proyectos de big data que funcionan con algún tipo de DataFrame son [Modin](#), [Koalas](#), [Vaex](#), [PySpark](#), [cuDF](#), [Ibis](#), y [PyArrow](#). Hablaremos brevemente de Modin en el próximo capítulo, pero aparte de eso, esto no es algo que vayamos a explorar más a fondo en este libro.

## Conclusión

El análisis de series de tiempo es el área en la que creo que Excel se ha quedado más rezagado, por lo que después de leer este capítulo, probablemente comprenderá por qué los pandas tienen un éxito tan grande en las finanzas, una industria que depende en gran medida de las series de tiempo. Hemos visto lo fácil que es trabajar con zonas horarias, volver a muestrear series de tiempo o producir matrices de correlación, una funcionalidad que no es compatible con Excel o requiere complicadas soluciones.

Sin embargo, saber cómo usar pandas no significa que tenga que deshacerse de Excel, ya que los dos mundos pueden funcionar muy bien juntos: pandas DataFrames es una excelente manera de transferir datos de un mundo a otro, como veremos en la siguiente parte, que trata sobre la lectura y escritura de archivos de Excel de manera que se omita por completo la aplicación de Excel. Esto es muy útil, ya que significa que puede manipular archivos de Excel con Python en todos los sistemas operativos compatibles con Python, incluido Linux. Para comenzar este diario, el siguiente capítulo le mostrará cómo se pueden usar los pandas para automatizar tediosos procesos manuales como la agregación de archivos de Excel en informes resumidos.



## **Leer y escribir archivos de Excel Sin Excel**



## **Manipulación de archivos de Excel con pandas**

Después de seis capítulos de intensas introducciones a herramientas, Python y pandas, te daré un descanso y comenzaré este capítulo con un caso práctico que te permitirá poner en práctica tus habilidades recién adquiridas: con solo diez líneas de código pandas, Consolidará docenas de archivos de Excel en un informe de Excel, listo para ser enviado a sus gerentes. Despues del estudio de caso, le daré una introducción más detallada a las herramientas que ofrece pandas para trabajar con archivos de Excel:read\_excel función y la Archivo Excel clase de lectura, y el para sobresalir método y el ExcelWriter clase para escribir archivos de Excel. pandas no depende de la aplicación de Excel para leer y escribir archivos de Excel, lo que significa que todos los ejemplos de código de este capítulo se ejecutan en todos los lugares donde se ejecuta Python, incluido Linux.

### **Estudio de caso: Informes de Excel**

Este estudio de caso está inspirado en algunos proyectos de informes del mundo real en los que estuve involucrado durante los últimos años. A pesar de que los proyectos se llevaron a cabo en industrias completamente diferentes, incluidas las telecomunicaciones, el marketing digital y las finanzas, todavía eran notablemente similares: el punto de partida suele ser un directorio con archivos de Excel que deben procesarse en un informe de Excel, a menudo de forma mensual, , semanalmente o diariamente. En el repositorio complementario, en *ellos datos de ventas* directorio, encontrará archivos de Excel con transacciones de ventas ficticias para un proveedor de telecomunicaciones que vende diferentes planes (Bronce, Plata, Oro) en algunas tiendas de los Estados Unidos. Por cada mes, hay dos archivos, uno en el *nuevo* subcarpeta para nuevos contratos y una en el *existentes* subcarpeta para clientes existentes. Como los informes provienen de diferentes sistemas, vienen en diferentes formatos: los nuevos clientes se entregan como *xlsx* archivos, mientras que los clientes existentes llegan a los *xls* formato. Cada uno de los archivos tiene hasta 10,000 transacciones, y nuestro objetivo es producir un informe de Excel que muestre las ventas totales por

tienda y mes. Para empezar, echemos un vistazo a *january.xlsx* archivo de la *nuevo* subcarpeta en [Figura 7-1](#).

	A	B	C	D	E	F	G
1	transaction_id	store	status	transaction_date	plan	contract_type	amount
2	abfbddd6d	Chicago	ACTIVE	1/1/2019	Silver	NEW	14.25
3	136a9997	San Francisco	ACTIVE	1/1/2019	Gold	NEW	19.35
4	c6688f32	San Francisco	ACTIVE	1/1/2019	Bronze	NEW	12.2
5	6ef349c1	Chicago	ACTIVE	1/1/2019	Gold	NEW	19.35
6	22066f29	San Francisco	ACTIVE	1/1/2019	Silver	NEW	14.25

Figura 7-1. Las primeras filas de *January.xlsx*

Los archivos de Excel en el *existente* la subcarpeta se ve prácticamente igual, excepto que les falta el estado columna y se almacenan en el legado *xls* formato. Como primer paso, leamos las nuevas transacciones de enero con pandas 'read\_excel función:

En [1]: importar pandas como pd

En [2]: df = pd.read\_excel("sales\_data / new / January.xlsx")  
df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9493 entradas, 0 a 9492
columnas de datos (total 7 columnas):
 # Columna          Recuento no nulo   Dtype
-----  -----
 0  ID de transacción    9493 no nulo    objeto
 1  Tienda            9493 no nulo    objeto
 2  estado             9493 no nulo    objeto
 3  Fecha de Transacción  9493 no nulo  datetime64 [ns]
 4  plan              9493 no nulo    objeto
 5  tipo de contrato  9493 no nulo    objeto
 6  Monto             9493 no nulo  float64
dtypes: datetime64 [ns] (1), float64 (1), object (5) uso
de memoria: 519,3+ KB
```



### La función read\_excel con Python 3.9

Esta es la misma advertencia que en [Capítulo 5](#): si estas corriendo pd.read\_excel con Python 3.9 o superior, asegúrese de usar al menos pandas 1.2 o obtendrá un error al leer *xlsx* archivos.

Como puede ver, pandas ha reconocido correctamente los tipos de datos de todas las columnas, incluido el formato de fecha de Fecha de Transacción. Esto nos permite trabajar con los datos sin más preparación. Como esta muestra es deliberadamente simple, podemos continuar con la creación de un breve script llamados *sales\_report\_pandas.py* como se muestra en [Ejemplo 7-1](#). Este script leerá todos los archivos de Excel de ambos directorios, agregará los datos y

escriba la tabla de resumen en un nuevo archivo de Excel. Use VS Code para escribir el script usted mismo o ábralo desde el repositorio complementario. Para un repaso sobre cómo crear o abrir archivos en VS Code, eche otro vistazo a [Capítulo 2](#). Si lo crea usted mismo, asegúrese de colocarlo junto a los datos de ventas carpeta: esto le permitirá ejecutar el script sin tener que ajustar ninguna ruta de archivo.

### Ejemplo 7-1. sales\_report\_pandas.py

```
de pathlib importar Sendero
```

```
importar pandas como pd
```

```
# Directorio de este archivo
this_dir = Sendero(__expediente__).resolver().padre ①

# Leer en todos los archivos de Excel de todas las subcarpetas de sales_data
partes = []
por sendero en (this_dir / "los datos de ventas").rglob("* .xls *"):
    ②
        impresión(F"Leyendo {ruta.nombre}")
        parte = pd.read_excel(sendero, index_col="ID de transacción")
        partes.adjuntar(parte)

# Combine los DataFrames de cada archivo en un solo DataFrame
# pandas se encarga de alinear correctamente las columnas
df = pd.concat(partes)

# Pivote cada tienda en una columna y resuma todas las transacciones por fecha
pivot = pd.tabla dinámica(df,
                           índice="Fecha de Transacción", columnas="Tienda",
                           valores="Monto", aggfunc="suma")

# Vuelva a muestrear al final del mes y asigne un nombre de índice
resumen = pivot.remuestrear("METRO").suma()resumen.índice.
nombre = "Mes"

# Escribe un informe de resumen en un archivo de Excel
resumen.para sobresalir(this_dir / "sales_report_pandas.xlsx")
```

- ① Hasta este capítulo, estaba usando cadenas para especificar rutas de archivo. Usando el `Sendero` clase de la biblioteca estándar `pathlib` en su lugar, obtiene acceso a un poderoso conjunto de herramientas: los objetos de ruta le permiten construir fácilmente rutas mediante la concatenación de partes individuales a través de barras diagonales, como se hace cuatro líneas a continuación con `this_dir / "sales_data"`. Estas rutas funcionan en todas las plataformas y le permiten aplicar filtros como `rglob` como se explica en el siguiente punto. `__expediente__` se resuelve en la ruta del archivo de código fuente cuando lo ejecuta, utilizando su padre le dará por tanto el nombre del directorio de este archivo. `resolver` método que usamos antes de llamar

padre convierte el camino en un camino absoluto. Si en su lugar ejecutara esto desde un cuaderno Jupyter, tendría que reemplazar esta línea con `this_dir = Ruta ("."). resolver ()`, con el punto que representa el directorio actual. En la mayoría de los casos, las funciones y las clases que aceptan una ruta en forma de cadena también aceptan un objeto de ruta.

- ② La forma más fácil de leer en todos los archivos de Excel de forma recursiva desde un directorio determinado es utilizar el `rglob` método del objeto de ruta. `glob` es la abreviatura de *globoso*, que se refiere a la expansión del nombre de ruta mediante comodines. Los ? El comodín representa exactamente un carácter, mientras que \* representa cualquier número de caracteres (incluido cero). `losr` en `rglob` medio *recursivo globbing*, es decir, buscará archivos coincidentes en todos los subdirectorios; en consecuencia, `glob` ignoraría los subdirectorios. Utilizando \*. xls \* ya que la expresión global se asegura de que se encuentren los archivos de Excel antiguo y nuevo, ya que coincide con ambos.xls y .xlsx. Por lo general, es una buena idea mejorar ligeramente la expresión de esta manera: [! ~ \$] \*.xls \*. Esto ignora los archivos temporales de Excel (su nombre de archivo comienza con ~ \$). Para obtener más información sobre cómo usar globbing en Python, consulte la [Documentación de Python](#).

Ejecute el script, por ejemplo, haciendo clic en el botón Ejecutar archivo en la parte superior derecha de VS Code. El script tardará un momento en completarse y, una vez hecho, el libro de `Excelsales_report_pandas.xlsx` aparecerá en el mismo directorio que el script. El contenido de Sheet1 debería verse como en [Figura 7-2](#). Ese es un resultado bastante impresionante para solo diez líneas de código, ¡incluso si necesita ajustar el ancho de la primera columna para poder ver las fechas!

	A	B	C	D	E	F	G	
1	Month	Boston	Chicago	Las Vegas	New York	an Francisco	washington DC	
2	#####	21784.1	51187.7	23012.75	49872.85	58629.85	14057.6	
3	#####	21454.9	52330.85	25493.1	46669.85	55218.65	15235.4	
4	#####	20043	48897.25	23451.1	41572.25	52712.95	14177.05	
5	#####	18791.05	47396.35	22710.15	41714.3	49324.65	13339.15	
6	#####	18036.75	45117.05	21526.55	40610.4	47759.6	13147.1	
7	#####	21556.25	49460.45	21985.05	47265.65	53462.4	14284.3	
8	#####	19853	47993.8	23444.3	40408.3	50181.6	14161.5	
9	#####	22332.9	50838.9	24927.65	45396.85	55336.35	16127.05	
10	#####	19924.5	49096.25	24410.7	42830.6	49931.45	14994.4	
11	#####	16550.95	42543.8	22827.5	34090.05	44311.65	12846.7	
12	#####	21312.9	52011.6	24860.25	46959.85	55056.45	14057.6	
13	#####	19722.6	49355.1	24535.75	42364.35	50933.45	14702.15	

Figura 7-2. `sales_report_pandas.xlsx` (tal cual, sin ajustar ningún ancho de columna)

Para casos simples como este, pandas ofrece una solución realmente fácil para trabajar con archivos de Excel. Sin embargo, podemos hacerlo mucho mejor: después de todo, un título, algo de formato (incluido el ancho de columna y un número constante de decimales) y un gráfico no estaría de más. Eso es exactamente de lo que nos ocuparemos en el próximo capítulo usando directamente las bibliotecas de escritor que los pandas usan debajo del capó. Sin embargo, antes de llegar allí, echemos un vistazo más detallado a cómo podemos leer y escribir archivos de Excel con pandas.

## Leer y escribir archivos de Excel con pandas

El estudio de caso estaba usando `read_excel` y para sobresalir con sus argumentos predeterminados para simplificar las cosas. En esta sección, le mostraré los argumentos y las opciones más utilizados al leer y escribir archivos de Excel con pandas. Empezaremos con la `read_excel` función y la `ExcelWriter` clase antes de mirar el para sobresalir método y la `ExcelWriter` clase. En el camino, también presentaré Python con declaración.

### La función `read_excel` y la clase `ExcelFile`

El estudio de caso utilizó libros de Excel donde los datos estaban convenientemente en la celda A1 de la primera hoja. En realidad, sus archivos de Excel probablemente no estén tan bien organizados. En este caso, pandas ofrece parámetros para afinar el proceso de lectura. Para las próximas muestras, usaremos el `stores.xlsx` archivo que encontrarás en el `SG` carpeta del repositorio complementario. La primera hoja se muestra en Figura 7-3.

A	B	C	D	E	F
1					
2	Store	Employees	Manager	Since	Flagship
3	New York		10 Sarah	7/20/2018	FALSE
4	San Francisco		12 Neriah	11/2/2019	MISSING
5	Chicago		4 Katelin	1/31/2020	
6	Boston		5 Georgiana	4/1/2017	TRUE
7	Washington DC		3 Evan		FALSE
8	Las Vegas		11 Paul	1/6/2020	FALSE
9					

Figura 7-3. La primera hoja de `stores.xlsx`

Usando los parámetros `sheet_name`, `skiprows`, y `usecols`, podemos decirle a los pandas sobre el rango de celdas en el que queremos leer. Como de costumbre, es una buena idea echar un vistazo a los tipos de datos del DataFrame devuelto ejecutando el `info` método:

```
En [3]: df = pd.read_excel("xl / stores.xlsx",
                           sheet_name="2019", saltos=1, usecols="B: F")
df
```

```
Fuera [3]:          Tienda    Empleados    Gerente      Desde buque insignia
              0 Nueva York     10   Sara 2018-07-20      Falso
              1 San Francisco    12  Nerías 2019-11-02  DESAPARECIDO
              2 Chicago            4   Katelin 2020-01-31      Yaya
              3 Bostón              5  Georgiana 01-04-2017  Cierto
              4 Washington DC      3   Evan NaT        Falso
              5 Las Vegas           11  Pablo 2020-01-06      Falso
```

En [4]: df.info()

```
<clase 'pandas.core.frame.DataFrame'>
RangeIndex: 6 entradas, 0 a 5
Columnas de datos (5 columnas en total):
 # Columna    Recuento no nulo    Dtype
-----  -----
 0 Tienda       6 no nulo        objeto
 1 Empleados    6 no nulo        int64
 2 Gerente      6 no nulo        objeto
 3 Ya que       5 no nulo        datetime64 [ns]
 4 Buque insignia 5 no nulo        objeto
dtypes: datetime64 [ns] (1), int64 (1), object (3) uso
de memoria: 368.0+ bytes
```

Todo se ve bien excepto por el Buque insignia columna: su tipo de datos debe ser bool en vez de objeto. Para solucionar esto, podemos proporcionar una función de conversión que se ocupa de las celdas ofensivas en esa columna (en lugar de escribir el fix\_missing función, también podríamos haber proporcionado una expresión lambda en su lugar):

En [5]: def fix\_missing(X):

```
    regreso Falso si X en ["", "DESAPARECIDO"] demás X
```

En [6]: df = pd.read\_excel("xl / stores.xlsx",

```
                           sheet_name="2019", saltos=1, usecols="B: F",
                           convertidores={"Buque insignia": fix_missing})
```

df

```
Fuera [6]:          Tienda    Empleados    Gerente      Ya que    Buque insignia
              0 Nueva York     10   Sara 2018-07-20      Falso
              1 San Francisco    12  Nerías 2019-11-02      Falso
              2 Chicago            4   Katelin 2020-01-31      Falso
              3 Bostón              5  Georgiana 01-04-2017  Cierto
              4 Washington DC      3   Evan NaT        Falso
              5 Las Vegas           11  Pablo 2020-01-06      Falso
```

En [7]: # La columna Flagship ahora tiene Dtype "bool"

df.info()

```
<clase 'pandas.core.frame.DataFrame'>
```

RangeIndex: 6 entradas, 0 a 5

Columnas de datos (5 columnas en total):

# Columna	Recuento no nulo	Dtype
0 Tienda	6 no nulo	objeto
1 Empleados	6 no nulo	int64
2 Gerente	6 no nulo	objeto

```

3    Ya que      5 no nulo      datetime64 [ns]
4    Buque insignia 6 no nulo      bool
dtypes: bool (1), datetime64 [ns] (1), int64 (1), object (2) uso de
memoria: 326.0+ bytes

```

los `read_excel` La función también acepta una lista de nombres de hojas. En este caso, devuelve un diccionario con el DataFrame como valor y el nombre de la hoja como clave. Para leer en todas las hojas, deberá proporcionar `sheet_name = Ninguno`. Además, tenga en cuenta la ligera variación de cómo estoy usando `usecols` proporcionando los nombres de las columnas de la tabla:

```

En [8]: hojas = pd.read_excel("xl / stores.xlsx", sheet_name=["2019", "2020"],
                               saltos=1, usecols=["Tienda", "Empleados"])
hojas["2019"].cabeza(2)

```

```

Fuera [8]:          Tienda   Empleados
              0     Nueva York      10
              1   San Francisco      12

```

Si el archivo de origen no tiene encabezados de columna, establezca `encabezado = Ninguno` y proporcionarlos a través de nombres. Tenga en cuenta que `sheet_name` también acepta índices de hoja:

```

En [9]: df = pd.read_excel("xl / stores.xlsx", sheet_name=0,
                           saltos=2, skipfooter=3, usecols="B: C",
                           F", encabezamiento=Ninguno,
                           nombres=["Rama", "Número de empleados", "Is_Flagship"])
df

```

```

Fuera [9]:          Rama   Employee_Count Is_Flagship
              0     Nueva York      10      Falso
              1   San Francisco      12  DESAPARECIDO
              2     Chicago            4        Yaya

```

Manejar Yaya valores, use una combinación de `na_values` y `keep_default_na`. La siguiente muestra le dice a los pandas que solo interpreten las celdas con la palabra DESAPARECIDO como Yaya y nada más:

```

En [10]: df = pd.read_excel("xl / stores.xlsx", sheet_name="2019",
                           saltos=1, usecols="B, C, F", skipfooter=2, na_values=
                           "DESAPARECIDO", keep_default_na=False)
df

```

```

Fuera [10]:          Tienda   Empleados insignia
              0     Nueva York      10      Falso
              1   San Francisco      12        Yaya
              2     Chicago            4
              3     Bostón             5      Cierto

```

pandas ofrece una forma alternativa de leer archivos de Excel utilizando el Archivo Excel clase. Esto marca la diferencia principalmente si desea leer en varias hojas de un archivo en el formato legado `/s`: en este caso, usando Archivo Excel será más rápido, ya que evita que los pandas lean el archivo completo varias veces. Archivo Excel se puede usar como administrador de contexto (ver barra lateral) para que el archivo se cierre correctamente nuevamente.

### Los administradores de contexto y la declaración with

En primer lugar, el con declaración en Python no tiene nada que ver con la Con declaración en VBA: en VBA, se usa para ejecutar una serie de declaraciones en el mismo objeto, mientras que en Python, se usa para administrar recursos como archivos o conexiones de bases de datos. Si desea cargar los últimos datos de ventas para poder analizarlos, es posible que deba abrir un archivo o establecer una conexión a una base de datos. Una vez que haya terminado de leer los datos, se recomienda volver a cerrar el archivo o la conexión lo antes posible. De lo contrario, puede encontrarse con situaciones en las que no pueda abrir otro archivo o no pueda establecer otra conexión a la base de datos; los manejadores de archivos y las conexiones a la base de datos son recursos limitados. Abrir y cerrar un archivo de texto manualmente funciona así (w significa abrir el archivo en escribir modo, que reemplaza el archivo si ya existe):

```
En [11]: F = abierto("salida.txt", "w")
          F.escribir("Algún texto")
          F.cerrar()
```

Ejecutar este código creará un archivo llamado *output.txt* en el mismo directorio que el cuaderno desde el que lo está ejecutando y escriba “algo de texto” en él. Para leer un archivo, usarías r en lugar de w y para adjuntar al final del archivo, use una. Dado que los archivos también se pueden manipular desde fuera de su programa, dicha operación podría fallar. Puede manejar esto usando el mecanismo try / except que presentaré en [Capítulo 11](#). Sin embargo, dado que esta es una operación tan común, Python proporciona la con declaración para facilitar las cosas:

```
En [12]: con abierto("salida.txt", "w") como F:
          F.escribir("Algún texto")
```

Cuando la ejecución de código abandona el cuerpo del con declaración, el archivo se cierra automáticamente, haya o no ocurriendo una excepción. Esto garantiza que los recursos se limpian adecuadamente. Objetos que soportan el con se llaman declaraciones *administradores de contexto*; esto incluye el Archivo Excel y ExcelWriter objetos en este capítulo, así como los objetos de conexión a la base de datos que veremos en [Capítulo 11](#).

Veamos el Archivo Excel clase en acción:

```
En [13]: con pd Archivo Excel("xl / stores.xls") como F:
          df1 = pd.read_excel(F, "2019", saltos=1, usecols="B: F", nrows=2)
          df2 = pd.read_excel(F, "2020", saltos=1, usecols="B: F", nrows=2)
```

df1

Fuera [13]:	Tienda	Gerente de empleados	Desde buque insignia
0	Nueva York	10 Sara 2018-07-20	Falso
1	San Francisco	12 Nerías 2019-11-02	DESAPARECIDO

Archivo Excel también le da acceso a los nombres de todas las hojas:

```
En [14]: historias = pd Archivo Excel("xl / stores.xlsx")
historias.sheet_names
```

Fuera [14]: ['2019', '2020', '2019-2020']

Finalmente, pandas le permite leer archivos de Excel desde una URL, de manera similar a como lo hicimos con archivos CSV en [Capítulo 5](#). Leámoslo directamente del repositorio complementario:

```
En [15]: url = ("https://raw.githubusercontent.com/fzumstein/"
               "python-for-excel / 1st-edition / xl / stores.xlsx")
pd.read_excel(url, saltos=1, usecols="SER", nrows=2)
```

Fuera [15]:

	Tienda	Gerente de empleados	Ya que
0	Nueva York	10	Sara 2018-07-20
1	San Francisco	12	Nerías 2019-11-02



#### Lectura de archivos xlsb a través de pandas

Si usa pandas con una versión inferior a 1.3, lea `xlsb` archivos requiere que especifique explícitamente el motor en la función `read_excel` o Archivo Excel clase:

```
pd.read_excel("xl / stores.xlsb", motor="pyxlsb")
```

Esto requiere que se instale el paquete `pyxlsb`, ya que no es parte de Anaconda; llegaremos a eso y a los otros motores en el próximo capítulo.

Para resumir, [Tabla 7-1](#) muestra los más utilizados `read_excel` parámetros. Encontrará la lista completa en el [documentos oficiales](#).

Tabla 7-1. Parámetros seleccionados para `read_excel`

Parámetro	Descripción
<code>sheet_name</code>	En lugar de proporcionar un nombre de hoja, también puede proporcionar el índice de la hoja (basado en cero), por ejemplo, <code>sheet_name = 0</code> . Si pones <code>sheet_name = Ninguno</code> , los pandas leerán todo el libro de trabajo y devolverán un diccionario en forma de {"nombre de hoja": df}. Para leer una selección de hojas, proporcione una lista con nombres de hojas o índices.
<code>saltos</code>	Esto le permite omitir el número de filas indicado.
<code>usecols</code>	Si el archivo de Excel incluye los nombres de los encabezados de las columnas, proporcione los en una lista para seleccionar las columnas, por ejemplo, ["Tienda", "Empleados"]. Alternativamente, también puede ser una lista de índices de columna, por ejemplo, [1, 2], o una cadena (no una lista) de nombres de columnas de Excel, incluidos rangos, p. ej., "B: D, G". También puede proporcionar una función: como ejemplo, para incluir solo las columnas que comienzan con Gerente, usar: <code>usecols = lambda x: x.startswith("Administrador")</code> .
<code>nrows</code>	Número de filas que desea leer.
<code>index_col</code>	Indica qué columna debe ser el índice, acepta un nombre de columna o un índice, por ejemplo, <code>index_col = 0</code> . Si proporciona una lista con varias columnas, se creará un índice jerárquico.

Parámetro	Descripción
encabezamiento	Si pones encabezado = Ninguno, los encabezados enteros predeterminados se asignan excepto si proporciona los nombres deseados a través del nombre parámetro. Si proporciona una lista de índices, se crearán encabezados de columna jerárquicos.
nombres	Proporcione los nombres deseados de sus columnas como lista.
na_values	Pandas interpreta los siguientes valores de celda como Yaya por defecto (presenté Yaya en Capítulo 5): celdas vacías, #NA, NA, nulo, # N / A, N / A, NaN, n / a, -NaN, 1. # IND, nan, # N / AN / A, -1. # QNAN, - nan, NULL, - 1. # IND, <NA>, 1. # QNAN. Si desea agregar uno o más valores a esa lista, proporcínelos a través de na_values.
keep_default_na	Si desea ignorar los valores predeterminados que los pandas interpretan como Yaya, colocar keep_default_na = Falso.
convert_float	Excel almacena todos los números internamente como flotantes y, de forma predeterminada, los pandas transforman los números sin decimales significativos en números enteros. Si desea cambiar ese comportamiento, establezca convert_float = False (esto puede ser un poco más rápido).
convertidores	Le permite proporcionar una función por columna para convertir sus valores. Por ejemplo, para convertir el texto en una determinada columna en mayúsculas, utilice lo siguiente: converters = {"column_name": lambda x: x.upper ()}

Demasiado para leer archivos de Excel con pandas. ¡Ahora cambiemos de lado y aprendamos a escribir archivos de Excel en la siguiente sección!

## El método to\_excel y la clase ExcelWriter

La forma más fácil de escribir un archivo de Excel con pandas es usar un DataFrame para sobresalir método. Le permite especificar en qué celda de la hoja desea escribir el DataFrame. También puede decidir si incluir o no los encabezados de columna y el índice del DataFrame y cómo tratar los tipos de datos comonp.nan y np.inf que no tienen una representación equivalente en Excel. Comencemos creando un DataFrame con diferentes tipos de datos y usemos supara sobresalir método:

```
En [dieciséis]: importar numpy como notario público
importar fecha y hora como dt

En [17]: datos=[[dt.fecha y hora(2020,1,1, 10, 13), 2.222, 1, Cierto],
           [dt.fecha y hora(2020,1,2), notario público.yaya, 2, Falso], [
             dt.fecha y hora(2020,1,2), notario público.inf, 3, Cierto]]
df = pd.Marco de datos(datos=datos,
                       columnas=["Fechas", "Flota", "Enteros", "Booleanos"])
df.índice.nombre="índice"
df
```

Fuera [17]: Fechas Flotantes Enteros Booleanos

índice				
0	01-01-2020 10:13:00	2.222	1	Cierto
1	2020-01-02 00:00:00	Yaya	2	Falso
2	2020-01-02 00:00:00	inf	3	Cierto

En [18]: df.para sobresalir("escrito\_con\_pandas.xlsx", sheet\_name="Producción", startrow=1, startcol=1, índice=Cierto, encabezamiento=Cierto, na\_rep="", inf\_rep="")

Ejecutando el para sobresalir comando creará el archivo de Excel como se muestra en [Figura 7-4](#) (necesitarás hacer una columna C más ancho para ver las fechas correctamente):

	A	B	C	D	E	F
1						
2		index	Dates	FLOATS	Integers	Booleans
3		0	2020-01-01 10:13:00	2.222	1	TRUE
4		1	2020-01-02 00:00:00	<NA>	2	FALSE
5		2	2020-01-02 00:00:00	<INF>	3	TRUE

Figura 7-4. escrito\_con\_pandas.xlsx

Si desea escribir varios DataFrames en la misma hoja o en hojas diferentes, deberá utilizar el ExcelWriter clase. El siguiente ejemplo escribe el mismo DataFrame en dos ubicaciones diferentes en Sheet1 y una vez más en Sheet2:

En [19]: `con pd.ExcelWriter("escrito_con_pandas2.xlsx") como escritor:  
df.para sobresalir(escritor, sheet_name="Hoja1", startrow=1, startcol=1)df.para  
sobresalir(escritor, sheet_name="Hoja1", startrow=10, startcol=1)df.para  
sobresalir(escritor, sheet_name="Hoja2")`

Dado que estamos usando el ExcelWriter class como administrador de contexto, el archivo se escribe automáticamente en el disco cuando sale del administrador de contexto, es decir, cuando se detiene la sangría. De lo contrario, tendrás que llamarWriter.save () explícitamente. Para obtener un resumen de los parámetros más utilizados para sobresalir acepta, echa un vistazo a [Tabla 7-2](#). Encontrará la lista completa de parámetros en el[documentos oficiales](#).

Tabla 7-2. Parámetros seleccionados para to\_excel

Parámetro	Descripción
sheet_name	Nombre de la hoja en la que escribir.
startrow y startcol	startrow es la primera fila donde se escribirá el DataFrame y startcol es la primera columna. Esto usa indexación de base cero, por lo que si desea escribir su DataFrame en la celda B3, use startrow = 2 y startcol = 1.
índice y encabezamiento	Si desea ocultar el índice y / o el encabezado, configúrelos en index = Falso y header = False, respectivamente.
na_rep y inf_rep	Por defecto, np.nan se convertirá en una celda vacía, mientras que np.inf, La representación del infinito de NumPy, se convertirá a la cadena inf. Proporcionar valores le permite cambiar este comportamiento.
freeze_panes	Congele el primer par de filas y columnas proporcionando una tupla: por ejemplo (2, 1) congelará las dos primeras filas y la primera columna.

Como puede ver, leer y escribir archivos simples de Excel con pandas funciona bien. Sin embargo, existen limitaciones, ¡veamos cuáles!

## Limitaciones al usar pandas con archivos de Excel

El uso de la interfaz de pandas para leer y escribir archivos de Excel funciona muy bien para casos simples, pero existen límites:

- Al escribir DataFrames en archivos, no puede incluir un título o un gráfico.
- No hay forma de cambiar el formato predeterminado del encabezado y el índice en Excel.
- Al leer archivos, los pandas transforman automáticamente las celdas con errores como #¡ÁRBITRO! o #NUM! dentro Yaya, haciendo imposible la búsqueda de errores específicos en sus hojas de cálculo.
- Trabajar con archivos grandes de Excel puede requerir configuraciones adicionales que son más fáciles de controlar usando los paquetes de lectura y escritura directamente, como veremos en el próximo capítulo.

## Conclusión

Lo bueno de los pandas es que ofrece una interfaz coherente para trabajar con todos los formatos de archivo de Excel compatibles, ya sea *xls*, *xlsx*, *xlsm*, o *xlsb*. Esto nos facilitó leer un directorio de archivos de Excel, agregar los datos y volcar el resumen en un informe de Excel, en solo diez líneas de código.

pandas, sin embargo, no hace el trabajo pesado por sí mismo: bajo el capó, selecciona un paquete de lector o escritor para hacer el trabajo. En el próximo capítulo, le mostraré qué paquetes de lectores y escritores usan los pandas y cómo los usa directamente o en combinación con los pandas. Esto nos permitirá solucionar las limitaciones que vimos en la sección anterior.

## **Manipulación de archivos de Excel con Reader y paquetes de escritor**

Este capítulo le presenta OpenPyXL, XlsxWriter, pyxlsb, xlrd y xlwt: estos son los paquetes que pueden leer y escribir archivos de Excel y son utilizados por pandas bajo el capó cuando llama al read\_excel o para sobresalir funciones. El uso directo de los paquetes de lectura y escritura le permite crear informes de Excel más complejos, así como ajustar el proceso de lectura. Además, si alguna vez trabaja en un proyecto en el que solo necesita leer y escribir archivos de Excel sin la necesidad del resto de la funcionalidad de pandas, instalar la pila completa de NumPy / pandas probablemente sería una exageración. Comenzaremos este capítulo aprendiendo cuándo usar qué paquete y cómo funciona su sintaxis antes de ver algunos temas avanzados, incluido cómo trabajar con archivos grandes de Excel y cómo combinar pandas con los paquetes de lector y escritor para mejorar el estilo de DataFrames. Para concluir, retomaremos el estudio de caso del principio del último capítulo y mejoraremos el informe de Excel formateando la tabla y agregando un gráfico. Como el ultimo capitulo

### **Los paquetes Reader y Writer**

El panorama de lectores y escritores puede ser un poco abrumador: vamos a ver no menos de seis paquetes en esta sección, ya que casi todos los tipos de archivos de Excel requieren un paquete diferente. El hecho de que cada paquete use una sintaxis diferente que a menudo se desvíe sustancialmente del modelo de objetos de Excel original no lo hace más fácil; diré más sobre el modelo de objetos de Excel en el próximo capítulo. Esto significa que probablemente tendrá que buscar muchos comandos, incluso si es un desarrollador de VBA experimentado. Esta sección comienza con una descripción general de cuándo necesita qué paquete antes de presentar un módulo auxiliar que facilita un poco el trabajo con estos paquetes. Después de eso,

presenta cada uno de los paquetes en un estilo de libro de cocina, donde puede buscar cómo funcionan los comandos más utilizados.

## Cuándo usar qué paquete

Esta sección presenta los siguientes seis paquetes para leer, escribir y editar archivos de Excel:

- [OpenPyXL](#)
- [XlsxWriter](#)
- [pyxlsb](#)
- [xlrd](#)
- [xlwt](#)
- [xlutils](#)

Para comprender qué paquete puede hacer qué, eche un vistazo a [Tabla 8-1](#). Por ejemplo, para leer el *xlsx* formato de archivo, tendrá que usar el paquete OpenPyXL:

*Tabla 8-1. Cuándo usar qué paquete*

Formato de archivo de Excel	Leer	Escribir	Editar
<code>xlsx</code>	OpenPyXL	OpenPyXL, XlsxWriter	OpenPyXL
<code>xlsm</code>	OpenPyXL	OpenPyXL, XlsxWriter	OpenPyXL
<code>xltx, xltm</code>	OpenPyXL	OpenPyXL	OpenPyXL
<code>xlsb</code>	pyxlsb	-	-
<code>xls, xlt</code>	<code>xlrd</code>	<code>xlwt</code>	<code>xlutils</code>

Si quieras escribir *xlsx* o *xlsm* archivos, debe decidir entre OpenPyXL y XlsxWriter. Ambos paquetes cubren una funcionalidad similar, pero cada paquete puede tener algunas características únicas que el otro no tiene. Dado que ambas bibliotecas se están desarrollando activamente, esto está cambiando con el tiempo. Aquí hay una descripción general de alto nivel de dónde se diferencian:

- OpenPyXL puede leer, escribir y editar mientras que XlsxWriter solo puede escribir.
- OpenPyXL facilita la producción de archivos de Excel con macros VBA.
- XlsxWriter está mejor documentado.
- XlsxWriter tiende a ser más rápido que OpenPyXL, pero dependiendo del tamaño del libro de trabajo que esté escribiendo, las diferencias pueden no ser significativas.



### ¿Dónde está xlwings?

Si se pregunta dónde está xlwings en la tabla [Tabla 8-1](#), entonces la respuesta es *en ningún lugar o En todas partes*, dependiendo de su caso de uso: a diferencia de cualquiera de los paquetes de este capítulo, xlwings depende de la aplicación Excel, que a menudo no está disponible, por ejemplo, si necesita ejecutar sus scripts en Linux. Si, por otro lado, está de acuerdo con ejecutar sus scripts en Windows o macOS donde tiene acceso a una instalación de Excel, xlwings puede usarse como una alternativa a todos los paquetes de este capítulo. Dado que la dependencia de Excel es una diferencia fundamental entre xlwings y todos los demás paquetes de Excel, presentaré xlwings en el siguiente capítulo, que comienza [Parte IV](#) de este libro.

pandas usa el paquete de escritura que puede encontrar y si tiene instalados OpenPyXL y XlsxWriter, XlsxWriter es el predeterminado. Si desea elegir qué paquete deben usar los pandas, especifique el motor parámetro en el `read_excel` o para sobresalir funciones o el Archivo Excel y `ExcelWriter` clases, respectivamente. El motor es el nombre del paquete en minúsculas, por lo que para escribir un archivo con OpenPyXL en lugar de XlsxWriter, ejecute lo siguiente:

```
df.to_excel("nombrearchivo.xlsx", motor="openpyxl")
```

Una vez que sepa qué paquete necesita, hay un segundo desafío esperándolo: la mayoría de estos paquetes requieren que escriba bastante código para leer o escribir un rango de celdas, y cada paquete usa una sintaxis diferente. Para facilitarle la vida, creé un módulo de ayuda que presentaré a continuación.

## El módulo `excel.py`

He creado el `excel.py` módulo para hacer su vida más fácil al usar los paquetes de lector y escritor, ya que se ocupa de los siguientes problemas:

### *Cambio de paquete*

Tener que cambiar el paquete de lector o escritor es un escenario relativamente común. Por ejemplo, los archivos de Excel tienden a aumentar de tamaño con el tiempo, lo que muchos usuarios luchan al cambiar el formato de archivo `xls/x` para `xlsb` ya que esto puede reducir sustancialmente el tamaño del archivo. En ese caso, tendrá que cambiar de OpenPyXL a pyxlsb. Esto le obliga a reescribir su código OpenPyXL para reflejar la sintaxis de pyxlsb.

### *Conversión de tipo de datos*

Esto está conectado con el punto anterior: al cambiar de paquete, no solo tiene que ajustar la sintaxis de su código, sino que también debe tener cuidado con los diferentes tipos de datos que estos paquetes devuelven para el mismo contenido de celda. Por ejemplo, OpenPyXL devuelve `Ninguno` para celdas vacías, mientras que xlrd devuelve una cadena vacía.

### Bucle de celda

Los paquetes de lector y escritor son *nivel bajo* paquetes: esto significa que carecen de funciones de conveniencia que le permitirían abordar tareas comunes fácilmente. Por ejemplo, la mayoría de los paquetes requieren que recorras cada celda que vas a leer o escribir.

Encontraras el excel.py módulo en el repositorio complementario y lo usaremos en las próximas secciones, pero como vista previa, aquí está la sintaxis para leer y escribir valores:

```
importar sobresalir  
valores = sobresalir.leer(objeto_hoja, primera_celda="A1", last_cell=Ninguno)  
sobresalir.escribir(objeto_hoja, valores, primera_celda="A1")
```

los leer la función acepta un hoja objeto de uno de los siguientes paquetes: xlrd, OpenPyXL o pyxlsb. También acepta los argumentos opcionales, primera\_celda y last\_cell. Se pueden proporcionar en el A1 notación o como fila-columna-tupla con índices basados en uno de Excel: (1, 1). El valor predeterminado para el primera\_celda es A1 mientras que el valor predeterminado para last\_cell es la esquina inferior derecha del rango utilizado. Por lo tanto, si solo proporciona el hoja objeto, leerá toda la hoja. Los escribir La función funciona de manera similar: espera una hoja objeto de xlwt, OpenPyXL o XlsxWriter junto con los valores como lista anidada y una opción primera\_celda, que marca la esquina superior izquierda de donde se escribirá la lista anidada. Los excel.py módulo también armoniza la conversión del tipo de datos como se muestra en [Tabla 8-2](#).

Tabla 8-2. Conversión de tipo de datos

Representación de Excel	Tipo de datos de Python
Celda vacía	Ninguno
Celda con formato de fecha	datetime.datetime (excepto pyxlsb)
Celda con booleano Celda	bool
con error	str (el mensaje de error)
Cuerda	str
Flotador	flotador o En t

Equipado con el excel.py módulo, ahora estamos listos para sumergirnos en los paquetes: las siguientes cuatro secciones son sobre OpenPyXL, XlsxWriter, pyxlsb y xlrd / xlwt / xlutils. Siguen un estilo de libro de cocina que le permite comenzar rápidamente con cada paquete. En lugar de leerlo secuencialmente, le recomendaría que elija el paquete que necesita según [Tabla 8-1](#), luego salte directamente a la sección correspondiente.



## La declaración con

Usaremos el con declaración en varias ocasiones en este capítulo. Si necesita un repaso, eche un vistazo a la barra lateral “[Los administradores de contexto y la declaración with](#)” en la página 150 en Capítulo 7.

## OpenPyXL

OpenPyXL es el único paquete en esta sección que puede leer y escribir archivos de Excel. Incluso puede usarlo para editar archivos de Excel, aunque solo sean simples. ¡Comencemos por ver cómo funciona la lectura!

### Leer con OpenPyXL

El siguiente código de muestra le muestra cómo realizar tareas comunes cuando usa OpenPyXL para leer archivos de Excel. Para obtener los valores de las celdas, debe abrir el libro de trabajo con `data_only = Verdadero`. El valor predeterminado está activado Falso, que devolvería las fórmulas de las celdas en su lugar:

```
En [1]: importar pandas como pd  
        importar openpyxl  
        importar sobresalir  
        importar fecha y hora como dt
```

```
En [2]: # Abra el libro de trabajo para leer los valores de las celdas.  
        # El archivo se vuelve a cerrar automáticamente después de cargar los datos.  
        libro = openpyxl.load_workbook("xl / stores.xlsx", datos_only=Cierto)
```

```
En [3]: # Obtener un objeto de hoja de trabajo por nombre o índice (basado en 0)  
        hoja = libro["2019"]hoja =  
        libro.hojas de trabajo[0]
```

```
En [4]: # Obtenga una lista con todos los nombres de las hojas  
        libro.nombres de hoja
```

Fuera [4]: ['2019', '2020', '2019-2020']

```
En [5]: # Recorre los objetos de la hoja.  
        # En lugar de "nombre", openpyxl usa "título".por I  
        en libro.hojas de trabajo:  
            impresión(I.título)
```

2019  
2020  
2019-2020

```
En [6]: # Obteniendo las dimensiones,  
        # es decir, el rango usado de la hoja  
        hoja.max_row, hoja.columna_máx
```

Fuera [6]: (8, 6)

```
En [7]: # Leer el valor de una sola celda  
        # usando la notación "A1" y usando índices de celda (basado en 1)
```

```
hoja["B6"].valor  
hoja.celda(hilera=6, columna=2).valor
```

Fuera [7]: 'Boston'

En [8]: # Leer en un rango de valores de celda usando nuestro módulo de Excel  
datos = sobresalir.leer(libro["2019"], (2, 2), (8, 6))  
datos[:2] # Imprime las dos primeras filas

Fuera [8]: [['Tienda', 'Empleados', 'Gerente', 'Desde', 'Buque insignia'],  
['Nueva York', 10, 'Sarah', datetime.datetime(2018, 7, 20, 0, 0), False]]

## Escribir con OpenPyXL

OpenPyXL crea el archivo de Excel en la memoria y escribe el archivo una vez que llama al ahorrar método. El siguiente código produce el archivo como se muestra en Figura 8-1:

```
En [9]: importar openpyxl  
        de openpyxl.drawing.image importar Imagende  
        openpyxl.chart importar Gráfico de barras, Referenciade  
        openpyxl.styles importar Fuente, coloresde  
        openpyxl.styles.borders importar Frontera, Ladode  
        openpyxl.styles.alignment importar Alineaciónde  
        openpyxl.styles.fills importar PatternFillimportar  
        sobresalir  
  
En [10]: # Crear una instancia de un libro de trabajo  
libro = openpyxl.Libro de trabajo()  
  
# Consigue la primera hoja y dale un nombre  
hoja = libro.hoja activa.título = "Hoja1"  
  
# Escribir celdas individuales usando notación A1  
# e índices de celda (basados en  
1)hoja["A1"].valor = "Hola 1"  
hoja.celda(hilera=2, columna=1, valor="Hola 2")  
  
# Formato: color de relleno, alineación, borde y fuente  
font_format = Fuente(color="FF0000", negrita=Cierto)  
delgada = Lado(estilo de borde="delgada", color="FF0000")  
hoja["A3"].valor = "Hola 3"hoja["A3"].fuente = font_format  
  
hoja["A3"].frontera = Frontera(cima=delgada, izquierda=delgada,  
                                Derecha=delgada, fondo=delgada)  
hoja["A3"].alineación = Alineación(horizontal="centrar")  
hoja["A3"].llenar = PatternFill(fgColor="FFFF00", fill_type="sólido")  
  
# Formateo de números (usando cadenas de formato de  
Excel)hoja["A4"].valor = 3.3333hoja["A4"].formato numérico =  
"0.00"  
  
# Formato de fecha (usando cadenas de formato de Excel)  
hoja["A5"].valor = dt.fecha(2016, 10, 13)
```

```

hoja["A5"].formato_numérico = "mm / dd / aa"

# Fórmula: debes usar el nombre en inglés de la fórmula
# con comas como delimitadoreshoja[
"A6"].valor = "= SUMA (A4, 2)"

# Imagen
hoja.añadir_imagen(Imagen("images / python.png"), "C1")

# Lista bidimensional (estamos usando nuestro módulo de
Excel)datos = [[Ninguno, "Norte", "Sur"],
               ["El año pasado", 2, 5],
               ["Este año", 3, 6]]sobresalir
.escribir(hoja, datos, "A10")

# Gráfico
gráfico = Gráfico_de_barras()
gráfico.escribe = "columna"
gráfico.título = "Ventas por región"
gráfico.x_axis.título = "Regiones"
gráfico.eje_y.título = "Ventas"
chart_data = Referencia(hoja, min_row=11, min_col=1,
                         max_row=12, max_col=3)
chart_categories = Referencia(hoja, min_row=10, min_col=2,
                               max_row=10, max_col=3)
# from_rows interpreta los datos de la misma manera
# como si agregara un gráfico manualmente en Excel
gráfico.agregar_datos(chart_data, títulos_de_datos=Cierto, from_rows=Cierto)
gráfico.set_categories(chart_categories)hoja.add_chart(gráfico, "A15")

# Guardar el libro crea el archivo en el discolibro
ahorrar("openpyxl.xlsx")

```

Si desea escribir un archivo de plantilla de Excel, deberá configurar el plantilla atribuir a Cierto antes de guardarlo:

```

En [11]: libro = openpyxl.Libro_de_trabajo()
          hoja = libro.activo
          hoja["A1"].valor = "Esta es una plantilla"libro.
          plantilla = Cierto libro.ahorrar("template.xlsx")

```

Como puede ver en el código, OpenPyXL establece colores al proporcionar una cadena como FF0000. Este valor se compone de tres valores hexadecimales (FF, 00, y 00) que corresponden a los valores rojo / verde / azul del color deseado. Hex significa *hexadecimal* y representa números usando una base de dieciséis en lugar de una base de diez que se usa en nuestro sistema decimal estándar.



#### Hallar el valor hexadecimal de un color

Para encontrar el valor hexadecimal deseado de un color en Excel, haga clic en el menú desplegable de pintura que usaría para cambiar el color de relleno de una celda, luego seleccione Más colores. Ahora seleccione su color y lea su valor hexadecimal en el menú.

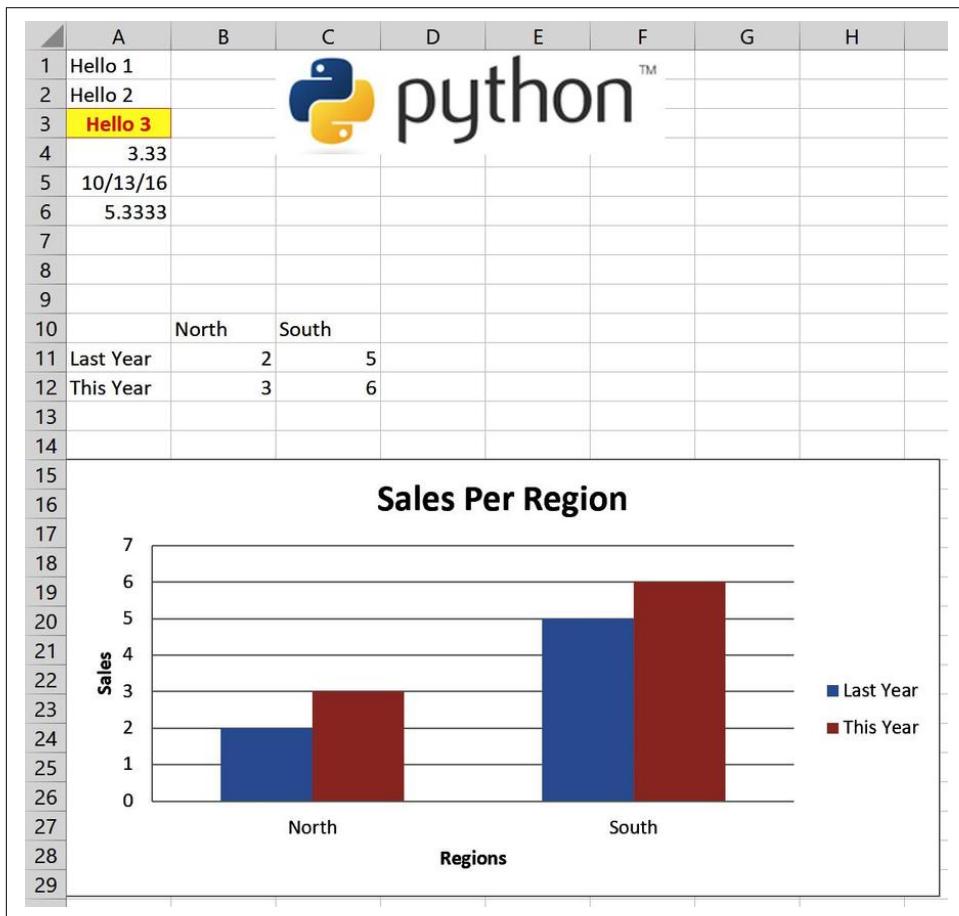


Figura 8-1. El archivo escrito por OpenPyXL (openpyxl.xlsx)

#### Editando con OpenPyXL

No existe un paquete de lector / escritor que realmente pueda editar archivos de Excel: en realidad, OpenPyXL lee el archivo con todo lo que comprende y luego escribe el archivo nuevamente desde cero, incluidos los cambios que realice en el medio. Esto puede ser muy poderoso para archivos simples de Excel que contienen principalmente celdas formateadas con datos y fórmulas, pero es limitado cuando tiene gráficos y otro contenido más avanzado en su hoja de cálculo como

OpenPyXL los cambiará o los eliminará por completo. Por ejemplo, a partir de la v3.0.5, OpenPyXL cambiará el nombre de los gráficos y eliminará su título. Aquí hay un ejemplo de edición simple:

```
En [12]: # Leer el archivo stores.xlsx, cambiar una celda  
# y guárdealo con una nueva ubicación / nombre.libro  
= openpyxl.load_workbook("xl / stores.xlsx")libro[  
"2019"] ["A1"].valor = "modificado"libro.ahorrar(  
"stores_edited.xlsx")
```

Si quieres escribir un *xlsm* archivo, OpenPyXL tiene que trabajar con un archivo existente que necesita cargar con el *keep\_vba* parámetro establecido en Ciento:

```
En [13]: libro = openpyxl.load_workbook("xl / macro.xlsm", keep_vba=Ciento)  
libro["Hoja1"] ["A1"].valor = "¡Haga clic en el botón!"libro.  
ahorrar("macro_openpyxl.xlsm")
```

El botón del archivo de ejemplo está llamando a una macro que muestra un cuadro de mensaje. Open- PyXL cubre muchas más funciones de las que puedo cubrir en esta sección; Por tanto, es una buena idea echar un vistazo a [los documentos oficiales](#). También veremos más funcionalidad al final de este capítulo cuando retomemos el estudio de caso del capítulo anterior nuevamente.

## XlsxWriter

Como sugiere el nombre, XlsxWriter solo puede escribir archivos de Excel. El siguiente código produce el mismo libro de trabajo que producimos anteriormente con OpenPyXL, que se muestra en [Figura 8-1](#). Tenga en cuenta que XlsxWriter usa índices de celda basados en cero, mientras que Open- PyXL usa índices de celda basados en uno; asegúrese de tener esto en cuenta si cambia entre paquetes:

```
En [14]: importar fecha y hora como dt  
importar xlsxwriter  
importar sobresalir  
  
En [15]: # Crear una instancia de un libro de trabajo  
libro = xlsxwriter.Libro de trabajo("xlsxwriter.xlsx")  
  
# Agrega una hoja y dale un nombrehoja  
= libro.add_sheet("Hoja1")  
  
# Escribir celdas individuales usando notación A1  
# e índices de celda (basados  
en 0)hoja.escribir("A1", "Hola 1")  
hoja.escribir(1, 0, "Hola 2")  
  
# Formato: color de relleno, alineación, borde y fuente  
formateo = libro.add_format({"color de fuente": "# FF0000",  
"bg_color": "# FFFF00", "negrita": Ciento,  
"alinear": "centrar", "frontera": 1, "color del  
borde": "# FF0000"})
```

```

hoja.escribir("A3", "Hola 3", formateo)

# Formateo de números (usando cadenas de formato de Excel)
formato numérico = libro.add_format({"num_format": "0.00"})hoja.
escribir("A4", 3.3333, formato numérico)

# Formato de fecha (usando cadenas de formato de Excel)
fecha = libro.add_format({"num_format": "mm / dd / aa"})hoja.
escribir("A5", dt.fecha(2016, 10, 13), formato de fecha)

# Fórmula: debes usar el nombre en inglés de la fórmula
# con comas como delimitadoreshoja.
escribir("A6", "= SUMA (A4, 2)")

# Imagen
hoja.insertar_imagen(0, 2, "images / python.png")

# Lista bidimensional (estamos usando nuestro módulo de
Excel)
datos = [[Ninguno, "Norte", "Sur"],
          ["El año pasado", 2, 5], [
              "Este año", 3, 6]]sobresalir
.escribir(hoja, datos, "A10")

# Gráfico: vea el archivo "sales_report_xlsxwriter.py" en el
# repositorio complementario para ver cómo puede trabajar con índices
# en lugar de direcciones de celda
gráfico = libro.add_chart({"escribe": "columna"})gráfico
.set_title({"nombre": "Ventas por región"})gráfico.
add_series({"nombre": "= Hoja1! A11",
            "categorías": "= Hoja1! B10: C10",
            "valores": "= Hoja1! B11: C11"})
gráfico.add_series({"nombre": "= Hoja1! A12",
                    "categorías": "= Hoja1! B10: C10",
                    "valores": "= Hoja1! B12: C12"})
gráfico.set_x_axis({"nombre": "Regiones"})
gráfico.set_y_axis({"nombre": "Ventas"})
hoja.insert_chart("A15", gráfico)

# Cerrar el libro crea el archivo en el disco.libro.cerrar
()

```

En comparación con OpenPyXL, XlsxWriter tiene que adoptar un enfoque más complicado para escribir *xlsm* archivos, ya que es un paquete de escritura puro. Primero, debe extraer el código de macro de un archivo de Excel existente en Anaconda Prompt (el ejemplo usa el *macro.xlsm* archivo, que encontrará en el *SG* carpeta del repositorio complementario):

### Ventanas

Empiece por cambiar a *SG* directorio, luego busque la ruta a *vba\_extract.py*, un script que viene con XlsxWriter:

```
(base)> cd C:\ Usuarios \nombre de usuario\ python-para-
excel \ xl(base)> donde vba_extract.py
C: \ Usuarios \nombre de usuario\ Anaconda3 \ Scripts \ vba_extract.py
```

Luego use esta ruta en el siguiente comando:

```
(base)> python C: \ ... \ Anaconda3 \ Scripts \ vba_extract.py macro.xlsxm
```

#### Mac OS

En macOS, el comando está disponible como secuencia de comandos ejecutable y se puede ejecutar así:

```
(base)> cd / Usuarios /nombre de usuario/ python-para-
excel / xl(base)> vba_extract.py macro.xlsxm
```

Esto guardará el archivo *vbaProject.bin* en el directorio donde está ejecutando el comando. También he incluido el archivo extraído en el SG carpeta del repositorio complementario. Lo usaremos en el siguiente ejemplo para escribir un libro con un botón de macro:

```
En [dieciséis]: libro = xlsxwriter.Libro de trabajo("macro_xlsxwriter.xlsxm")
hoja = libro.add_worksheet("Hoja1")hoja.
escribir("A1", "¡Haga clic en el botón!")libro.
add_vba_project("xl / vbaProject.bin")
hoja.insert_button("A3", {"macro": "Hola", "subtítulo": "Botón 1",
"ancho": 130, "altura": 35})
libro.cerrar()
```

## pyxlsb

En comparación con las otras bibliotecas de lectores, pyxlsb ofrece menos funcionalidad, pero es su única opción cuando se trata de leer archivos de Excel en binario. *xlsb* formato. pyxlsb no es parte de Anaconda, por lo que deberá instalarlo si aún no lo ha hecho. Actualmente tampoco está disponible a través de Conda, así que use pip para instalarlo:

```
(base)> pip instalar pyxlsb
```

Lea las hojas y los valores de celda de la siguiente manera:

```
En [17]: importar pyxlsb
importar sobresalir
```

```
En [18]: # Pase las hojas. Con pyxlsb, el libro de trabajo
# y los objetos de hoja se pueden usar como administradores de contexto.
# book.sheets devuelve una lista de nombres de hojas, ¡no objetos!
# Para obtener un objeto de hoja, use get_sheet () en su lugar.
con pyxlsb.open_workbook("xl / stores.xlsxb") como libro:
    por sheet_name en libro.hojas:
        con libro.get_sheet(sheet_name) como hoja:
            oscuro = hoja.dimensión
            impresión(F"La hoja '{sheet_name}' tiene"
F"{dim.h} filas y {dim.w} columnas")
```

```
La hoja '2019' tiene 7 filas y 5 columnas La hoja  
'2020' tiene 7 filas y 5 columnas La hoja  
'2019-2020' tiene 20 filas y 5 columnas
```

```
En [19]: # Lea los valores de un rango de celdas usando nuestro módulo de Excel.  
# En lugar de "2019", también puede usar su índice (basado en 1).  
con pyxlsb.open_workbook("xl / stores.xlsb") como libro:  
    con libro.get_sheet("2019") como hoja:  
        datos = sobresalir.leer(hoja, "B2")  
    datos[:2] # Imprime las dos primeras filas
```

```
Fuera [19]: [['Tienda', 'Empleados', 'Gerente', 'Desde', 'Buque insignia'],  
['Nueva York', 10.0, 'Sarah', 43301.0, False]]
```

pyxlsb actualmente no ofrece forma de reconocer celdas con fechas, por lo que tendrá que convertir manualmente los valores de las celdas con formato de fecha en fecha y hora objetos así:

```
En [20]: de pyxlsb importar convert_date  
convert_date(datos[1][3])
```

```
Fuera [20]: datetime.datetime(2018, 7, 20, 0, 0)
```

Recuerde, cuando lea el *xlsb* formato de archivo con una versión de pandas inferior a 1.3, debe especificar el motor explícitamente:

```
En [21]: df = pd.read_excel("xl / stores.xlsb", motor="pyxlsb")
```

## xlrd, xlwt y xlutils

La combinación de xlrd, xlwt y xlutils ofrece aproximadamente la misma funcionalidad para el legado *xls* formato que OpenPyXL ofrece para el *xlsx* formato: lecturas xlrd, escrituras xlwt y ediciones xlutils *xls* archivos. Estos paquetes ya no se desarrollan de forma activa, pero es probable que sean relevantes siempre que todavía existan *xls* archivos alrededor. xlutils no es parte de Anaconda, así que instálelo si aún no lo ha hecho:

```
(base)> conda instalar xlutils
```

¡Empecemos con la parte de lectura!

### Leer con xlrd

El siguiente código de muestra le muestra cómo leer los valores de un libro de trabajo de Excel con xlrd:

```
En [22]: importar xlrd  
importar xlwt  
de xlwt.Utils importar cell_to_rowcol2  
importar xlutils  
importar sobresalir
```

```
En [23]: # Abra el libro de trabajo para leer los valores de las celdas. El archivo es  
# se cierra automáticamente de nuevo después de cargar los  
datos.libro = xlrd.open_workbook("xl / stores.xls")
```

En [24]: # Obtenga una lista con todos los nombres de las hojas  
libro.sheet\_names()

Fuera [24]: ['2019', '2020', '2019-2020']

En [25]: # Recorrer los objetos de la hoja  
por hoja en libro.hojas():  
    impresión(hoja.nombre)

2019  
2020  
2019-2020

En [26]: # Obtener un objeto de hoja por nombre o índice (basado en 0)  
hoja = libro.sheet\_by\_index(0) hoja =  
libro.sheet\_by\_name("2019")

En [27]: # Dimensiones  
hoja.nrows, hoja.ncols

Fuera [27]: (8, 6)

En [28]: # Leer el valor de una sola celda  
# usando la notación "A1" y usando índices de celda (basados en 0).  
# El "\*" descomprime la tupla que devuelve cell\_to\_rowcol2  
# en argumentos individuales.  
hoja.celda(\*cell\_to\_rowcol2("B3")).hoja de  
valor.celda(2, 1).valor

Fuera [28]: 'Nueva York'

En [29]: # Leer en un rango de valores de celda usando nuestro módulo de Excel  
datos = sobresalir.leer(hoja, "B2") datos[2] #  
Imprime las dos primeras filas

Fuera [29]: [['Tienda', 'Empleados', 'Gerente', 'Desde', 'Buque insignia'],  
['Nueva York', 10.0, 'Sarah', datetime.datetime(2018, 7, 20, 0, 0), False]]



Rango utilizado

A diferencia de OpenPyXL y pyxlsb, xlrd devuelve las dimensiones de las celdas con un valor, en lugar del *rango usado* de una hoja al usar sheet.nrows y sheet.ncols. Lo que Excel devuelve como rango usado a menudo contiene filas y columnas vacías en la parte inferior y en el borde derecho del rango. Esto puede suceder, por ejemplo, cuando elimina el contenido de las filas (presionando la tecla Eliminar), en lugar de eliminar las filas en sí mismas (haciendo clic derecho y seleccionando Eliminar).

## Escribir con xlwt

El siguiente código reproduce lo que hemos hecho anteriormente con OpenPyXL y XlsxWriter como se muestra en Figura 8-1. xlwt, sin embargo, no puede producir gráficos y solo admite la bmp formato para imágenes:

```
En [30]: importar xlwt
de xlwt.Utils importar cell_to_rowcol2importar
fecha y hora como dtimportar sobresalir

En [31]: # Crear una instancia de un libro de trabajo
libro = xlwt.Libro de trabajo()

# Agrega una hoja y dale un nombre
hoja = libro.add_sheet("Hoja1")

# Escribir celdas individuales usando notación A1
# e índices de celda (basados en 0)
hoja.escribir(*cell_to_rowcol2("A1"), "Hola 1")hoja.
escribir(r=1, C=0, etiqueta="Hola 2")

# Formato: color de relleno, alineación, borde y fuente
formateo = xlwt.easyxf("fuente: negrita activada, color rojo;""
                        "alinear: centro del horizonte;""
                        "bordes: top_color rojo, bottom_color rojo,""
                                "right_color red, left_color red", "izquierda
                                delgada, derecha delgada",
                                "superior delgada, inferior delgada";
                        "patrón: patrón sólido, fore_color amarillo;")
hoja.escribir(r=2, C=0, etiqueta="Hola 3", estilo=formateo)

# Formateo de números (usando cadenas de formato de Excel)
formato numérico = xlwt.easyxf(num_format_str="0.00")hoja.
escribir(3, 0, 3.3333, formato numérico)

# Formato de fecha (usando cadenas de formato de Excel)formato
de fecha = xlwt.easyxf(num_format_str="mm / dd / aaaa")hoja.
escribir(4, 0, dt.fecha y hora(2012, 2, 3), formato de fecha)

# Fórmula: debes usar el nombre en inglés de la fórmula
# con comas como delimitadores
hoja.escribir(5, 0, xlwt.Fórmula("SUMA (A4, 2)"))

# Lista bidimensional (estamos usando nuestro módulo de
Excel)/datos = [[Ninguno, "Norte", "Sur"],
                 ["El año pasado", 2, 5], [
                     "Este año", 3, 6]]sobresalir
escribir(hoja, datos, "A10")

# Imagen (solo permite agregar formato bmp)hoja.
insert_bitmap("images / python.bmp", 0, 2)
```

```
# Esto escribe el archivo en el disco
libro.ahorrar("xlwt.xls")
```

### Editando con xlutils

xlutils actúa como un puente entre xlrd y xlwt. Esto hace que sea explícito que esta no es una verdadera operación de edición: la hoja de cálculo se lee, incluido el formato a través de xlrd (configurandoformatting\_info = Verdadero) y luego escrito de nuevo por xlwt, incluidos los cambios que se realizaron en el medio:

En [32]: `import xlutils.copy`

```
En [33]: libro = xlrd.open_workbook("xl / stores.xls", formatting_info=Cierto)
libro = xlutils.Copiar.Copiar(libro)
libro.get_sheet(0).escribir(0, 0, "¡cambió!")
ahorrar("stores_edited.xls")
```

En este punto, ya sabe cómo leer y escribir un libro de Excel en un formato específico. La siguiente sección continúa con algunos temas avanzados que incluyen trabajar con archivos grandes de Excel y usar pandas y los paquetes de lector y escritor juntos.

## Temas avanzados de lectores y escritores

Si sus archivos son más grandes y más complejos que los archivos simples de Excel que usamos en los ejemplos hasta ahora, es posible que confiar en las opciones predeterminadas ya no sea lo suficientemente bueno. Por lo tanto, comenzamos esta sección analizando cómo trabajar con archivos más grandes. Luego, aprenderemos cómo usar pandas junto con los paquetes de lector y escritor: esto abrirá la capacidad de diseñar sus DataFrames de pandas de la manera que deseé. Para concluir esta sección, usaremos todo lo que aprendimos en este capítulo para hacer que el informe de Excel del estudio de caso del capítulo anterior parezca mucho más profesional.

### Trabajar con archivos grandes de Excel

Trabajar con archivos grandes puede causar dos problemas: el proceso de lectura y escritura puede ser lento o su computadora puede quedarse sin memoria. Por lo general, el problema de la memoria es de mayor preocupación, ya que hará que su programa se bloquee. Cuando se considera exactamente un archivo grande siempre depende de los recursos disponibles en su sistema y su definición de *lento*. Esta sección muestra las técnicas de optimización que ofrecen los paquetes individuales, lo que le permite trabajar con archivos de Excel que superan los límites. Comenzaré mirando las opciones para las bibliotecas de escritura, seguidas de las opciones para las bibliotecas de lectura. Al final de esta sección, le mostraré cómo leer las hojas de un libro de trabajo en paralelo para reducir el tiempo de procesamiento.

## Escribir con OpenPyXL

Al escribir archivos grandes con OpenPyXL, asegúrese de tener instalado el paquete lxml, ya que esto acelera el proceso de escritura. Está incluido en Anaconda, por lo que no es necesario que haga nada al respecto. La opción crítica, sin embargo, es la write\_only = Verdadero bandera, que asegura que el consumo de memoria permanezca bajo. Sin embargo, le obliga a escribir fila por fila utilizando laadjuntar método y ya no le permitirá escribir celdas individuales:

```
En [34]: libro = openpyxl.Libro de trabajo(escribir solamente=Cierto)
# Con write_only = True, book.active no funciona
hoja =
libro.create_sheet()
# Esto producirá una hoja con 1000 x 200 celdas por
hilera en distancia(1000):
    hoja.adjuntar(lista(distancia(200)))libro.
ahorrar("openpyxl_optimized.xlsx")
```

## Escribir con XlsxWriter

XlsxWriter tiene una opción similar como OpenPyXL llamada constante\_memoria. También te obliga a escribir filas secuenciales. Habilita la opción proporcionando unopciones diccionario como este:

```
En [35]: libro = xlsxwriter.Libro de trabajo("xlsxwriter_optimized.xlsx",
opciones={"constante_memoria": Cierto})
hoja = libro.add_worksheet()
# Esto producirá una hoja con 1000 x 200 celdas por
hilera en distancia(1000):
    hoja.write_row(hilera , 0, lista(distancia(200)))
libro.cerrar()
```

## Leer con xlrd

Al leer archivos grandes en el legado x/s formato, xlrd le permite cargar hojas bajo demanda, así:

```
En [36]: con xlrd.open_workbook("xl / stores.xls", Bajo demanda=Cierto) como libro:
    hoja = libro.sheet_by_index(0) # Carga solo la primera hoja
```

Si no usaría el libro de trabajo como administrador de contexto como lo hacemos aquí, deberá llamar book.release\_resources () manualmente para cerrar correctamente el libro de trabajo nuevamente. Para usar xlrd en este modo con pandas, utilícelo así:

```
En [37]: con xlrd.open_workbook("xl / stores.xls", Bajo demanda=Cierto) como libro:
    con pd.Archivo Excel(libro, motor="xlrd") como F:
        df = pd.read_excel(F, sheet_name=0)
```

## Leer con OpenPyXL

Para mantener la memoria bajo control al leer archivos grandes de Excel con OpenPyXL, debe cargar el libro con `read_only = Verdadero`. Dado que OpenPyXL no es compatible con la declaración, deberá asegurarse de cerrar el archivo nuevamente cuando haya terminado. Si su archivo contiene enlaces a libros de trabajo externos, es posible que también desee utilizar `keep_links = Falso` para hacerlo más rápido. `keep_links` se asegura de que se mantengan las referencias a libros de trabajo externos, lo que puede ralentizar innecesariamente el proceso si solo está interesado en leer los valores de un libro de trabajo:

```
En [38]: libro = openpyxl.load_workbook("xl / big.xlsx",
                                         datos_only=Cierto, solo lectura=
                                         Cierto, keep_links=Falso)
# Realice aquí las operaciones de lectura deseadas
libro.cerrar() # Requerido con read_only = True
```

## Leer hojas en paralelo

Cuando usas pandas' `read_excel` función para leer en varias hojas de un gran libro de trabajo, encontrará que esto lleva mucho tiempo (llegaremos a un ejemplo concreto en un momento). La razón es que los pandas leen las hojas de forma secuencial, es decir, una tras otra. Para acelerar las cosas, puede leer las hojas en paralelo. Si bien no existe una manera fácil de parallelizar la escritura de libros de trabajo debido a cómo los archivos están estructurados internamente, leer varias hojas en paralelo es bastante simple. Sin embargo, dado que la parallelización es un tema avanzado, lo dejé fuera de la introducción de Python y tampoco entraré en detalles aquí.

En Python, si desea aprovechar los múltiples núcleos de CPU que tiene cada computadora moderna, use el paquete de multiprocesamiento que es parte de la biblioteca estándar. Esto generará varios intérpretes de Python (generalmente uno por núcleo de CPU), que trabajan en una tarea en paralelo. En lugar de procesar una hoja tras otra, tiene un intérprete de Python que procesa la primera hoja, mientras que al mismo tiempo un segundo intérprete de Python está procesando la segunda hoja, etc. Sin embargo, cada intérprete de Python adicional toma algún tiempo para iniciarse y utiliza memoria adicional, por lo que si tiene archivos pequeños, lo más probable es que se ejecuten más lentamente cuando paralelice el proceso de lectura en lugar de hacerlo más rápido. En el caso de un archivo grande con varias hojas grandes, el multiprocesamiento puede acelerar el proceso sustancialmente, sin embargo, siempre asumiendo que su sistema tiene la memoria necesaria para manejar la carga de trabajo. Si ejecuta el cuaderno Jupyter en Binder como se muestra en [Capítulo 2](#), no tendrá suficiente memoria y, por lo tanto, la versión paralela se ejecutará más lentamente. En el repositorio complementario, encontrará `paralelo_parallel_das.py`, que es una implementación simple para leer las hojas en paralelo, utilizando OpenPyXL como motor. Es fácil de usar, por lo que no necesitará saber nada sobre multiprocesamiento:

```
importar pandas_parallel
pandas_parallel.read_excel(nombre del archivo, sheet_name=Ninguno)
```

De forma predeterminada, se leerá en todas las hojas, pero puede proporcionar una lista de los nombres de las hojas que desea procesar. Al igual que los pandas, la función devuelve un diccionario de la siguiente forma: `{"sheetname": df}`, es decir, las claves son los nombres de las hojas y los valores son los DataFrames.

## El comando mágico %% tiempo

En las siguientes muestras, voy a hacer uso de %%tiempo magia celular. Introduce comandos mágicos en [Capítulo 5](#) en relación con Matplotlib. %%tiempo es una celda mágica que puede ser muy útil para un ajuste de rendimiento simple, ya que facilita la comparación del tiempo de ejecución de dos celdas con diferentes fragmentos de código. *Tiempo de pared* es el tiempo transcurrido desde el inicio hasta el final del programa, es decir, la celda. Si está en macOS o Linux, no solo obtendrá el tiempo de la pared, sino una línea adicional para *Tiempos de CPU* a lo largo de estas líneas:

```
Tiempos de CPU: usuario 49,4 s, sys: 108 ms, total: 49,5 s
```

Los tiempos de CPU miden el tiempo dedicado a la CPU, que puede ser menor que el tiempo de la pared (si el programa tiene que esperar a que la CPU esté disponible) o mayor (si el programa se ejecuta en varios núcleos de CPU en paralelo). Para medir el tiempo con mayor precisión, use %%cronometralo en lugar de %%tiempo, que ejecuta la celda varias veces y toma el promedio de todas las ejecuciones. %%tiempo y %%cronometralo son magia celular, es decir, deben estar en la primera línea de la celda y medirán el tiempo de ejecución de toda la celda. Si, en cambio, desea medir solo una línea, comience esa línea con %%tiempo o %%cronometralo.

Veamos cuánto más rápido lee la versión paralelizada *big.xlsx* archivo que encontrará en el repositorio complementario *SG* carpeta:

En [39]: %%tiempo

```
datos = pd.read_excel("xl / big.xlsx",
                      sheet_name=Ninguno, motor="openpyxl")
```

Tiempo de pared: 49,5 s

En [40]: %%tiempo

```
importar pandas_paralelos
datos = pandas_paralelos.read_excel("xl / big.xlsx", sheet_name=Ninguno)
```

Tiempo de pared: 12,1 s

Para obtener el DataFrame que representa Sheet1, debe escribir datos ["Sheet1"] en ambos casos. Al observar el tiempo de pared de ambas muestras, verá que la versión paralelizada fue varias veces más rápida que `pd.read_excel` con este libro de trabajo en particular y en mi computadora portátil con 6 núcleos de CPU. Si lo desea aún más rápido, paralelice OpenPyXL directamente: también encontrará una implementación para eso en el repositorio complementario (*paralelo\_openpyxl.py*), junto con una implementación para `xlrd` para leer el legado *xls* formato en paralelo (*paralelo\_xlrd.py*). Revisar los paquetes subyacentes en lugar de pandas le permitirá omitir la transformación en un DataFrame o solo aplicar

los pasos de limpieza que necesita, que probablemente lo ayudarán a hacer las cosas más rápido si esa es su mayor preocupación.

## Leer una hoja en paralelo con Modin

Si solo está leyendo en una hoja enorme, vale la pena mirar [Modin](#), un proyecto que actúa como un reemplazo directo de los pandas. Paraleliza el proceso de lectura de una sola hoja y proporciona impresionantes mejoras de velocidad. Dado que Modin requiere una versión específica de pandas, podría degradar la versión que viene con Anaconda cuando lo instale. Si desea probarlo, le recomendaría que cree un entorno Conda separado para esto para asegurarse de que no está alterando su entorno base. Ver[Apéndice A](#) para obtener instrucciones más detalladas sobre cómo crear un entorno Conda:

```
(base)> conda crear --nombre modin python = 3.8 -y  
(base)> conda activar modin  
(modin)> conda install -c conda-forge modin -y
```

En mi máquina y usando el *big.xlsx*, ejecutar el siguiente código tomó aproximadamente cinco segundos, mientras que los pandas tardaron unos doce segundos:

```
importar modin.pandas  
datos = modin.pandas.read_excel("xl / big.xlsx",  
                                 sheet_name=0, motor="openpyxl")
```

Ahora que sabe cómo tratar con archivos grandes, sigamos adelante y veamos cómo podemos usar pandas y los paquetes de bajo nivel juntos para mejorar el formato predeterminado al escribir DataFrames en archivos de Excel.

## Formateo de DataFrames en Excel

Para formatear DataFrames en Excel de la manera que queramos, podemos escribir código que use pandas junto con OpenPyXL o XlsxWriter. Primero usaremos esta combinación para agregar un título al DataFrame exportado. Luego formatearemos el encabezado y el índice de un DataFrame antes de terminar esta sección formateando la parte de datos de un DataFrame. La combinación de pandas con OpenPyXL para leer también puede ser útil ocasionalmente, así que comencemos con esto:

```
En [41]: con pd.Archivo Excel("xl / stores.xlsx", motor="openpyxl") como xfile:  
        # Leer un DataFrame  
        df = pd.read_excel(xfile, sheet_name="2020")  
  
        # Obtener el objeto del libro de trabajo de  
        OpenPyXLlibro = xfile.libro  
  
        # A partir de aquí, es código OpenPyXL
```

```
hoja = libro["2019"]
valor = hoja["B3"].valor # Leer un solo valor
```

Al escribir libros de trabajo, funciona de manera análoga, lo que nos permite agregar fácilmente un título a nuestro informe DataFrame:

```
En [42]: con pd.ExcelWriter("pandas_and_openpyxl.xlsx",
                             motor="openpyxl") como escritor:
    df = pd.Marco de datos({"col1": [1, 2, 3, 4], "col2": [5, 6, 7, 8]})
    # Escribir un DataFrame
    df.para sobresalir(escritor, "Hoja1", startrow=4, startcol=2)

    # Obtenga el libro de trabajo y los objetos de hoja de
    OpenPyXLlibro = escritor.libro
    hoja = escritor.hojas["Hoja1"]

    # A partir de aquí, es código OpenPyXL
    hoja["A1"].valor = "Este es un título" # Escribe un valor de celda única
```

Estas muestras usan OpenPyXL, pero conceptualmente funciona igual con los otros paquetes. Continuemos ahora descubriendo cómo podemos formatear el índice y el encabezado de un DataFrame.

#### Formatear el índice y los encabezados de un DataFrame

La forma más fácil de obtener un control completo sobre el formato del índice y los encabezados de las columnas es simplemente escribirlos usted mismo. El siguiente ejemplo le muestra cómo hacer esto con OpenPyXL y XlsxWriter, respectivamente. Puedes ver la salida en [Figura 8-2](#). Comencemos creando un DataFrame:

```
En [43]: df = pd.Marco de datos({"col1": [1, -2], "col2": [-3, 4]},
                                 índice=["fila1", "fila2"])
df.índice.nombre = "ix"
df
```

```
Fuera [43]:      col1  col2
ix
fila1      1     - 3
fila2     - 2      4
```

Para formatear el índice y los encabezados con OpenPyXL, haga lo siguiente:

```
En [44]: de openpyxl.styles importar PatternFill
```

```
En [45]: con pd.ExcelWriter("formatting_openpyxl.xlsx",
                           motor="openpyxl") como escritor:
    # Escriba el df con el formato predeterminado en A1df.
    para sobresalir(escritor, startrow=0, startcol=0)

    # Escriba el df con formato personalizado de índice / encabezado en
    A6startrow, startcol = 0, 5
    # 1. Escriba la parte de datos del DataFrame df.para
    sobresalir(escritor, encabezamiento=False, índice=False,
```

```

startrow=startrow + 1, startcol=startcol + 1)
# Obtenga el objeto de hoja y cree un objeto de estilo
hoja = escritor.hojas["Hoja1"]
estilo = PatternFill(fgColor="D9D9D9", fill_type="sólido")

# 2. Escriba los encabezados de las columnas con estilo
por I, columna en enumerar(df.columnas):
    hoja.celda(hilera=startrow + 1, columna=I + startcol + 2,
                valor=columna).llenar = estilo

# 3. Escribe el índice con estilo
índice = [df.índice.nombre si df.índice.nombre demás Ninguno] + lista(df.índice)
por I, hilera en enumerar(índice):
    hoja.celda(hilera=I + startrow + 1, columna=startcol + 1,
                valor=hilera).llenar = estilo

```

Para formatear el índice y los encabezados con XlsxWriter, deberá ajustar el código ligeramente:

```

En [46]: # Formateo de índices / encabezados con XlsxWriter
con pd.ExcelWriter("formatting_xlsxwriter.xlsx",
                    motor="xlsxwriter") como escritor:
    # Escriba el df con el formato predeterminado en A1df.
    para sobresalir(escritor, startrow=0, startcol=0)

    # Escriba el df con formato personalizado de índice / encabezado en
    A6startrow, startcol = 0, 5
    # 1. Escriba la parte de datos del DataFrame df.para
    sobresalir(escritor, encabezamiento=False, índice=False,
               startrow=startrow + 1, startcol=startcol + 1)
    # Obtenga el libro y el objeto de hoja y cree un objeto de estilo
    libro = escritor.libro
    hoja = escritor.hojas["Hoja1"]
    estilo = libro.add_format({"bg_color": "# D9D9D9"})

    # 2. Escriba los encabezados de las columnas con estilo
    por I, columna en enumerar(df.columnas):
        hoja.escribir(startrow, startcol + I + 1, columna, estilo)

    # 3. Escribe el índice con estilo
    índice = [df.índice.nombre si df.índice.nombre demás Ninguno] + lista(df.índice)
    por I, hilera en enumerar(índice):
        hoja.escribir(startrow + I, startcol, hilera, estilo)

```

Con el índice y el encabezado formateados, ¡veamos cómo podemos diseñar la parte de datos!

	A	B	C	D	E	F	G	H
1	ix	col1	col2			ix	col1	col2
2	row1		1	3		row1		3
3	row2		2	4		row2		4

Figura 8-2. Un DataFrame con el formato predeterminado (izquierda) y con un formato personalizado (derecha)

#### Formatear la parte de datos de un DataFrame

Las posibilidades que tiene para formatear la parte de datos de un DataFrame dependen del paquete que esté usando: si usa pandas<sup>1</sup> para sobresalir, OpenPyXL puede aplicar un formato a cada celda, mientras que XlsxWriter solo puede aplicar formatos por filas o columnas. Por ejemplo, para establecer el formato numérico de las celdas en tres decimales y alinear al centro el contenido como se muestra en [Figura 8-3](#), haga lo siguiente con OpenPyXL:

```
En [47]: de openpyxl.styles importar Alineación
En [48]: con pd.ExcelWriter("data_format_openpyxl.xlsx",
                           motor="openpyxl") como escritor:
# Escriba el DataFrame df para
sobresalir(escritor)

# Obtener el libro y los objetos de la hoja
libro = escritor.libro
hoja = escritor.hojas["Hoja1"]

# Formatear celdas individuales
nrows, ncols = df.formapor hilera
en distancia(nrows):
    por columna en distancia(ncols):
        # + 1 para dar cuenta del encabezado / índice
        # + 1 ya que OpenPyXL está basado en
        1celda = hoja.celda(hilera=hilera + 2,
                             columna=columna + 2)
        celda.formato numérico = "0.000"
        celda.alineación = Alineación(horizontal="centrar")
```

Para XlsxWriter, ajuste el código de la siguiente manera:

```
En [49]: con pd.ExcelWriter("data_format_xlsxwriter.xlsx",
                           motor="xlsxwriter") como escritor:
# Escriba el DataFrame df para
sobresalir(escritor)

# Obtener el libro y los objetos de la hoja
libro = escritor.libro
hoja = escritor.hojas["Hoja1"]

# Formatear las columnas (las celdas individuales no se pueden formatear)
formato numérico = libro.add_format({"num_format": "0.000",
                                       "alinear": "centrar"})
hoja.set_column(first_col=1, last_col=2,
                 cell_format=formato numérico)
```

	A	B	C
1		col1	col2
2	row1	1.000	-3.000
3	row2	-2.000	4.000

Figura 8-3. Un DataFrame con una parte de datos formateada

Como alternativa, pandas ofrece *experimental* apoyo para el estilo propiedad de Data- Frames. Experimental significa que la sintaxis puede cambiar en cualquier momento. Dado que los estilos se introdujeron para formatear los DataFrames en formato HTML, utilizan CSS sintaxis. CSS significa *Hojas de estilo en cascada* y se utiliza para definir el estilo de elementos HTML. Para aplicar el mismo formato que en el ejemplo anterior (tres decimales y alineación central), deberá aplicar una función a cada elemento de unEstilista objeto a través de applymap. Obtienes un Estilista objeto a través del df.style atributo:

```
En [50]: df.estilo.aplicar mapa(lambda X: "formato de número: 0.000;"  
                                "text-align: center") \  
        . para sobresalir("styled.xlsx")
```

El resultado de este código es el mismo que se muestra en Figura 8-3. Para obtener más detalles sobre el enfoque de estilo DataFrame, consulte directamente el [documentos de estilo](#).

Sin tener que depender del atributo de estilo, pandas ofrece soporte para formatear los objetos de fecha y hora como se muestra en Figura 8-4.:

```
En [51]: df = pd.Marco de datos({"Fecha": [dt.fecha(2020, 1, 1)],  
                                "Fecha y hora": [dt.fecha y hora(2020, 1, 1, 10)]})  
        con pd.ExcelWriter("fecha.xlsx",  
                            formato de fecha="aaaa-mm-dd",  
                            datetime_format="aaaa-mm-dd hh: mm: ss") como escritor:  
        df.para sobresalir(escritor)
```

	A	B	C
1		Date	Datetime
2	0	2020-01-01	2020-01-01 10:00:00

Figura 8-4. Un DataFrame con fechas formateadas

## Otros paquetes de lectura y escritura

A parte de los paquetes que hemos visto en este capítulo, hay algunos otros que pueden ser interesantes para casos de uso específicos:

### *pyexcel*

**pyexcel** ofrece una sintaxis armonizada en diferentes paquetes de Excel y otros formatos de archivo, incluidos archivos CSV y archivos OpenOffice.

### *PyExcelerate*

El objetivo de **PyExcelerate** es escribir archivos de Excel de la manera más rápida posible.

### *pylightxl*

**pylightxl** puedo leer *xlsx* y *xsm* archivos y escribir *xlsx* archivos.

### *marco de estilo*

**marco de estilo** envuelve pandas y OpenPyXL para producir archivos de Excel con DataFrames muy bien formateados.

### *olétoles*

**olétoles** no es un paquete clásico de lectura o escritura, pero se puede utilizar para analizar documentos de Microsoft Office, por ejemplo, para el análisis de malware. Ofrece una forma conveniente de extraer código VBA de libros de Excel.

Ahora que ya sabe cómo formatear DataFrames en Excel, es hora de volver a intentar el estudio de caso del capítulo anterior y ver si podemos mejorar el informe de Excel con el conocimiento de este capítulo.

## Estudio de caso (revisado): Informes de Excel

Habiendo llegado al final de este capítulo, sabe lo suficiente para poder volver al informe de Excel del estudio de caso del capítulo anterior y hacerlo visualmente más atractivo. Si quieres, vuelve a *sales\_report\_pandas.py* en el repositorio complementario e intente convertirlo en el informe como se muestra en [Figura 8-5](#).

Los números rojos son cifras de ventas inferiores a 20.000. No he tocado todos los aspectos del formato en este capítulo (como cómo aplicar el formato condicional), por lo que tendrá que usar la documentación del paquete con el que elija trabajar. Para comparar su solución, he incluido dos versiones del script que producen este informe en el repositorio complementario. La primera versión está basada en OpenPyXL (*sales\_report\_open-pyxl.py*) y el otro está basado en XlsxWriter (*sales\_report\_xlsxwriter.py*). Ver los scripts uno al lado del otro también puede permitirle tomar una decisión más informada sobre qué paquete desea elegir para su próxima tarea de redacción. Volveremos a este estudio de caso una vez más en el próximo capítulo: allí, confiaremos en una instalación de Microsoft Excel para trabajar con plantillas de informes.

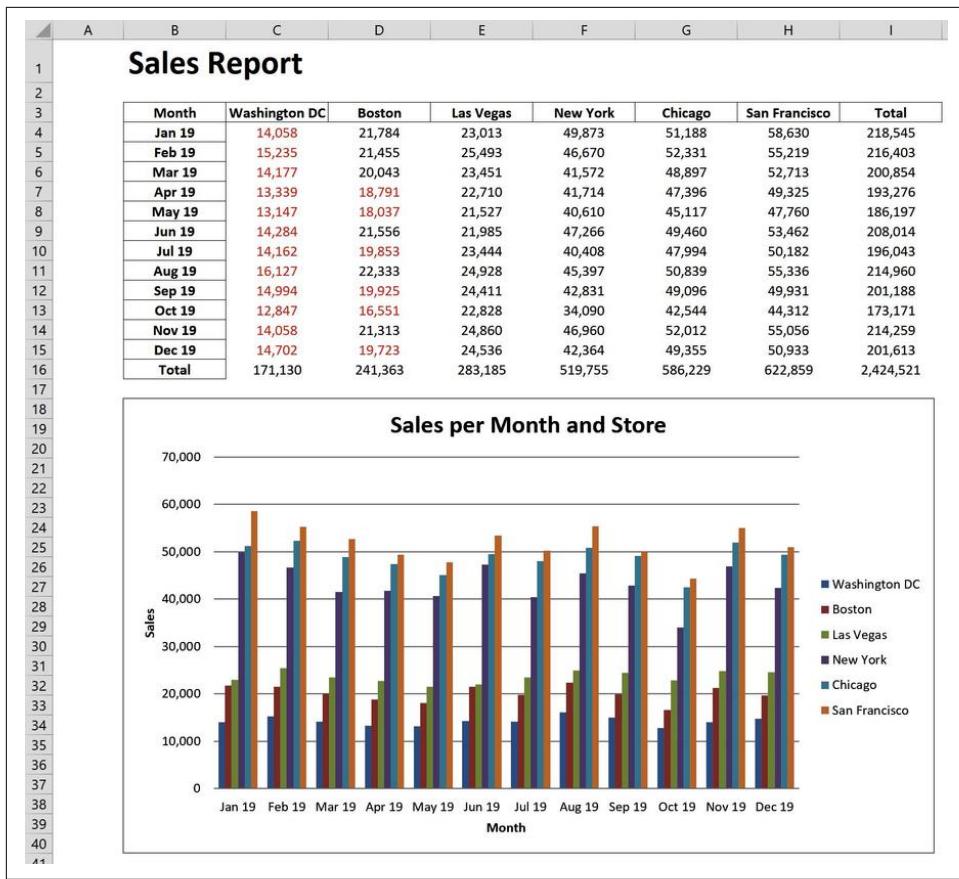


Figura 8-5. El informe de ventas revisado creado por `sales_report_openpyxl.py`

## Conclusión

En este capítulo, le presenté los paquetes de lectura y escritura que los pandas usan debajo del capó. Usarlos directamente nos permite leer y escribir libros de Excel sin necesidad de tener pandas instalados. Sin embargo, usarlos en combinación con pandas nos permite mejorar los informes de Excel DataFrame agregando títulos, gráficos y formato. Si bien los paquetes de lectura y escritura actuales son increíblemente poderosos, todavía espero que algún día veamos un “momento NumPy” que una los esfuerzos de todos los desarrolladores en un solo proyecto. Sería genial saber qué paquete usar sin tener que mirar primero una tabla y sin tener que usar una sintaxis diferente para cada tipo de archivo de Excel. En ese sentido,

Excel, sin embargo, es mucho más que un archivo de datos o un informe: la aplicación Excel es una de las interfaces de usuario más intuitivas donde los usuarios pueden ingresar algunos números y hacer que muestre la información que están buscando. Automatizar la aplicación de Excel en lugar de leer y escribir archivos de Excel abre una nueva gama de funcionalidades que vamos a explorar enParte IV. El siguiente capítulo comienza este viaje mostrándole cómo controlar Excel desde Python de forma remota.

**PARTE IV**

---

## **Programación de la aplicación Excel con xlwings**



### Automatización de Excel

Hasta ahora, hemos aprendido cómo reemplazar las tareas típicas de Excel con pandas ([Parte II](#)) y cómo utilizar archivos de Excel como fuente de datos y como formato de archivo para sus informes ([Parte III](#)). Este capítulo comienza [Parte IV](#), donde dejamos de manipular Excel *archivos* con los paquetes de lector y escritor y comience a automatizar Excel *solicitud* con xlwings.

El caso de uso principal de xlwings es crear aplicaciones interactivas donde las hojas de cálculo de Excel actúan como la interfaz de usuario, lo que le permite llamar a Python haciendo clic en un botón o llamando a una función definida por el usuario; ese es el tipo de funcionalidad que no cubre los paquetes de lector y escritor. Pero eso no significa que xlwings no se pueda usar para leer y escribir archivos, siempre que esté en macOS o Windows y tenga Excel instalado. Una ventaja que xlwings tiene en esta área es la capacidad de editar realmente archivos de Excel, en todos los formatos, sin cambiar ni perder nada del contenido o formato existente. Otra ventaja es que puede leer los valores de celda de un libro de Excel sin la necesidad de guardarlo primero. Sin embargo, también puede tener mucho sentido usar un paquete de lector / escritor de Excel y xlwings juntos, [Capítulo 7](#) una vez más.

Comenzaré este capítulo presentándole el modelo de objetos de Excel y xlwings: primero aprenderemos los conceptos básicos como conectarse a un libro de trabajo o leer y escribir valores de celda antes de profundizar un poco más para comprender cómo los convertidores y las opciones nos permiten para trabajar con pandas DataFrames y matrices NumPy. También analizamos cómo interactuar con gráficos, imágenes y nombres definidos antes de pasar a la última sección, que explica cómo funciona xlwings bajo el capó: esto le dará el conocimiento necesario para hacer que sus scripts funcionen a la vez que funcionan. en torno a la funcionalidad faltante.

A partir de este capítulo, deberá ejecutar los ejemplos de código en Windows o macOS, ya que dependen de una instalación local de Microsoft Excel.<sup>1</sup>

## Empezando con xlwings

Uno de los objetivos de xlwings es servir como un reemplazo directo de VBA, lo que le permite interactuar con Excel desde Python en Windows y macOS. Dado que la cuadrícula de Excel es el diseño perfecto para mostrar las estructuras de datos de Python como listas anidadas, matrices NumPy y Pandas DataFrames, una de las características principales de xlwings es hacer que leerlos y escribirlos desde y hacia Excel sea lo más fácil posible. Comenzaré esta sección presentándole Excel como visor de datos; esto es útil cuando interactúa con DataFrames en un cuaderno de Jupyter. Luego explicaré el modelo de objetos de Excel antes de explorarlo de forma interactiva con xlwings. Para concluir esta sección, le mostraré cómo llamar al código VBA que aún puede tener en los libros de trabajo heredados. Dado que xlwings es parte de Anaconda, no es necesario que lo instalemos manualmente.

### Usar Excel como visor de datos

Probablemente haya notado en los capítulos anteriores que, de forma predeterminada, los cuadernos de Jupyter ocultan la mayoría de los datos para DataFrames más grandes y solo muestran las filas superior e inferior, así como la primera y la última columna. Una forma de tener una mejor idea de los datos es trazarlos; esto le permite detectar valores atípicos u otras irregularidades. A veces, sin embargo, es realmente útil poder desplazarse por una tabla de datos. Despues de leer Capítulo 7, sabes cómo usar el método `to_clipboard()` en su DataFrame. Si bien esto funciona, puede ser un poco engorroso: debe darle un nombre al archivo de Excel, buscarlo en el sistema de archivos, abrirlo y, después de realizar cambios en su DataFrame, debe cerrar el archivo de Excel y ejecutar el todo el proceso de nuevo. Una mejor idea puede ser correr `df.to_clipboard()`, que copia el DataFrame `df` al portapapeles, lo que le permite pegarlo en Excel, pero hay una forma aún más sencilla: utilice el vista función que viene con xlwings:

*En [1]: # Primero, importemos los paquetes que usaremos en este capítulo.*

```
importar fecha y hora como dt
importar xlwings como xwimportar
pandas como pdimportar numpy
como notario público
```

---

<sup>1</sup> En Windows, necesita al menos Excel 2007, y en macOS, necesita al menos Excel 2016. Alternativamente, puede instalar la versión de escritorio de Excel, que es parte de su suscripción a Microsoft 365. Consulte su suscripción para obtener detalles sobre cómo hacer esto.

```

En [2]: # Creamos un DataFrame basado en números pseudoaleatorios y
          # con suficientes filas para que solo se muestren la cabeza y la coladf = pd.
          Marco de datos(datos=notario.público.aleatorio.randn(100, 5),
                         columnas=[F"Prueba {i}" por I en distancia(1, 6)])
df

Fuera [2]:      Prueba 1    Prueba 2    Prueba 3    Prueba 4    Prueba 5
0   -1.313877  1,164258 -1,306419 -0,529533 -0,524978
1   -0,854415  0,022859 -0,246443 -0,229146 -0,005493
2   -0,327510 -0,492201 -1,353566 -1,229236      0,024385
3   -0,728083 -0,080525  0,628288 -0,382586 -0,590157
4   -1,227684  0,498541 -0,266466  0,297261 -1,297985
..   ...     ...     ...     ...
95 -0,903446  1,103650  0,033915  0,336871  0,345999  96
-1,354898 -1,290954 -0,738396 -1,102659  0,115076  97
-0,070092 -0,416991 -0,203445 -0,686915 -1,163205  98
-1,201963      0,471854 -0,458501 -0,357171  1,954585
99  1,863610   0,214047 -1,426806  0,751906 -2,338352

[100 filas x 5 columnas]

```

```

En [3]: # Ver el DataFrame en Excel
xw.vista(df)

```

los vista La función acepta todos los objetos comunes de Python, incluidos números, cadenas, listas, diccionarios, tuplas, matrices NumPy y pandas DataFrames. De forma predeterminada, abre un nuevo libro de trabajo y pega el objeto en la celda A1 de la primera hoja; incluso ajusta el ancho de las columnas mediante la función Autoajustar de Excel. En lugar de abrir un libro nuevo cada vez, también puede reutilizar el mismo proporcionando el vista función y xlwings hoja objeto como segundo argumento: xw.view (df, mi hoja). ¿Cómo se accede a un hoja objeto y cómo encaja en el modelo de objetos de Excel es lo que explicaré a continuación.<sup>2</sup>

---

<sup>2</sup> Tenga en cuenta que xlwings 0.22.0 introdujo el xw.load función, que es similar a xw.view, pero funciona en el opuesto dirección del sitio: le permite cargar un rango de Excel fácilmente en un cuaderno de Jupyter como un DataFrame de pandas, consultelos [docs](#).



## macOS: permisos y preferencias

En macOS, asegúrese de ejecutar los cuadernos Jupyter y VS Code desde un indicador de Anaconda (es decir, a través de la terminal) como se muestra en [Capítulo 2](#). Esto asegura que será recibido por dos ventanas emergentes cuando use xlwings por primera vez: la primera es "Terminal quiere acceso para controlar eventos del sistema" y la segunda es "Terminal quiere acceso para controlar Microsoft Excel". Deberá confirmar ambas ventanas emergentes para permitir que Python automatice Excel. En teoría, estas ventanas emergentes deberían ser activadas por cualquier aplicación desde la que ejecute el código xlwings, pero en la práctica, a menudo ese no es el caso, por lo que ejecutarlas a través de la Terminal le evitará problemas. Además, deberá abrir las Preferencias de Excel y desmarcar "Mostrar galería de libros al abrir Excel" en la categoría General. Esto abre Excel directamente en un libro de trabajo vacío en lugar de abrir la galería primero, lo que se interpondría en su camino cuando abra una nueva instancia de Excel a través de xlwings.

## El modelo de objetos de Excel

Cuando trabaja con Excel mediante programación, interactúa con sus componentes como un libro de trabajo o una hoja. Estos componentes están organizados en el *Modelo de objetos de Excel*, una estructura jerárquica que representa la interfaz gráfica de usuario de Excel (ver [Figura 9-1](#)). Microsoft utiliza en gran medida el mismo modelo de objetos con todos los lenguajes de programación que admite oficialmente, ya sea VBA, Office Scripts (la interfaz de JavaScript para Excel en la web) o C#. En contraste con los paquetes de lector y escritor de [Capítulo 8](#), xlwings sigue muy de cerca el modelo de objetos de Excel, solo con un soplo de aire fresco: por ejemplo, xlwings usa los nombres aplicación en lugar de solicitud y libro en lugar de libro de trabajo:

- Un aplicación contiene la libros colección
- A libro contiene la hojas colección
- A hoja da acceso a distancia objetos y colecciones como gráficos
- A distancia contiene una o más celdas contiguas como elementos

Las casillas punteadas son *colecciones* y contienen uno o más objetos del mismo tipo. Una aplicación corresponde a una instancia de Excel, es decir, una aplicación de Excel que se ejecuta como un proceso separado. Los usuarios avanzados a veces usan varias instancias de Excel en paralelo para abrir el mismo libro dos veces, por ejemplo, para calcular un libro con diferentes entradas en paralelo. Con las versiones más recientes de Excel, Microsoft hizo que fuera un poco más complicado abrir varias instancias de Excel manualmente: inicie Excel, luego haga clic con el botón derecho en su ícono en la barra de tareas de Windows. En el menú que aparece, haga clic con el botón izquierdo en la entrada de Excel mientras mantiene presionada la tecla Alt al mismo tiempo (asegúrese de mantener la tecla Alt).

presionado hasta después de soltar el botón del mouse): una ventana emergente le preguntará si desea iniciar una nueva instancia de Excel. En macOS, no existe una forma manual de iniciar más de una instancia del mismo programa, pero puede iniciar varias instancias de Excel mediante programación a través de xlwings, como veremos más adelante. En resumen, una instancia de Excel es una *en caja de arena* entorno, lo que significa que una instancia no se puede comunicar con la otra.<sup>3</sup> Los hoja object le da acceso a colecciones como gráficos, imágenes y nombres definidos, temas que veremos en la segunda sección de este capítulo.

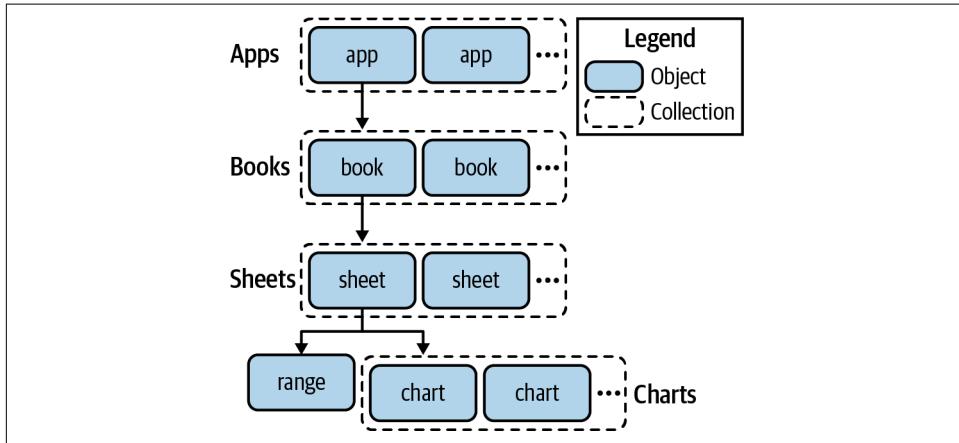


Figura 9-1. El modelo de objetos de Excel implementado por xlwings (extracto)

### Configuración regional y de idioma

Este libro se basa en la versión de Excel en inglés estadounidense. Ocasionalmente me referiré a nombres predeterminados como "Libro1" u "Hoja1", que serán diferentes si usa Excel en otro idioma. Por ejemplo, "Hoja1" se llama "Feuille1" en francés y "Hoja1" en español. También el *separador de lista*, que es el separador que utiliza Excel en las fórmulas de celda, depende de su configuración: usará la coma, pero su versión puede requerir un punto y coma u otro carácter. Por ejemplo, en lugar de escribir =SUMA (A1, A2), necesitarás escribir =SUMME (A1; A2) en una computadora con configuración regional alemana.

En Windows, si desea cambiar el separador de lista de un punto y coma a una coma, debe cambiarlo fuera de Excel a través de la configuración de Windows: haga clic en el botón de inicio de Windows, busque Configuración (o haga clic en el ícono del engranaje), luego vaya a "Hora e idioma" > "Región e idioma" > "Fecha, hora y configuración regional adicionales", donde finalmente haga clic en "Región" > "Cambiar ubicación". En "Separador de lista", podrá cambiarlo de un punto y coma a una coma. Tenga en cuenta que esto solo

<sup>3</sup> Ver "[¿Qué son las instancias de Excel y por qué es importante?](#)" para obtener más información sobre instancias de Excel independientes.

funciona si su "símbolo decimal" (en el mismo menú) no es también una coma. Para anular los separadores de miles y decimales de todo el sistema (pero no el separador de listas), en Excel, vaya a "Opciones">> "Avanzado", donde encontrará la configuración en "Opciones de edición".

En macOS, funciona de manera similar, excepto que no puede cambiar el separador de lista directamente: en Preferencias del sistema de su macOS (no Excel), seleccione Idioma y región. Allí, configure una región específica globalmente (en la pestaña General) o específicamente para Excel (en la pestaña Aplicaciones).

Para familiarizarse con el modelo de objetos de Excel, como de costumbre, es mejor jugar con él de forma interactiva. Empecemos con el libro clase: le permite crear nuevos libros de trabajo y conectarse a los existentes; ver [Tabla 9-1](#) para obtener una descripción general.

*Tabla 9-1. Trabajar con libros de Excel*

Mando	Descripción
xw.Book ()	Devuelve un libro objeto que representa un nuevo libro de Excel en la instancia activa de Excel. Si no hay una instancia activa, se iniciará Excel.
xw.Book ("Libro1")	Devuelve un libro objeto que representa un libro de trabajo no guardado con el nombre Libro1 (nombre sin extensión de archivo).
xw.Book ("Libro1.xlsx")	Devuelve un libro objeto que representa un libro previamente guardado con el nombre Libro1.xlsx (nombre con extensión de archivo). El archivo debe estar abierto o en el directorio de trabajo actual.
xw.Book (r "C: \ ruta \ Libro1.xlsx")	Devuelve un libro objeto de un libro previamente guardado (ruta de archivo completa). El archivo puede estar abierto o cerrado. El lidierr convierte la cadena en una cadena sin procesar para que las barras invertidas (\) de la ruta se interpreten literalmente en Windows (introduce cadenas sin procesar en <a href="#">Capítulo 5</a> ). En macOS, el r no es necesario ya que las rutas de archivo utilizan barras diagonales en lugar de barras invertidas.
xw.books.active	Devuelve un libro objeto que representa el libro activo en la instancia activa de Excel.

Veamos cómo podemos recorrer la jerarquía del modelo de objetos desde el libro Objeto hasta el distancia objeto:

**En [4]:** # Cree un nuevo libro de trabajo vacío e imprima su nombre. Este es el libro que usaremos para ejecutar la mayoría de los ejemplos de código en este capítulo.  
libro = xw.Book(libro.nombre)

Fuera [4]: 'Libro2'

**En [5]:** # Accediendo a la colección de sábanas  
libro.hojas

Salida [5]: Hojas ([<Hoja [Libro2] Hoja1>])

**En [6]:** # Obtenga un objeto de hoja por índice o nombre. Necesitarás ajustarte  
# "Hoja1" si su hoja se llama de manera diferente.

```
hoja1 = libro.hojas[0]hoja1 = libro.  
hojas["Hoja1"]
```

En [7]: hoja1.distancia("A1") Fuera [7]:

```
<Rango [Libro2] Hoja1! $ A $ 1>
```

Con el distancia objeto, hemos llegado al final de la jerarquía. La cadena que se imprime entre corchetes angulares le brinda información útil sobre ese objeto, pero para hacer algo, generalmente usa el objeto con un atributo, como muestra el siguiente ejemplo:

En [8]: # Tareas más habituales: escribir valores ...

```
hoja1.distancia("A1").valor = [[1, 2],  
                                 [3, 4]]  
hoja1.distancia("A4").valor = "¡Hola!"
```

En [9]: # ...y leer valores

```
hoja1.distancia("A1: B2").valor
```

Fuera [9]: [[1.0, 2.0], [3.0, 4.0]]En [10]:

```
hoja1.distancia("A4").valorFuera [10]:
```

```
'¡Hola!'
```

Como puede ver, por defecto, el valor atributo de un xlwings distancia El objeto acepta y devuelve una lista anidada para rangos bidimensionales y un escalar para una sola celda. Todo lo que hemos usado hasta ahora es casi idéntico a VBA: asumiendo quelibro es un objeto de libro de trabajo de VBA o xlwings, respectivamente, así es como se accede al valor atributo de las celdas A1 a B2 en VBA y con xlwings:

```
libro.Hojas(1).Distancia("A1: B2").Libro de      # VBA  
valor.hojas[0].distancia("A1: B2").valor        # xlwings
```

Las diferencias son:

### Atributos

Python usa letras minúsculas, potencialmente con guiones bajos como sugiere PEP 8, la guía de estilo de Python que presenté en [Capítulo 3](#).

### Indexación

Python usa corchetes e índices de base cero para acceder a un elemento en el hojas colección.

[Tabla 9-2](#) le da una descripción general de las cadenas que un xlwings distancia acepta.

Tabla 9-2. Cadenas para definir un rango en notación A1

Referencia	Descripción
"A1"	Una sola celda
"A1: B2"	Celdas de A1 a B2
"AUTOMÓVIL CLUB BRASILEÑO"	Columna A
"A: B"	Columnas A a B
"1: 1"	Fila 1
"1: 2"	Filas 1 a 2

Trabajo de indexación y división con xlwings distancia objetos: observe la dirección entre corchetes angulares (la representación del objeto impreso) para ver con qué rango de celdas termina:

En [11]: # Indexación  

```
hoja1.distancia("A1: B2") [0, 0]
```

Fuera [11]: <Rango [Libro2] Hoja1! \$ A \$ 1>

En [12]: # Rebanar  

```
hoja1.distancia("A1: B2") [:, 1]
```

Fuera [12]: <Rango [Libro2] Hoja1! \$ B \$ 1: \$ B \$ 2>

La indexación corresponde al uso de Células propiedad en VBA:

```
libro.Hojas(1).Distancia("A1: B2").Células(1, 1) # VBA
libro.hojas[0].distancia("A1: B2") [0, 0] # xlwings
```

En lugar de usar distancia explícitamente como un atributo de la hoja objeto, también puede obtener un distancia objeto indexando y cortando el hoja objeto. Usar esto con la notación A1 le permitirá escribir menos, y usar esto con índices enteros hace que la hoja de Excel se sienta como una matriz NumPy:

En [13]: # Celda única: notación A1  

```
hoja1["A1"]
```

Fuera [13]: <Rango [Libro2] Hoja1! \$ A \$ 1>

En [14]: # Varias celdas: notación A1  

```
hoja1["A1: B2"]
```

Fuera [14]: <Rango [Libro2] Hoja1! \$ A \$ 1: \$ B \$ 2>

En [15]: # Celda única: indexación  

```
hoja1[0, 0]
```

Fuera [15]: <Rango [Libro2] Hoja1! \$ A \$ 1>

En [dieciséis]: # Varias celdas: rebanar  

```
hoja1[:, :2]
```

Fuera [16]: <Rango [Libro2] Hoja1! \$ A \$ 1: \$ B \$ 2>

A veces, sin embargo, puede resultar más intuitivo definir un rango haciendo referencia a las celdas superior izquierda e inferior derecha de un rango. Las siguientes muestras se refieren a los rangos de celdas D10 y D10: F11, respectivamente, lo que le permite comprender la diferencia entre indexar / cortar unhoja objeto y trabajando con un distancia objeto:

En [17]: # D10 mediante indexación de hojas  
hoja1[9, 3]

Fuera [17]: <Rango [Libro2] Hoja1! \$ D \$ 10>

En [18]: # D10 mediante objeto de rango  
hoja1.distancia((10, 4))

Fuera [18]: <Rango [Libro2] Hoja1! \$ D \$ 10>

En [19]: # D10: F11 mediante corte de hojas  
hoja1[9:11, 3:6]

Fuera [19]: <Rango [Libro2] Hoja1! \$ D \$ 10: \$ F \$ 11>

En [20]: # D10: F11 a través del objeto de rango  
hoja1.distancia((10, 4), (11, 6))

Fuera [20]: <Rango [Libro2] Hoja1! \$ D \$ 10: \$ F \$ 11>

Definiendo distancia objetos con tuplas es muy similar a cómo el Células La propiedad funciona en VBA, como muestra la siguiente comparación; esto supone nuevamente que libro es un objeto de libro de trabajo de VBA o un xlwings libro objeto. Veamos primero la versión de VBA:

Con libro.Hojas(1)  
myrange = .Distancia(.Células(10, 4), .Células(11, 6))

Terminar con

Esto es equivalente a la siguiente expresión xlwings:

myrange = libro.hojas[0].distancia((10, 4), (11, 6))



#### Índices basados en cero frente a uno

Como paquete de Python, xlwings utiliza consistentemente la indexación basada en cero siempre que accede a elementos a través del índice de Python o la sincronización de segmentos, es decir, a través de corchetes. xlwingsdistancia los objetos, sin embargo, usan los índices de fila y columna basados en uno de Excel. Tener los mismos índices de fila / columna que la interfaz de usuario de Excel a veces puede ser beneficioso. Si prefiere usar solo la indexación de base cero de Python, simplemente usa el hoja [row\_selection, column\_selection] sintaxis.

El siguiente ejemplo le muestra cómo obtener de un distancia objeto hoja1 ["A1"] todo el camino hasta el aplicación objeto. Recuerda que la aplicación El objeto representa una instancia de Excel (la salida entre paréntesis angulares representa el ID de proceso de Excel y, por lo tanto, será diferente en su máquina):

En [21]: hoja1["A1"].hoja.libro.aplicación

Fuera [21]: <Excel App 9092>

Habiendo llegado a la cima del modelo de objetos de Excel, es un buen momento para ver cómo puede trabajar con múltiples instancias de Excel. Necesitará utilizar la aplicación objeto explícitamente si desea abrir el mismo libro de trabajo en varias instancias de Excel o si desea distribuir específicamente sus libros de trabajo en diferentes instancias por razones de rendimiento. Otro caso de uso común para trabajar con una aplicación El objetivo es abrir su libro de trabajo en una instancia oculta de Excel: esto le permite ejecutar un script xlwings en segundo plano sin que le impida hacer otro trabajo en Excel mientras tanto:

En [22]: # Obtener un objeto de aplicación del libro de trabajo abierto  
# y cree una instancia de aplicación invisible adicional  
visible\_app = hoja1.libro.aplicación invisible\_app = xw.  
Aplicación(visible=False)

En [23]: # Enumere los nombres de libros que están abiertos en cada instancia  
# mediante el uso de una lista de comprensión  
[libro.nombre por libro en visible\_app.libros]

Fuera [23]: ['Libro1', 'Libro2']

En [24]: [libro.nombre por libro en invisible\_app.libros]

Fuera [24]: ['Libro3']

En [25]: # Una clave de aplicación representa el ID de proceso (PID)  
xw.aplicaciones.teclas()

Fuera [25]: [5996, 9092]

En [26]: # También se puede acceder a través del atributo pid  
xw.aplicaciones.activo.pid

Fuera [26]: 5996

En [27]: # Trabajar con el libro en la instancia invisible de Excel  
libro\_invisible = invisible\_app.libros[0]  
libro\_invisible.hojas[0] ["A1"].valor = "Creado por una aplicación invisible".

En [28]: # Guarde el libro de Excel en el directorio xl  
libro\_invisible.ahorrar("xl / invisible.xlsx")

En [29]: # Salir de la instancia invisible de Excel  
invisible\_app.dejar()



#### macOS: acceder al sistema de archivos mediante programación

Si ejecuta el comando `ahorrar` en macOS, obtendrá una ventana emergente que le permitirá conceder acceso a los archivos en Excel. Deberá confirmar haciendo clic en el botón `Seleccionar` antes de hacer clic en `Conceder acceso`. En macOS, Excel es *en caja de arena*, lo que significa que su programa solo puede acceder a archivos y carpetas fuera de la aplicación de Excel al confirmar este mensaje. Una vez confirmado, Excel recordará las ubicaciones y no volverá a molestarlo cuando ejecute el script la próxima vez.

Si tiene el mismo libro abierto en dos instancias de Excel o si desea especificar en qué instancia de Excel desea abrir un libro, no puede usar `xw.Book` ya sea que no. En su lugar, debe utilizar la colección de libros como se presenta en **Tabla 9-3**. Tenga en cuenta que `myapp` representa un objeto `xlwings` aplicación. Si reemplaza `myapp.books` con `xw.books` en su lugar, `xlwings` usará el activo aplicación.

Tabla 9-3. Trabajando con la colección de libros

Mando	Descripción
<code>myapp.books.add()</code>	Crea un nuevo libro de Excel en la instancia de Excel que <code>myapp</code> se refiere y devuelve el correspondiente libro objeto.
<code>myapp.books.open(r"C:\ruta\Libro.xlsx")</code>	Devuelve el libro si ya está abierto, de lo contrario, lo abre primero en la instancia de Excel que <code>myapp</code> se refiere a. Recuerde que el lidierr convierte la ruta del archivo en una cadena sin formato para interpretar las barras invertidas literalmente.
<code>myapp.books["Book1.xlsx"]</code>	Devuelve el libro objeto si está abierto. Esto levantará un <code>KeyError</code> si aún no está abierto. Asegúrese de utilizar el nombre y no la ruta completa. Use esto si necesita saber si un libro de trabajo ya está abierto en Excel.

Antes de profundizar en cómo las `xlwings` pueden *reemplazar* sus macros VBA, veamos cómo `xlwings` puede *interactuar* con su código VBA existente: esto puede ser útil si tiene mucho código heredado y no tiene tiempo para migrar todo a Python.

### Ejecutando código VBA

Si tiene proyectos de Excel heredados con mucho código VBA, puede ser mucho trabajo migrar todo a Python. En ese caso, puede usar Python para ejecutar sus macros VBA. El siguiente ejemplo utiliza el `vba.xls` archivo que encontrarás en el `SG` carpeta del repositorio complementario. Contiene el siguiente código en `Module1`:

**Función MySum(X Como Doble, y Como Doble) Como Doble**

`MySum = X + y`

**Función final**

```
Sub ShowMsgBox(msg Como Cuerda)
    MsgBox msg
End Sub
```

Para llamar a estas funciones a través de Python, primero debe crear una instancia de xlwings macro objeto que llama posteriormente, haciéndolo sentir como si fuera una función nativa de Python:

```
En [30]: vba_book = xw.Libro("xl / vba.xlsx")
En [31]: # Crear una instancia de un objeto macro con la función VBA
mysum = vba_book.macro("Module1.MySum")
# Llamar a una función de
VBAmysum(5, 4)
```

Fuera [31]: 9.0

```
En [32]: # Funciona igual con un procedimiento Sub de VBA
show_msgbox = vba_book.macro("Module1.ShowMsgBox")
show_msgbox("¡Hola xlwings!")
```

```
En [33]: # Vuelva a cerrar el libro (asegúrese de cerrar el cuadro de mensajes primero)
vba_book.cerrar()
```



#### No almacene las funciones de VBA en los módulos Sheet y ThisWorkbook

Si almacena la función VBA MySum en el módulo del libro de trabajo Este libro de trabajo o un módulo de hoja (p. ej., Hoja1), tienes que referirte a él como ThisWorkbook.MySum o Sheet1.MySum. Sin embargo, no podrás para acceder al valor de retorno de la función desde Python, así que asegúrese de almacenar las funciones VBA en un módulo de código VBA estándar que inserta haciendo clic derecho en la carpeta Módulos en el editor VBA.

Ahora que sabe cómo interactuar con el código VBA existente, podemos continuar nuestra exploración de xlwings viendo cómo usarlo con DataFrames, matrices NumPy y colecciones como gráficos, imágenes y nombres definidos.

## Convertidores, opciones y colecciones

En los ejemplos de código introductorio de este capítulo, ya estábamos leyendo y escribiendo una cadena y una lista anidada desde y hacia Excel usando el valor atributo de un xlwings distancia objeto. Comenzaré esta sección mostrándole cómo funciona esto con los marcos de datos de pandas antes de echar un vistazo más de cerca a lopciones método que nos permite influir en cómo xlwings lee y escribe valores. Seguimos adelante con cuadros, imágenes y nombres definidos, las colecciones a las que normalmente accedes desde unhoja objeto. Armados con estos conceptos básicos de xlwings, veremos de nuevo el estudio de caso de informes de Capítulo 7.

## Trabajar con DataFrames

Escribir un DataFrame en Excel no es diferente de escribir un escalar o una lista anidada en Excel: simplemente asigne el DataFrame a la celda superior izquierda de un rango de Excel:

```
En [34]: datos=[["Marcos", 55, "Italia", 4.5, "Europa"],  
           ["John", 33, "ESTADOS UNIDOS", 6,7, "America"]]  
        = pd.Marco de datos(datos=datos,  
                           columnas=["nombre", "la edad", "país",  
                                      "puntaje", "continente"],  
                           índice=[1001, 1000])  
        df.índice.nombre = "user_id"
```

	nombre	edad	país	puntaje	continente
user_id					
1001	Marcos	55	Italia	4.5	Europa
1000	John	33	Estados Unidos	6,7	America

```
En [35]: hoja1["A6"].valor = df
```

Sin embargo, si desea suprimir los encabezados de columna y / o el índice, utilice el opciones método como este:

```
En [36]: hoja1["B10"].opciones(encabezamiento=False, índice=False).valor = df
```

La lectura de rangos de Excel como DataFrames requiere que proporcione el Marco de datos clase como el convertir parámetro en el opciones método. De forma predeterminada, espera que sus datos tengan tanto un encabezado como un índice, pero puede volver a usar el índice y encabezamiento parámetros para cambiar esto. En lugar de usar el convertidor, también puede leer los valores primero como una lista anidada y luego construir manualmente su DataFrame, pero usar el convertidor hace que sea un poco más fácil manejar el índice y el encabezado.



### El método de expansión

En el siguiente ejemplo de código, presento el expandir método que facilita la lectura de un bloque contiguo de celdas, entregando el mismo rango que si estuviera haciendo Shift + Ctrl + Flecha abajo + Flecha derecha en Excel, excepto que expandir salta sobre una celda vacía en la esquina superior izquierda.

```
En [37]: df2 = hoja1["A6"].expandir().opciones(pd.Marco de datos).valor  
df2
```

	nombre	edad	país	puntaje	continente
user_id					
1001,0	Marcos	55,0	Italia	4.5	Europa
1000,0	John	33,0	Estados Unidos	6,7	America

```
En [38]: # Si desea que el índice sea un índice entero,  
# puedes cambiar su tipo de datos
```

```
df2.índice = df2.índice.astype(En t)df2
```

```
Fuera [38]:      nombre    edad país     puntaje continente
 1001   Marcos  55,0  Italia     4.5    Europa
 1000    John   33,0  Estados Unidos  6,7  America
```

```
En [39]: # Al establecer index = False, pondrá todos los valores de Excel en
          # la parte de datos del DataFrame y usará el índice predeterminado
hoja1["A6"].expandir().opciones(pd.Marco de datos, índice=False).valor
```

```
Fuera [39]:      user_id    nombre    edad país     puntaje continente
 0    1001,0   Marcos  55,0  Italia     4.5    Europa
 1    1000,0    John   33,0  Estados Unidos  6,7  America
```

Leer y escribir DataFrames fue un primer ejemplo de cómo funcionan los convertidores y las opciones. Cómo se definen formalmente y cómo se utilizan con otras estructuras de datos es lo que analizaremos a continuación.

## Convertidores y opciones

Como acabamos de ver, el opciones método de los xlwings distancia El objeto le permite influir en la forma en que se leen y escriben los valores desde y hacia Excel. Es decir, opciones solo se evalúan cuando llamas al valor atributo en un distancia objeto. La sintaxis es la siguiente (myrange es un xlwings distancia objeto):

```
myrange.opciones(convertir=Ninguno, Opción 1=valor1, opcion 2=valor2, ...).valor
```

Cuadro 9-4 muestra los convertidores incorporados, es decir, los valores que el convertir el argumento acepta. Se les llama *incorporado* as xlwings ofrece una forma de escribir sus propios convertidores, lo que podría ser útil si tiene que aplicar repetidamente transformaciones adicionales antes de escribir o después de leer valores. Para ver cómo funciona, eche un vistazo a la [xlwings docs](#).

Cuadro 9-4. Convertidores integrados

Convertidor	Descripción
dictar	Diccionarios simples sin anidamiento, es decir, en la forma {clave1: valor1, clave2: valor2, ...}
np.array	Matrices NumPy, requiere importar numpy como np
serie pd.	Serie pandas, requiere importar pandas como pd
pd.DataFrame	pandas DataFrame, requiere importar pandas como pd

Ya hemos utilizado el índice y encabezamiento opciones con el ejemplo de DataFrame, pero hay más opciones disponibles, como se muestra en Tabla 9-5.

Tabla 9-5. Opciones integradas

Opción	Descripción
vacío	De forma predeterminada, las celdas vacías se leen como Ninguno. Cambie esto proporcionando un valor para vacío. Acepta una función que se aplica a valores de celdas con formato de fecha. Acepta una función que se aplica a números.
fecha	
número	
ndim	<i>Número de dimensiones:</i> al leer, use ndim para forzar que los valores de un rango lleguen en una cierta dimensionalidad. Debe ser cualquiera Ninguno, 1, o 2. Se puede utilizar al leer valores como listas o matrices NumPy.
transponer	Transpone los valores, es decir, convierte las columnas en filas o viceversa.
índice	Para usar con pandas DataFrames y Series: al leer, utilícelo para definir si el rango de Excel contiene el índice. Puede ser Verdadero Falso o un número entero. El entero define cuántas columnas deben convertirse en un MultiIndex. Por ejemplo, 2 utilizará las dos columnas más a la izquierda como índice. Al escribir, puede decidir si desea escribir el índice configurando índice para Cierto o Falso.
encabezamiento	Funciona igual que índice, pero aplicado a los encabezados de columna.

Echemos un vistazo más de cerca a ndim: de forma predeterminada, cuando lee en una sola celda de Excel, obtendrá un escalar (por ejemplo, un flotante o una cadena); cuando lea en una columna o fila, obtendrá una lista simple; y finalmente, cuando lea en un rango bidimensional, obtendrá una lista anidada (es decir, bidimensional). Esto no solo es consistente en sí mismo, sino que también es equivalente a cómo funciona el corte con matrices NumPy, como se ve en [Capítulo 4](#). El caso unidimensional es especial: a veces, una columna puede ser simplemente un caso marginal de lo que de otro modo sería un rango bidimensional. En este caso, tiene sentido forzar un rango para que siempre llegue como una lista bidimensional usandondim = 2:

En [40]: # Rango horizontal (unidimensional)  
hoja1["A1: B1"].valor

Fuera [40]: [1.0, 2.0]

En [41]: # Rango vertical (unidimensional)  
hoja1["A1: A2"].valor

Fuera [41]: [1.0, 3.0]

En [42]: # Rango horizontal (bidimensional)  
hoja1["A1: B1"].opciones(ndim=2).valor

Fuera [42]: [[1.0, 2.0]]

En [43]: # Rango vertical (bidimensional)  
hoja1["A1: A2"].opciones(ndim=2).valor

Fuera [43]: [[1.0], [3.0]]

En [44]: # El uso del convertidor de matrices NumPy se comporta de la misma manera:  
# el rango vertical conduce a una matriz unidimensional hoja1["A1:  
A2"].opciones(notario público.formación).valor

Fuera [44]: matriz ([1., 3.])

En [45]: # Preservando la orientación de la columna  
hoja1["A1: A2"].opciones(notario público.formación, ndim=2).valor

Fuera [45]: matriz ([[1],  
[3.]])

En [46]: # Si necesita escribir una lista verticalmente,  
# la opción "transponer" es útil  
hoja1["D1"].opciones(transponer=Cierto).valor = [100, 200]

Usar ndim = 1 para forzar que el valor de una sola celda se lea como una lista en lugar de un escalar. No necesitarás ndim con pandas, ya que un DataFrame es siempre bidimensional y una Serie siempre es unidimensional. Aquí hay un ejemplo más que muestra cómo mover, fecha, y número las opciones funcionan:

En [47]: # Escribe algunos datos de muestra  
hoja1["A13"].valor = [dt.fecha y hora(2020, 1, 1), Ninguno, 1.0]

En [48]: # Vuelve a leerlo usando las opciones predeterminadas  
hoja1["A13: C13"].valor

Fuera [48]: [datetime.datetime (2020, 1, 1, 0, 0), Ninguno, 1.0]

En [49]: # Vuelve a leerlo usando opciones no predeterminadas  
hoja1["A13: C13"].opciones(vacio="N / A",  
fechas=dt.fecha,  
números=En t).valor

Fuera [49]: [datetime.date (2020, 1, 1), 'NA', 1]

Hasta ahora, hemos trabajado con libro, hoja, y distancia objetos. Pasemos ahora a aprender cómo manejar colecciones como gráficos a los que accede desde el hoja ¡objeto!

## Gráficos, imágenes y nombres definidos

En esta sección, le mostraré cómo trabajar con tres colecciones a las que accede a través del hoja o libro objeto: gráficos, imágenes y nombres definidos.<sup>4</sup> xlwings solo admite la funcionalidad de gráficos más básica, pero como puede trabajar con plantillas, es posible que ni siquiera se pierda mucho. Y para compensar, xlwings le permite incrustar diagramas de Matplotlib como imágenes; tal vez recuerde Capítulo 5 que Matplotlib es el backend de trazado predeterminado de Pandas. ¡Comencemos por crear un primer gráfico de Excel!

### Gráficos de Excel

Para agregar un nuevo gráfico, use el agregar método del gráficos recopilación y, a continuación, establezca el tipo de gráfico y los datos de origen:

En [50]: hoja1["A15"].valor = [[Ninguno, "Norte", "Sur"],  
["El año pasado", 2, 5],  
["Este año", 3, 6]]

---

4 Otra colección popular es mesas. Para usarlos, necesita al menos xlwings 0.21.0; ver el [docs](#).

```
En [51]: gráfico = hoja1.gráficos.agregar(cima=hoja1["A19"].cima,
                                             izquierda=hoja1["A19"].izquierda)
gráfico.chart_type = "column_clustered" gráfico.
set_source_data(hoja1["A15"].expandir())
```

Esto producirá la tabla que se muestra en el lado izquierdo de [Figura 9-2](#). Para buscar los tipos de gráficos disponibles, eche un vistazo a la [xlwings docs](#). Si le gusta trabajar con gráficos de pandas más que con gráficos de Excel, o si desea utilizar un tipo de gráfico que no está disponible en Excel, xlwings lo tiene cubierto, ¡veamos cómo!

#### Imágenes: parcelas de Matplotlib

Cuando utiliza el backend de trazado predeterminado de pandas, está creando un diagrama de Matplotlib. Para llevar un gráfico de este tipo a Excel, primero debe obtener su figura objeto, que proporciona como argumento para pictures.add—esto convertirá el gráfico en una imagen y lo enviará a Excel:

```
En [52]: # Leer en los datos del gráfico como DataFrame
df = hoja1["A15"].expandir().opciones(pd.Marco de datos).valor df
```

Fuera [52]:	norte	Sur
El año pasado	2,0	5,0
Este año	3,0	6,0

```
En [53]: # Habilite Matplotlib usando el comando mágico del cuaderno
# y cambia al estilo "seaborn"%matplotlib en línea
importar matplotlib.pyplot como plt
plt.estilo.usar("marinero")
```

```
En [54]: # El método de trazado de pandas devuelve un objeto "eje" de
# donde puedes conseguir la figura. "T" transpone la
# DataFrame para llevar la trama a la orientación deseada
hacha = df.T.trama.bar()
higo = hacha.get_figure()
```

```
En [55]: # Envía la trama a Excel
trama = hoja1.imágenes.agregar(higo, nombre="SalesPlot",
                                cima=hoja1["H19"].cima,izquierda=hoja1["H19"].izquierda)
# Escalemos la trama al 70%
trama.ancho, trama.altura = trama.ancho * 0,7, trama.altura * 0,7
```

Para actualizar la imagen con un nuevo gráfico, simplemente use el actualizar método con otro figura objeto: técnicamente, esto reemplazará la imagen en Excel, pero conservará todas las propiedades como la ubicación, el tamaño y el nombre:

```
En [56]: hacha = (df + 1).T.trama.bar()
trama = trama.actualizar(hacha.get_figure())
```

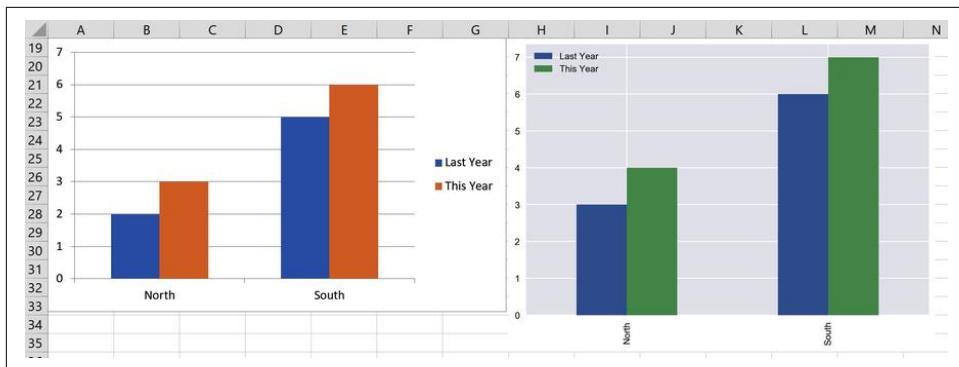


Figura 9-2. Un gráfico de Excel (izquierda) y un gráfico de Matplotlib (derecha)

**Figura 9-2** muestra cómo se comparan el gráfico de Excel y el gráfico de Matplotlib después de la actualizar llama.



#### Asegúrese de que la almohada esté instalada

Cuando trabaje con imágenes, asegúrese de que [Almohada](#), La biblioteca de referencia de Python para imágenes, está instalada: esto asegurará que las imágenes lleguen en el tamaño y la proporción correctos en Excel. Pillow es parte de Anaconda, por lo que si usa una distribución diferente, necesitará instálelo ejecutando `conda install almohada` o instalación de pip almohada. Tenga en cuenta que `pictures.add` también acepta una ruta a una imagen en el disco en lugar de una figura de Matplotlib.

Los gráficos y las imágenes son colecciones a las que se accede a través de un hoja objeto. Nombres definidos, la colección que vamos a ver a continuación, se puede acceder desde elhoja o la libro objeto. ¡Veamos qué diferencia hace esto!

#### Nombres definidos

En Excel, crea un *nombre definido* asignando un nombre a un rango, una fórmula o una constante.<sup>5</sup> Asignar un nombre a un rango es probablemente el caso más común y se denomina *rango con nombre*. Con un rango con nombre, puede hacer referencia al rango de Excel en fórmulas y código utilizando un nombre descriptivo en lugar de una dirección abstracta en forma de A1: B2. Usarlos con xlwings hace que su código sea más flexible y sólido: leer y escribir valores desde y hacia rangos con nombre le brinda la flexibilidad de reestructurar su libro de trabajo sin tener que ajustar su código Python: un nombre se adhiere a la celda,

---

<sup>5</sup> Los nombres definidos con fórmulas también se utilizan para *funciones lambda*, una nueva forma de definir funciones definidas por el usuario ciones sin VBA o JavaScript, que Microsoft anunció como una nueva característica para los suscriptores de Microsoft 365 en diciembre de 2020.

incluso si lo mueve insertando una nueva fila, por ejemplo. Los nombres definidos se pueden establecer en el ámbito del libro global o en el ámbito de la hoja local. La ventaja de un nombre con alcance de hoja es que puede copiar la hoja sin tener conflictos con rangos con nombre duplicados. En Excel, puede agregar nombres definidos manualmente yendo a Fórmulas> Definir nombre o seleccionando un rango, luego escribiendo el nombre deseado en el Cuadro de nombre; este es el cuadro de texto a la izquierda de la barra de fórmulas, donde verá la dirección de la celda por defecto. Así es como administra los nombres definidos con xlwings:

En [57]: # El alcance del libro es el alcance predeterminado  
hoja1["A1: B2"].nombre = "matriz1"

En [58]: # Para el alcance de la hoja, anteponga el nombre de la hoja con  
# un signo de exclamación  
hoja1["B10: E11"].nombre = "Hoja1! Matriz2"

En [59]: # Ahora puede acceder al rango por nombre  
hoja1["matriz1"]

Fuera [59]: <Rango [Libro2] Hoja1! \$ A \$ 1: \$ B \$ 2>

En [60]: # Si accede a la colección de nombres a través del objeto "sheet1",  
# contiene solo nombres con el alcance de esa hoja  
hoja1.nombres

Fuera [60]: [<Nombre 'Hoja1! Matriz2': = Hoja1! \$ B \$ 10: \$ E \$ 11>]

En [61]: # Si accede a la colección de nombres a través del objeto "libro",  
# contiene todos los nombres, incluido el alcance del libro y la hoja  
.nombres

Fuera [61]: [<Nombre 'matriz1': = Hoja1! \$ A \$ 1: \$ B \$ 2>, <Nombre 'Hoja1! Matriz2':  
= Hoja1! \$ B \$ 10: \$ E \$ 11>]

En [62]: # Los nombres tienen varios métodos y atributos.  
# Puede, por ejemplo, obtener el objeto de rango respectivo.  
libro.nombres["matriz1"].refiere\_al\_rango

Fuera [62]: <Rango [Libro2] Hoja1! \$ A \$ 1: \$ B \$ 2>

En [63]: # Si desea asignar un nombre a una constante  
# o una fórmula, use el método "agregar"libro.  
nombres.agregar("EURUSD", "= 1,1151")

Fuera [63]: <Nombre 'EURUSD': = 1.1151>

Eche un vistazo a los nombres definidos generados en Excel abriendo el Administrador de nombres a través de Fórmulas> Administrador de nombres (consulte Figura 9-3). Tenga en cuenta que Excel en macOS no tiene un Administrador de nombres; en su lugar, vaya a Fórmulas> Definir nombre, desde donde verá los nombres existentes.

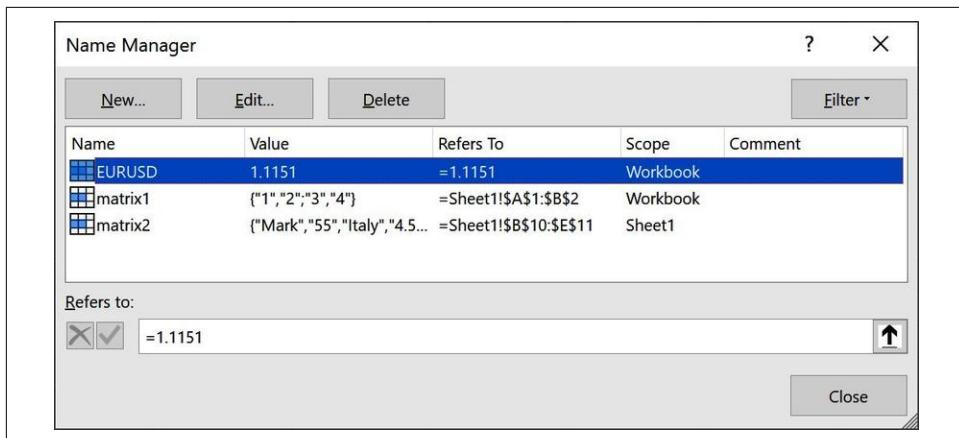


Figura 9-3. Administrador de nombres de Excel después de agregar algunos nombres definidos a través de xlwings

En este punto, sabe cómo trabajar con los componentes más utilizados de un libro de Excel. Esto significa que podemos ver el estudio de caso de informes de Capítulo 7 una vez más: ¡veamos qué cambia cuando traemos xlwings a la imagen!

### Estudio de caso (revisado): Informes de Excel

Ser capaces de editar realmente archivos de Excel a través de xlwings nos permite trabajar con archivos de plantilla que se conservarán al 100%, sin importar cuán complejos sean o en qué formato estén almacenados; por ejemplo, puede editar fácilmente un *xlsb* file, un caso que actualmente no es compatible con ninguno de los paquetes de escritor que conocimos en el capítulo anterior. Cuando mira *sales\_report\_openpxyl.py* en el repositorio complementario, verá que después de preparar el resumen DataFrame, tuvimos que escribir casi cuarenta líneas de código para crear un gráfico y diseñar un DataFrame con OpenPyXL. Con xlwings, logra lo mismo en solo seis líneas de código, como se muestra en [Ejemplo 9-1](#). Ser capaz de manejar el formateo en la plantilla de Excel le ahorrará mucho trabajo. Sin embargo, esto tiene un precio: xlwings requiere una instalación de Excel para ejecutarse; por lo general, está bien si tiene que crear estos informes con poca frecuencia en su propia máquina, pero puede ser menos ideal si intenta crear informes en un servidor como parte de una aplicación web.

En primer lugar, debe asegurarse de que su licencia de Microsoft Office cubra la instalación en un servidor y, en segundo lugar, Excel no se creó para la automatización desatendida, lo que significa que puede tener problemas de estabilidad, especialmente si necesita generar muchos informes en un poco tiempo. Dicho esto, he visto a más de un cliente hacer esto con éxito, por lo que si no puede usar un paquete de escritura por cualquier motivo, ejecutar xlwings en un servidor puede ser una opción que valga la pena explorar. Solo asegúrese de ejecutar cada script en una nueva instancia de Excel a través de `deaplication = xw.App()` para sortear los problemas típicos de estabilidad.

Encontrará el script xlwings completo en `sales_report_xlwings.py` en el repositorio complementario (la primera mitad es la misma que usamos con OpenPyXL y XlsxWriter). También es un ejemplo perfecto para combinar un paquete de lector con xlwings: mientras que pandas (a través de OpenPyXL y xlrd) es más rápido al leer muchos archivos del disco, xlwings hace que sea más fácil completar una plantilla preformateada.

#### *Ejemplo 9-1. sales\_report\_xlwings.py (solo segunda parte)*

```
# Abra la plantilla, pegue los datos, ajuste automáticamente las columnas
# y ajuste la fuente del gráfico. Luego guárdelo con un nombre diferente.
plantilla = xw.Libro(this_dir / "SG" / "sales_report_template.xlsx")hoja =
plantilla.hojas["Hoja1"]hoja["B3"].valor = resumen

hoja["B3"].expandir().columnas.autoajuste()
hoja.gráficos["Gráfico 1"].set_source_data(hoja["B3"].expandir() [-1,:-1])plantilla.
ahorrar(this_dir / "sales_report_xlwings.xlsx")
```

Cuando ejecute este script por primera vez en macOS (por ejemplo, abriéndolo en VS Code y haciendo clic en el botón Ejecutar archivo), tendrá que confirmar nuevamente una ventana emergente para otorgar acceso al sistema de archivos, algo que ya hemos encontrado anteriormente en este capítulo.

Con las plantillas de Excel formateadas, puede crear hermosos informes de Excel muy rápidamente. También obtienes acceso a métodos como autoajuste algo que no está disponible con los paquetes de escritor, ya que se basa en cálculos realizados por la aplicación Excel: esto le permite establecer correctamente el ancho y alto de sus celdas de acuerdo con su contenido.

**Figura 9-4.** muestra la parte superior del informe de ventas generado por xlwings con un encabezado de tabla personalizado, así como columnas donde autoajuste se ha aplicado el método.

Cuando empiezas a usar xlwings para algo más que rellenar un par de celdas en una plantilla, es bueno saber un poco sobre sus aspectos internos: la siguiente sección analiza cómo funciona xlwings bajo el capó.

	A	B	C	D	E	F	G	H	I
1	<h2>Sales Report</h2>								
2									
3	Month	Washington DC	Boston	Las Vegas	New York	Chicago	San Francisco	Total	
4	Jan 19	14,058	21,784	23,013	49,873	51,188	58,630	218,545	
5	Feb 19	15,235	21,455	25,493	46,670	52,331	55,219	216,403	
6	Mar 19	14,177	20,043	23,451	41,572	48,897	52,713	200,854	
7	Apr 19	13,339	18,791	22,710	41,714	47,396	49,325	193,276	
8	May 19	13,147	18,037	21,527	40,610	45,117	47,760	186,197	
9	Jun 19	14,284	21,556	21,985	47,266	49,460	53,462	208,014	
10	Jul 19	14,162	19,853	23,444	40,408	47,994	50,182	196,043	
11	Aug 19	16,127	22,333	24,928	45,397	50,839	55,336	214,960	
12	Sep 19	14,994	19,925	24,411	42,831	49,096	49,931	201,188	
13	Oct 19	12,847	16,551	22,828	34,090	42,544	44,312	173,171	
14	Nov 19	14,058	21,313	24,860	46,960	52,012	55,056	214,259	
15	Dec 19	14,702	19,723	24,536	42,364	49,355	50,933	201,613	
16	Total	171,130	241,363	283,185	519,755	586,229	622,859	2,424,521	

Figura 9-4. La tabla del informe de ventas basada en una plantilla preformatoada

## Temas avanzados de xlwings

Esta sección le muestra cómo hacer que su código xlwings funcione y cómo solucionar la falta de funcionalidad. Sin embargo, para comprender estos temas, primero debemos decir algunas palabras sobre la forma en que xlwings se comunica con Excel.

### Fundaciones xlwings

xlwings depende de otros paquetes de Python para comunicarse con el mecanismo de automatización del sistema operativo respectivo:

#### Ventanas

En Windows, xlwings se basa en la tecnología COM, abreviatura de *Modelo de objeto componente*. COM es un estándar que permite que dos procesos se comuniquen entre sí, en nuestro caso Excel y Python. xlwings usa el paquete Python [pywin32](#) para manejar las llamadas COM.

#### Mac OS

En macOS, xlwings se basa en *AppleScript*. AppleScript es el lenguaje de secuencias de comandos de Apple para automatizar aplicaciones con secuencias de comandos; afortunadamente, Excel es una aplicación con secuencias de comandos. Para ejecutar comandos de AppleScript, xlwings usa el paquete Python [applescript](#).

## Windows: cómo prevenir procesos zombies

Cuando juegue con xlwings en Windows, a veces notará que Excel parece estar completamente cerrado, sin embargo, cuando abra el Administrador de tareas (haga clic con el botón derecho en la barra de tareas de Windows, luego seleccione Administrador de tareas), verá Microsoft Excel en Procesos en segundo plano en la pestaña Procesos. Si no ve ninguna pestaña, primero haga clic en "Más detalles". Alternativamente, vaya a la pestaña Detalles, donde verá Excel listado como "EXCEL.EXE". Para finalizar un proceso zombie, haga clic con el botón derecho en la fila correspondiente y seleccione "Finalizar tarea" para forzar el cierre de Excel.

Porque estos procesos son *muertos vivientes* en lugar de terminarse correctamente, a menudo se les llama *procesos zombies*. Dejarlos alrededor consume recursos y puede llevar a comportamientos no deseados: por ejemplo, los archivos pueden estar bloqueados o los complementos pueden no cargarse correctamente cuando abre una nueva instancia de Excel. La razón por la que Excel a veces no logra cerrarse correctamente es que los procesos solo pueden terminarse una vez que no hay más referencias COM, por ejemplo, en forma de xlwingsaplicación objeto. Lo más común es que termine con un proceso zombie de Excel después de matar al intérprete de Python, ya que esto le impide limpiar correctamente las referencias COM. Considere este ejemplo en un indicador de Anaconda:

```
(base)> pitón
>>> importar xlwings como xw
>>> aplicación = xw.App()
```

Una vez que se esté ejecutando la nueva instancia de Excel, ciérrala nuevamente a través de la interfaz de usuario de Excel: mientras Excel se cierra, el proceso de Excel en el Administrador de tareas seguirá ejecutándose. Si cierra la sesión de Python correctamente ejecutandodejar() o usando el atajo Ctrl + Z, el proceso de Excel eventualmente se cerrará. Sin embargo, si mata el indicador Anaconda haciendo clic en la "x" en la parte superior derecha de la ventana, notará que el proceso se queda como un proceso zombi. Lo mismo sucede si mata el indicador Anaconda antes de cerrar Excel o si lo mata mientras está ejecutando un servidor Jupyter y mantiene un xlwingsaplicación objeto en una de las celdas del cuaderno de Jupyter. Para minimizar las posibilidades de terminar con procesos zombie de Excel, aquí hay algunas sugerencias:

- Correr app.quit () desde Python en lugar de cerrar Excel manualmente. Esto asegura que las referencias se limpien correctamente.
- No elimine las sesiones interactivas de Python cuando trabaje con xlwings, por ejemplo, si ejecuta un REPL de Python en un indicador de Anaconda, apague el intérprete de Python correctamente ejecutando dejar() o usando el atajo Ctrl + Z. Cuando trabaje con portátiles Jupyter, apague el servidor haciendo clic en Salir en la interfaz web.
- Con las sesiones interactivas de Python, ayuda a evitar el uso de aplicación objeto directamente, por ejemplo, usando xw.Book () en lugar de myapp.books.add (). Esto debería terminar correctamente Excel incluso si se mata el proceso de Python.

Ahora que tiene una idea sobre la tecnología subyacente de xlwings, ¡veamos cómo podemos acelerar los scripts lentos!

## Mejorando el desempeño

Para mantener el rendimiento de sus scripts xlwings, existen algunas estrategias: la más importante es mantener las llamadas entre aplicaciones al mínimo absoluto. El uso de valores brutos puede ser otra opción y, finalmente, configurar la aplicación las propiedades también pueden ayudar.

¡Repasemos estas opciones una tras otra!

### Minimice las llamadas entre aplicaciones

Es crucial saber que cada llamada entre aplicaciones desde Python a Excel es "cara", es decir, lenta. Por lo tanto, estas llamadas deben reducirse tanto como sea posible. La forma más fácil de hacer esto es leyendo y escribiendo rangos completos de Excel en lugar de recorrer celdas individuales. En el siguiente ejemplo, leemos y escribimos 150 celdas, primero recorriendo cada celda y luego tratando con todo el rango en una llamada:

```
En [64]: # Agregue una nueva hoja y escriba 150 valores
# para tener algo con lo que trabajar
hoja2 = libro.hojas.agregar()
hoja2["A1"].valor = notario.público.orange(150).remodelar(30, 5)
```

```
En [sesenta y cinco]: %%tiempo
# Esto hace 150 llamadas entre aplicaciones
por celda en hoja2["A1: E30"]:
    celda.valor += 1
```

Tiempo de pared: 909 ms

```
En [66]: %%tiempo
# Esto hace solo dos llamadas entre aplicaciones
valores = hoja2["A1: E30"].opciones(notario.público.formación).hoja de valor 2[
    "A1"].valor = valores + 1
```

Tiempo de pared: 97,2 ms

Estos números son aún más extremos en macOS, donde la segunda opción es aproximadamente 50 veces más rápida que la primera en mi máquina.

### Valores brutos

xlwings se diseñó principalmente con un enfoque en la conveniencia más que en la velocidad. Sin embargo, si trabaja con rangos de celdas enormes, puede encontrarse con situaciones en las que puede ahorrar tiempo omitiendo el paso de limpieza de datos de xlwings: xlwings recorre cada valor cuando lee y escribe datos, por ejemplo, para alinear tipos de datos entre Windows y macOS. Usando la celdacrud como convertidor en el opciones método, omite este paso. Si bien esto debería hacer que todas las operaciones sean más rápidas, es posible que la diferencia no sea significativa a menos que escriba matrices grandes en Windows. Sin embargo, el uso de valores sin procesar significa que ya no puede trabajar directamente con DataFrames. En cambio, necesitas

proporcione sus valores como listas anidadas o tuplas. Además, deberá proporcionar la dirección completa del rango en el que está escribiendo, siempre que la celda superior izquierda ya no sea suficiente:

```
En [67]: # Con valores brutos, debe proporcionar el
# rango objetivo, la hoja ["A35"] ya no funciona
hoja1["A35: B36"]
    .opciones("crudo").valor = [[1, 2], [3, 4]]
```

#### Propiedades de la aplicación

Dependiendo del contenido de su libro de trabajo, cambiar las propiedades de su aplicación Los objetos también pueden ayudar a que el código se ejecute más rápido. Por lo general, desea ver las siguientes propiedades (myapp es un xlwings aplicación objeto):

- myapp.screen\_updating = Falso
- myapp.calculation = "manual"
- myapp.display\_alerts = Falso

Al final de la secuencia de comandos, asegúrese de volver a establecer los atributos en su estado original. Si está en Windows, también puede ver una ligera mejora en el rendimiento al ejecutar su script en una instancia oculta de Excel a través dexw.App (visible = Falso).

Ahora que sabe cómo mantener el rendimiento bajo control, echemos un vistazo a cómo ampliar la funcionalidad de xlwings.

## Cómo solucionar la falta de funcionalidad

xlwings proporciona una interfaz Pythonic para los comandos de Excel más utilizados y los hace funcionar en Windows y macOS. Sin embargo, existen muchos métodos y atributos del modelo de objetos de Excel que aún no están cubiertos de forma nativa por xlwings, ¡pero no todo está perdido! xlwings le da acceso al objeto pywin32 subyacente en Windows y al objeto appscript en macOS mediante el uso de api atributo en cualquier objeto xlwings. De esta manera, tiene acceso a todo el modelo de objetos de Excel, pero a su vez, pierde la compatibilidad multiplataforma. Por ejemplo, suponga que desea borrar el formato de una celda. Así es como lo haría:

- Compruebe si el método está disponible en xlwings distancia objeto, por ejemplo, utilizando la tecla Tab después de poner un punto al final de un distancia objeto en un cuaderno de Jupyter, ejecutando dir (hoja ["A1"]) o buscando en el [Referencia de la API de xlwings](#). En VS Code, los métodos disponibles deben mostrarse automáticamente en una descripción emergente.
- Si falta la funcionalidad deseada, utilice el api atributo para obtener el objeto subyacente: en Windows, hoja ["A1"].api le dará un objeto pywin32 y en macOS, obtendrá un objeto appscript.

- Verifique el modelo de objetos de Excel en el [Referencia de Excel VBA](#). Para borrar el formato de un rango, terminaría bajo `Range.ClearFormats`.
- En Windows, en la mayoría de los casos, puede utilizar el método o la propiedad VBA directamente con su api objeto. Si es un método, asegúrese de agregar paréntesis en Python:  
hoja ["A1"]. api.ClearFormats (). Si está haciendo esto en macOS, las cosas son más complicadas ya que Appscript usa una sintaxis que puede ser difícil de adivinar. Su mejor enfoque es consultar la guía para desarrolladores que forma parte del [código fuente de xlwings](#). Sin embargo, borrar el formato de la celda es bastante fácil: simplemente aplique las reglas de sintaxis de Python en el nombre del método usando caracteres en minúscula con debajo puntuaciones: hoja ["A1"]. api.clear\_formats () .

Si necesita asegurarse de que ClearFormats funciona en ambas plataformas, puede hacerlo de la siguiente manera (darwin es el núcleo de macOS y lo utiliza como su nombre sys.platform):

```
importar sys
si sys.plataforma.comienza con("darwin"):
    hoja["A10"].api.clear_formats()elif
sys.plataforma.comienza con("ganar"):
    hoja["A10"].api.ClearFormats()
```

En cualquier caso, vale la pena abrir un problema en xlwings '[Repositorio de GitHub](#)' para tener la funcionalidad incluida en una versión futura.

## Conclusión

Este capítulo le presentó el concepto de automatización de Excel: a través de xlwings, puede usar Python para tareas que tradicionalmente haría en VBA. Aprendimos sobre el modelo de objetos de Excel y cómo xlwings le permite interactuar con sus componentes como el hoja y distancia objetos. Equipados con este conocimiento, volvimos al estudio de caso de informes de [Capítulo 7](#) y usó xlwings para completar una plantilla de informe preformatoada; esto le mostró que hay un caso para usar los paquetes de lector y xlwings uno al lado del otro. También aprendimos sobre las bibliotecas que xlwings usa bajo el capó para comprender cómo podemos mejorar el rendimiento y solucionar la funcionalidad faltante. Mi característica favorita de xlwings es que funciona igual de bien en macOS que en Windows. Esto es aún más emocionante ya que Power Query en macOS aún no tiene todas las características de la versión de Windows: lo que falte, debería poder reemplazarlo fácilmente con una combinación de pandas y xlwings.

Ahora que conoce los conceptos básicos de xlwings, está listo para el siguiente capítulo: allí, daremos el siguiente paso y llamaremos a los scripts de xlwings desde el propio Excel, lo que le permitirá crear herramientas de Excel que funcionan con Python.

### Herramientas de Excel impulsadas por Python

En el último capítulo, aprendimos cómo escribir scripts de Python para automatizar Microsoft Excel. Si bien esto es muy poderoso, el usuario debe sentirse cómodo usando Anaconda Prompt o un editor como VS Code para ejecutar los scripts. Lo más probable es que este no sea el caso si sus herramientas son utilizadas por usuarios comerciales. Para ellos, querrá ocultar la parte de Python para que la herramienta de Excel se sienta como un libro de trabajo normal habilitado para macros nuevamente. Cómo lograr eso con xlwings es el tema de este capítulo. Comenzaré mostrándole la ruta más corta para ejecutar código Python desde Excel antes de analizar los desafíos de implementar herramientas de xlwings; esto también nos permitirá tener una visión más detallada de las configuraciones disponibles que ofrece xlwings. Al igual que el último capítulo, este capítulo requiere que tenga una instalación de Microsoft Excel en Windows o macOS.

### Usando Excel como Frontend con xlwings

los *Interfaz* es la parte de una aplicación que el usuario ve y con la que interactúa. Otros nombres comunes para frontend son *interfaz gráfica del usuario* (GUI) o simplemente *interfaz de usuario* (Interfaz de usuario). Cuando les pregunto a los usuarios de xlwings por qué están creando su herramienta con Excel en lugar de crear una aplicación web moderna, lo que suelo escuchar es esto: "Excel es la interfaz con la que nuestros usuarios están familiarizados". Confiar en las celdas de la hoja de cálculo permite a los usuarios proporcionar entradas de forma rápida e intuitiva, lo que las hace a menudo más productivas que si tuvieran que usar una interfaz web a medio hacer. Comenzaré esta sección presentándole el complemento de Excel xlwings y la CLI (interfaz de línea de comandos) de xlwings antes de crear nuestro primer proyecto a través del inicio rápido mando. Concluiré esta sección mostrándole dos formas de llamar al código Python desde Excel: haciendo clic en el botón Ejecutar principal en el complemento y usando el RunPython función en VBA. ¡Comencemos instalando el complemento de Excel xlwings!

## Complemento de Excel

Dado que xlwings está incluido en la distribución de Anaconda, en el capítulo anterior, pudimos ejecutar comandos xlwings en Python de inmediato. Sin embargo, si desea llamar a los scripts de Python desde Excel, debe instalar el complemento de Excel o configurar el libro en el modo independiente. Si bien presentaré el modo independiente en "["Implementación" en la página 218](#)", esta sección le muestra cómo trabajar con el complemento. Para instalarlo, ejecute lo siguiente en un indicador de Anaconda:

```
(base)> Instalación del complemento xlwings
```

Deberá mantener sincronizadas la versión del paquete de Python y la versión del complemento cada vez que actualice xlwings. Por lo tanto, siempre debe ejecutar dos comandos cuando actualice xlwings: uno para el paquete de Python y otro para el complemento de Excel. Dependiendo de si usa el administrador de paquetes Conda o pip, así es como actualiza su instalación de xlwings:

*Conda (use esto con la distribución Anaconda Python)*

```
(base)> actualización de conda xlwings(base)>  
Instalación del complemento xlwings
```

*pip (use esto con cualquier otra distribución de Python)*

```
(base)> instalar pip - actualizar xlwings(base)>  
Instalación del complemento xlwings
```



### Software antivirus

Desafortunadamente, el complemento xlwings a veces se marca como un complemento malicioso por parte del software antivirus, especialmente si está utilizando una versión nueva. Si esto sucede en su máquina, vaya a la configuración de su software antivirus, donde debería poder marcar xlwings como seguro de ejecutar. Por lo general, también es posible informar esos falsos positivos a través de la página de inicio del software.

Cuando escribes xlwings en un indicador de Anaconda, está utilizando la CLI de xlwings. Además de facilitar la instalación del complemento xlwings, ofrece algunos comandos más: los introduciré cuando los necesitemos, pero siempre puedes escribir xlwings en un indicador de Anaconda y presione Enter para imprimir las opciones disponibles. Ahora echemos un vistazo más de cerca a lo que *Instalación del complemento xlwings* lo hace:

### Instalación

La instalación real del complemento se realiza copiando *xlwings.xlam* desde el directorio del paquete de Python en Excel *XLSTART* carpeta, que es una carpeta especial: Excel abrirá todos los archivos que se encuentran en esta carpeta cada vez que inicie Excel. Cuando corres este comando del complemento xlwings en un indicador de Anaconda, se imprimirá

donde el `XLSTART` el directorio está en su sistema y si el complemento está instalado o no.

### Configuración

Cuando instale el complemento por primera vez, también lo configurará para usar el intérprete de Python o el entorno Conda desde donde está ejecutando el Instalar en pc comando: como ves en Figura 10-1, los valores para Camino de Conda y Conda Env se completan automáticamente mediante la CLI de xlwings.<sup>1</sup> Estos valores se almacenan en un archivo llamado `xlwings.conf` en el `.xlwings` carpeta en su directorio personal. En Windows, esto suele ser `C:\Users\<nombre de usuario>\.xlwings\xlwings.conf` y en macOS `/Users/<nombre de usuario>/\.xlwings/xlwings.conf`. En macOS, las carpetas y los archivos con un punto inicial están ocultos de forma predeterminada. Cuando esté en el Finder, escriba el atajo del teclado Command-Shift-. para alternar su visibilidad.

Después de ejecutar el comando de instalación, tendrá que reiniciar Excel para ver la pestaña xlwings en la cinta como se muestra en Figura 10-1.

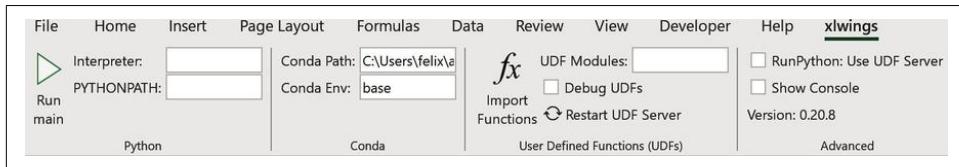


Figura 10-1. El complemento de cinta xlwings después de ejecutar el comando de instalación



### El complemento Ribbon en macOS

En macOS, la cinta se ve un poco diferente ya que le faltan las secciones sobre funciones definidas por el usuario y Conda: mientras que las funciones definidas por el usuario no son compatibles con macOS, los entornos Conda no requieren un tratamiento especial, es decir, están configurados como intérprete bajo el grupo Python.

Ahora que tiene instalado el complemento xlwings, necesitaremos un libro de trabajo y algo de código Python para probarlo. La forma más rápida de llegar es utilizando el `Inicio rápido` comando, como te mostraré a continuación.

<sup>1</sup> Si está en macOS o usa una distribución de Python que no sea Anaconda, configurará el intérprete en lugar de la configuración de Conda.

## Comando de inicio rápido

Para que la creación de su primera herramienta xlwings sea lo más fácil posible, la CLI de xlwings ofrece la Inicio rápido mando. En un indicador de Anaconda, use el CD comando para cambiar al directorio donde desea crear su primer proyecto (por ejemplo, CD escritorio), luego ejecute lo siguiente para crear un proyecto con el nombre primer\_proyecto:

```
(base)> primer_proyecto de inicio rápido de xlwings
```

El nombre del proyecto debe ser un nombre de módulo de Python válido: puede contener caracteres, números y guiones bajos, pero no espacios ni guiones, y no debe comenzar con un número. Te mostraré debajo “[Función RunPython](#)” en la página 213 cómo puede cambiar el nombre del archivo de Excel a algo que no tenga que seguir estas reglas. Ejecutando el Inicio rápido comando creará una carpeta llamada *primer\_proyecto* en su directorio actual. Cuando lo abra en el Explorador de archivos en Windows o el Finder en macOS, verá dos archivos: *first\_project.xlsxm* y *first\_project.py*. Abra ambos archivos, el archivo de Excel en Excel y el archivo de Python en VS Code. La forma más sencilla de ejecutar el código Python desde Excel es utilizando el botón Ejecutar principal en el complemento. ¡Veamos cómo funciona!

## Ejecutar principal

Antes de mirar *first\_project.py* con más detalle, continúe y haga clic en el botón Ejecutar principal en el extremo izquierdo del complemento xlwings mientras *first\_project.xlsxm* es su archivo activo; escribirá “¡Hola xlwings!” en la celda A1 de la primera hoja. Vuelva a hacer clic en el botón y cambiará a “¡Adiós xlwings!” ¡Felicitaciones, acaba de ejecutar su primera función de Python desde Excel! Después de todo, eso no fue mucho más difícil que escribir una macro VBA, ¿verdad? Ahora echemos un vistazo a *first\_project.py* en [Ejemplo 10-1](#).

### Ejemplo 10-1. *first\_project.py*

```
importar xlwings como xw
```

```
def principal():
    wb = xw.Libro.llamador ①
    (hoja = wb.hojas[0]
    si hoja["A1"].valor == "¡Hola xlwings!":
        hoja["A1"].valor = "¡Adiós xlwings!"
    demás:
        hoja["A1"].valor = "¡Hola xlwings!"

@ xw.func ②
def Hola(nombre):
    regreso F"¡Hola, {nombre}!"
```

```
si __nombre__ == "__principal__": ❸  
    xw.Libro("primer_proyecto.xlsx").set_mock_caller()  
    principal()
```

- ❶ `xw.Book.caller()` es un xlwings libro objeto que hace referencia al libro de Excel que está activo cuando hace clic en el botón principal Ejecutar. En nuestro caso, corresponde a `xw.Book ("primer_proyecto.xlsx")`. Utilizando `xw.Book.caller()` Te permite cambie el nombre y mueva su archivo de Excel en el sistema de archivos sin romper la referencia. También se asegura de que esté manipulando el libro de trabajo correcto si lo tiene abierto en varias instancias de Excel.
- ❷ En este capítulo, ignoraremos la función Hola ya que este será el tema de [Capítulo 12](#). Si ejecuta el Inicio rápido comando en macOS, no verá el Hola funcionar de todos modos, ya que las funciones definidas por el usuario solo son compatibles con Windows.
- ❸ Explicaré las últimas tres líneas cuando veamos la depuración en el próximo capítulo. A los efectos de este capítulo, ignore o incluso elimine todo lo que esté debajo de la primera función.

El botón principal Ejecutar en el complemento de Excel es una característica conveniente: le permite llamar a una función con el nombre principal en un módulo de Python que tiene el mismo nombre que el archivo de Excel sin tener que agregar un botón primero a su libro de trabajo. Incluso funcionará si guarda su libro de trabajo sin macros.xlsx formato. Sin embargo, si desea llamar a una o más funciones de Python que no se llaman principal y no son parte de un módulo con el mismo nombre que el libro de trabajo, debe usar el RunPython función de VBA en su lugar. ¡La siguiente sección tiene los detalles!

## Función RunPython

Si necesita más control sobre cómo llama a su código Python, use la función VBA RunPython. Como consecuencia, RunPython requiere que su libro de trabajo se guarde como un libro de trabajo habilitado para macros.



### Habilitar macros

Debe hacer clic en Habilitar contenido (Windows) o Habilitar macros (macOS) cuando abra un libro habilitado para macros (.xlsm extensión) como la que genera el Inicio rápido comando. En Windows, cuando trabaja con .xlsm archivos del repositorio complementario, también debe hacer clic en Habilitar edición o Excel no abrirá los archivos que se descargan de Internet correctamente.

RunPython acepta una cadena con código Python: por lo general, importa un módulo Python y ejecuta una de sus funciones. Cuando abra el editor de VBA a través de Alt + F11 (Windows) u Opción-F11 (macOS), verá que el Inicio rápido comando agrega una macro llamada SampleCall en un módulo VBA con el nombre "Module1" (ver Figura 10-2). Si no ves el SampleCall, haga doble clic en Module1 en el árbol del proyecto de VBA en el lado izquierdo.

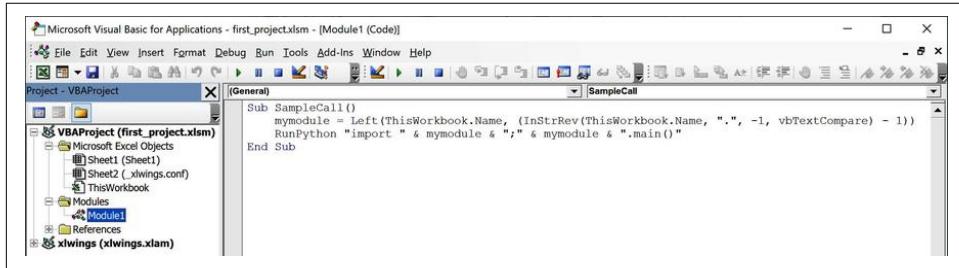


Figura 10-2. El editor de VBA que muestra Module1

El código parece un poco complicado, pero esto es solo para que funcione dinámicamente sin importar el nombre del proyecto que elija al ejecutar el Inicio rápido mando. Como se llama nuestro módulo de Python primer\_proyecto, podría reemplazar el código con el siguiente equivalente fácil de entender:

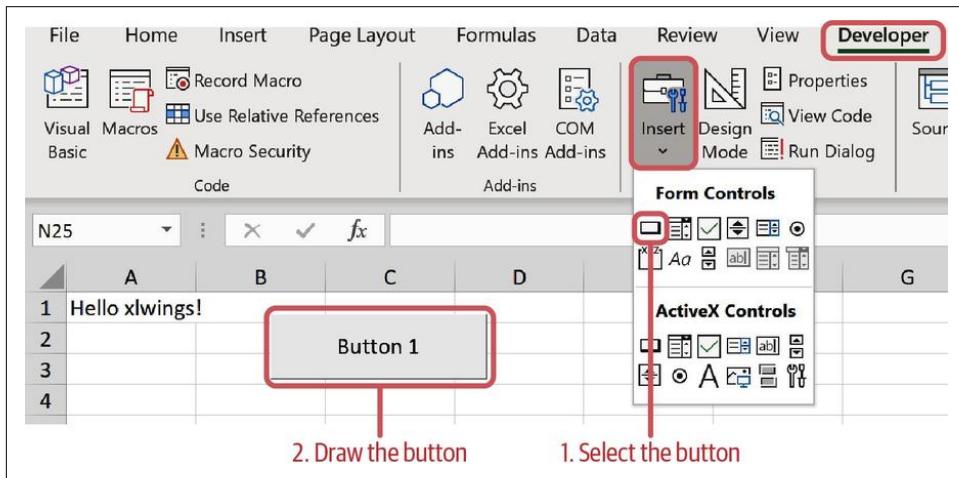
```

Sub SampleCall()
    RunPython "importar primer_proyecto; primer_proyecto.main ()" End
Sub

```

Dado que no es divertido escribir cadenas de varias líneas en VBA, usamos un punto y coma que Python acepta en lugar de un salto de línea. Hay un par de formas en las que puede ejecutar este código: por ejemplo, mientras está en el editor de VBA, coloque el cursor en cualquier línea del SampleCall macro y presione F5. Sin embargo, normalmente ejecutará el código desde una hoja de Excel y no desde el editor de VBA. Por lo tanto, cierre el editor de VBA y vuelva al libro de trabajo. Al escribir Alt + F8 (Windows) u Opción-F8 (macOS), aparecerá el menú de macros: seleccione SampleCall y haga clic en el botón Ejecutar. O, para hacerlo más fácil de usar, agregue un botón a su libro de Excel y conéctelo con el SampleCall: Primero, asegúrese de que se muestre la pestaña Desarrollador en la cinta. Si no es así, ve a Archivo > Opciones > Personalizar cinta y active la casilla de verificación junto a Desarrollador (en macOS, la encontrará en Excel > Preferencias > Cinta y barra de herramientas). Para insertar un botón, vaya a la pestaña Desarrollador y en el grupo Controles, haga clic en Insertar > Botón (en Controles de formulario). En macOS, se le presentará el botón sin tener que ir primero a Insertar. Cuando haces clic en el ícono del botón, tu cursor se convierte en una pequeña cruz: úsalo para dibujar un botón en tu hoja manteniendo presionado el botón izquierdo del mouse mientras dibujas una forma rectangular. Una vez que suelte el botón del mouse, aparecerá el menú Asignar macro; seleccione el

SampleCall y haga clic en Aceptar. Haga clic en el botón que acaba de crear (en mi caso, se llama "Botón 1") y ejecutará nuestro principal funcionar de nuevo, como en [Figura 10-3](#).



*Figura 10-3. Dibujar un botón en una hoja*



Controles de formulario frente a controles ActiveX

En Windows, tiene dos tipos de controles: controles de formulario y controles ActiveX. Si bien puede usar un botón de cualquier grupo para conectarse a su SampleCall macro, solo el de los controles de formulario también funcionará en macOS. En el próximo capítulo, usaremos Rectángulos como botones para que se vean un poco más modernos.

Ahora echemos un vistazo a cómo podemos cambiar los nombres predeterminados que fueron asignados por el Inicio rápido comando: vuelva a su archivo Python y cámbiele el nombre de `first_project.py` para `hola.py`. Además, cambie el nombre de suprincipal funcionar en Hola Mundo. Asegúrese de guardar el archivo, luego abra el editor VBA nuevamente a través de Alt + F11 (Windows) u Opción-F11 (macOS) y edite SampleCall de la siguiente manera para reflejar los cambios:

```
Sub SampleCall()
    RunPython "import hola; hola.hello_world ()"End
Sub
```

De vuelta en la hoja, haga clic en el "Botón 1" para asegurarse de que todo sigue funcionando. Por último, es posible que también desee mantener la secuencia de comandos de Python y el archivo de Excel en dos directorios diferentes. Para comprender las implicaciones de esto, primero tendré que decir una palabra sobre Python *ruta de búsqueda del módulo*: si importa un módulo en su código, Python lo busca en varios directorios. Primero, Python comprueba si hay un módulo incorporado con este nombre y, si no encuentra uno, pasa a buscar en el directorio de trabajo actual y en los directorios proporcionados por el llamado PYTHONPATH. xlwings

agrega automáticamente el directorio del libro de trabajo al PYTHONPATH y le permite agregar rutas adicionales a través del complemento. Para probar esto, tome la secuencia de comandos de Python que ahora se llama `hola.py` muévelo a una carpeta llamada `pyscripts` que crea en su directorio de inicio: en mi caso, esto sería `C:\ Usuarios\ felix\ pyscripts` en Windows o `/Usuarios / felix / pyscripts` en macOS. Cuando vuelva a hacer clic en el botón, aparecerá el siguiente error en una ventana emergente:

```
Rastreo (llamadas recientes más última):
Archivo "<string>", línea 1, en <module>
ModuleNotFoundError: Ningún módulo llamado 'first_project'
```

Para solucionar esto, simplemente agregue la ruta del `pyscripts` directorio al PYTHONPATH en su cinta xlwings, como en [Figura 10-4..](#) Cuando haga clic en el botón una vez más, volverá a funcionar.

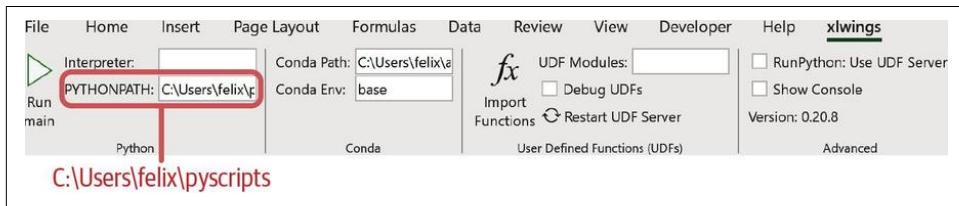


Figura 10-4. La configuración de PYTHONPATH

Lo que no he mencionado todavía es el nombre del libro de Excel: una vez que RunPython La llamada a la función usa un nombre de módulo explícito como `primer_proyecto` en lugar del código que fue agregado por Inicio rápido, puede cambiar el nombre de su libro de Excel como deseé.

Confíando en el Inicio rápido comando es la forma más fácil si inicia un nuevo proyecto xlwings. Sin embargo, si tiene un libro de trabajo existente, es posible que prefiera configurarlo manualmente. ¡Véamos cómo se hace!

### RunPython sin comando de inicio rápido

Si desea utilizar el RunPython función con un libro de trabajo existente que no fue creado por el Inicio rápido comando, debe encargarse manualmente de las cosas que el Inicio rápido el comando lo hace por usted de otra manera. Tenga en cuenta que los siguientes pasos solo son necesarios para RunPython llamar pero no cuando desee utilizar el botón Ejecutar principal:

1. En primer lugar, asegúrese de guardar su libro de trabajo como un libro de trabajo habilitado para macros con el `xlsm` o `xlsb` extensión.
2. Agregue un módulo VBA; Para hacerlo, abra el editor de VBA a través de Alt + F11 (Windows) u Opción-F11 (macOS) y asegúrese de seleccionar el VBAProject de su libro de trabajo en la vista de árbol en el lado izquierdo, luego haga clic con el botón derecho en él y elija Insertar > Módulo, como en [Figura 10-5](#). Esto insertará un módulo VBA vacío donde puede escribir su macro VBA con el RunPython llama.

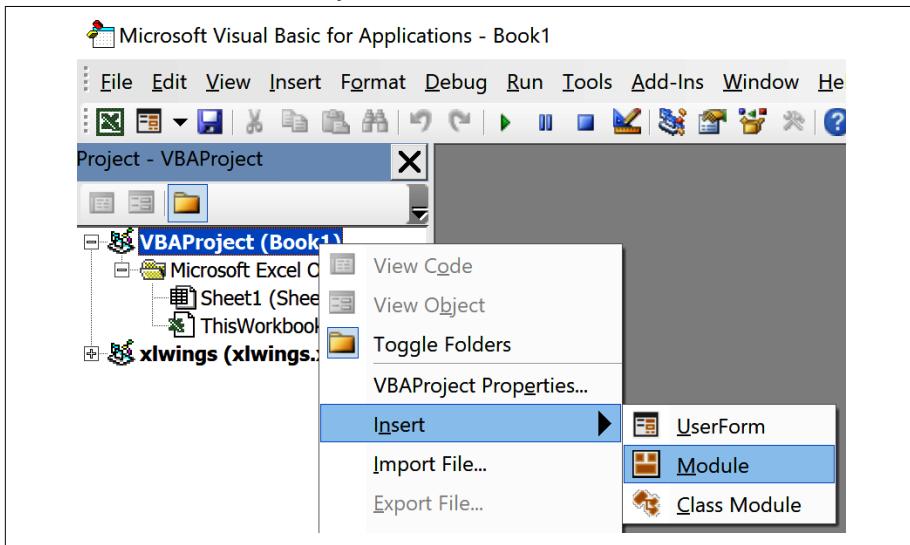


Figura 10-5. Agregar un módulo VBA

3. Agregue una referencia a xlwings: RunPython es una función que forma parte del complemento xlwings. Para usarlo, deberá asegurarse de tener un conjunto de referencia para xlwings en su proyecto VBA. Nuevamente, comience seleccionando el libro de trabajo correcto en la vista de árbol en el lado izquierdo del editor de VBA, luego vaya a Herramientas > Referencia y active la casilla de verificación para xlwings, como se ve en [Figura 10-6](#).

Su libro de trabajo ahora está listo para ser utilizado con el RunPython llama de nuevo. Una vez que todo funciona en su máquina, el siguiente paso suele ser hacer que funcione en la máquina de su colega. ¡Repasemos un par de opciones para facilitar esta parte!

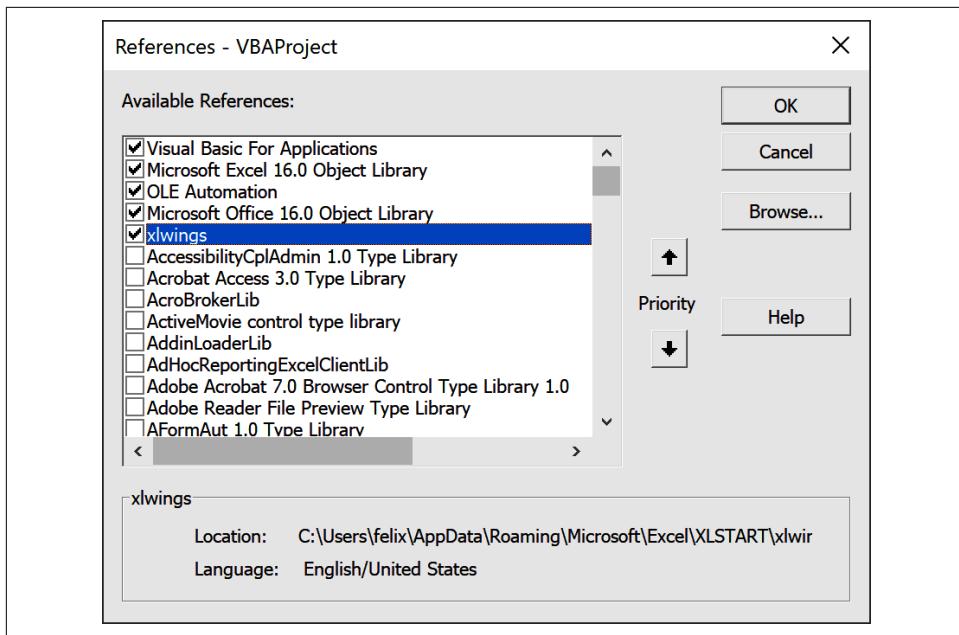


Figura 10-6. RunPython requiere una referencia a xlwings

## Despliegue

En el desarrollo de software, el término *despliegue* se refiere a distribuir e instalar software para que los usuarios finales puedan utilizarlo. En el caso de las herramientas xlwings, es útil saber qué dependencias son necesarias y qué configuraciones pueden facilitar la implementación. Comenzaré con la dependencia más importante, que es Python, antes de mirar los libros de trabajo que se han configurado en el modo independiente para deshacerme del complemento de Excel xlwings. Concluiré esta sección observando más de cerca cómo funciona la configuración con xlwings.

### Dependencia de Python

Para poder ejecutar herramientas xlwings, sus usuarios finales deben tener una instalación de Python. Pero el hecho de que todavía no tengan Python no significa que no haya formas de facilitar el proceso de instalación. Aquí hay un par de opciones:

#### *Anaconda o WinPython*

Indique a sus usuarios que descarguen e instalen la distribución de Anaconda. Para estar seguro, debe aceptar una versión específica de Anaconda para asegurarse de que estén usando las mismas versiones de los paquetes contenidos que está usando. Esta es una buena opción si solo usa paquetes que son parte de Anaconda. **WinPython** es una alternativa interesante a Anaconda, ya que se distribuye bajo el MIT

licencia de código abierto y también viene con xlwings preinstalado. Como sugiere el nombre, solo está disponible en Windows.

#### *Unidad compartida*

Si tiene acceso a una unidad compartida razonablemente rápida, es posible que pueda instalar Python directamente allí, lo que permitirá que todos usen las herramientas sin una instalación local de Python.

#### *Ejecutable congelado*

En Windows, xlwings le permite trabajar con *ejecutables congelados*, que son archivos con la .exe extensión que contiene Python y todas las dependencias. Un paquete popular para producir ejecutables congelados es [PyInstaller](#). Los ejecutables congelados tienen la ventaja de que solo empaquetan lo que su programa necesita y pueden producir un solo archivo, lo que puede facilitar la distribución. Para obtener más detalles sobre cómo trabajar con ejecutables congelados, eche un vistazo a [la xlwings docs](#). Tenga en cuenta que los ejecutables congelados no funcionarán cuando use xlwings para funciones definidas por el usuario, la funcionalidad que presentaré en [Capítulo 12](#).

Si bien Python es un requisito estricto, la instalación del complemento xlwings no lo es, como explicaré a continuación.

### **Libros de trabajo independientes: deshacerse del complemento xlwings**

En este capítulo, siempre nos hemos basado en el complemento xlwings para llamar al código Python, ya sea haciendo clic en el botón Ejecutar principal o usando el RunPython función. Incluso si la CLI de xlwings facilita la instalación del complemento, aún puede ser una molestia para los usuarios menos técnicos que no se sienten cómodos usando Anaconda Prompt. Además, dado que el complemento xlwings y el paquete xlwings Python deben tener la misma versión, es posible que surja un conflicto en el que sus destinatarios ya tengan instalado el complemento xlwings, pero con una versión diferente a la que requiere su herramienta. Sin embargo, existe una solución simple: xlwings no requiere el complemento de Excel y se puede configurar como un *libro de trabajo independiente* en lugar de. En este caso, el código VBA del complemento se almacena directamente en su libro de trabajo. Como de costumbre, la forma más sencilla de configurar todo es mediante el **Iniciar rápido comando**, esta vez con el -ser único bandera:

```
(base)> Inicio rápido de xlwings second_project --standalone
```

Cuando abres el generado *second\_project.xlsxm* libro de trabajo en Excel y presione Alt + F11 (Windows) u Opción-F11 (macOS), verá el xlwings módulo y el Diccionario módulo de clase que se requieren en lugar del complemento. Lo más importante es que un proyecto independiente ya no debe tener una referencia a xlwings. Si bien esto se configura automáticamente cuando se utiliza el -ser único flag, es importante que elimine la referencia en caso de que desee convertir un libro de trabajo existente: vaya a Herramientas> Referencias en su editor de VBA y desactive la casilla de verificación para xlwings.



#### Crear un complemento personalizado

Si bien esta sección le muestra cómo deshacerse de la dependencia del complemento xlwings, es posible que a veces desee crear su propio complemento para la implementación. Esto tiene sentido si desea utilizar las mismas macros con muchos libros de trabajo diferentes. Encontrará instrucciones sobre cómo crear su propio complemento personalizado en el [xlwings docs](#).

Habiendo tocado Python y el complemento, ahora echemos un vistazo más en profundidad a cómo funciona la configuración de xlwings.

### Jerarquía de configuración

Como se mencionó al principio de este capítulo, la cinta almacena su configuración en el directorio de inicio del usuario, en `.xlwings\xlwings.conf`. La configuración consta de individuo *ajustes*, como el `PYTHONPATH` que ya vimos al principio de este capítulo. La configuración que establezca en su complemento se puede anular en el nivel de directorio y libro de trabajo; xlwings busca la configuración en las siguientes ubicaciones y orden:

#### *Configuración del libro de trabajo*

Primero, xlwings busca una hoja llamada `xlwings.conf`. Esta es la forma recomendada de configurar su libro de trabajo para la implementación, ya que no tiene que manejar un archivo de configuración adicional. Cuando ejecuta el `Inicio rápido` comando, creará una configuración de muestra en una hoja llamada `_xlwings.conf`: elimine el subrayado inicial en el nombre para activarlo. Si no desea usarlo, no dude en eliminar la hoja.

#### *Configuración de directorio*

A continuación, xlwings busca un archivo llamado `xlwings.conf` en el mismo directorio que su libro de Excel.

#### *Configuración de usuario*

Finalmente, xlwings busca un archivo llamado `xlwings.conf` en el `.xlwings` carpeta en el directorio de inicio del usuario. Normalmente, no edita este archivo directamente; en su lugar, el complemento lo crea y edita cada vez que cambia una configuración.

Si xlwings no encuentra ninguna configuración en estas tres ubicaciones, vuelve a los valores predeterminados.

Cuando edita la configuración a través del complemento de Excel, se editarán automáticamente la `xlwings.conf` expediente. Si desea editar el archivo directamente, busque el formato exacto y la configuración disponible yendo a la [xlwings docs](#), pero a continuación señalaré las configuraciones más útiles en el contexto de la implementación.

## Ajustes

La configuración más crítica es sin duda el intérprete de Python; si su herramienta de Excel no puede encontrar el intérprete de Python correcto, nada funcionará. Los `PYTHONPATH` La configuración le permite controlar dónde coloca sus archivos fuente de Python, y la configuración Usar servidor UDF mantiene al intérprete de Python ejecutándose entre llamadas en Windows, lo que puede mejorar en gran medida el rendimiento.

### *Intérprete de Python*

`xlwings` se basa en una instalación de Python instalada localmente. Sin embargo, esto no significa necesariamente que el destinatario de su herramienta `xlwings` deba jugar con la configuración antes de poder usar la herramienta. Como se mencionó anteriormente, podría decirles que instalen la distribución de Anaconda con la configuración predeterminada, que la instalará en el directorio de inicio del usuario. Si utiliza *Variables de entorno* en su configuración, `xlwings` encontrará la ruta correcta al intérprete de Python. Una variable de entorno es un conjunto de variables en la computadora del usuario que permite a los programas consultar información específica de este entorno, como el nombre de la carpeta de inicio del usuario actual. Por ejemplo, en Windows, configure el `Conda`

Sendero para `%PERFIL DE USUARIO%\anaconda3` y en macOS, configure `Interpreter_Mac` para `$INICIO/opt/anaconda3/bin/python`. Estas rutas luego se resolverán dinámicamente a la ruta de instalación predeterminada de Anaconda.

### *PYTHONPATH*

De forma predeterminada, `xlwings` busca el archivo fuente de Python en el mismo directorio que el archivo de Excel. Es posible que esto no sea ideal cuando le da su herramienta a usuarios que no están familiarizados con Python, ya que podrían olvidarse de mantener los dos archivos juntos al mover el archivo de Excel. En su lugar, puede colocar sus archivos fuente de Python en una carpeta dedicada (esto podría estar en una unidad compartida) y agregar esta carpeta a la `PYTHONPATH` configuración. Alternativamente, también puede colocar sus archivos de origen en una ruta que ya es parte de la ruta de búsqueda del módulo de Python. Una forma de lograr esto sería distribuir su código fuente como un paquete de Python; instalarlo lo colocará en *el paquetes de sitio* directorio, donde Python encontrará su código. Para obtener más información sobre cómo crear un paquete de Python, consulte la [Guía del usuario de empaquetado de Python](#).

### *RunPython: use el servidor UDF (solo Windows)*

Es posible que haya notado que un `RunPython` la llamada puede ser bastante lenta. Esto se debe a que `xlwings` inicia un intérprete de Python, ejecuta el código de Python y finalmente apaga el intérprete nuevamente. Esto puede no ser tan malo durante el desarrollo, ya que se asegura de que todos los módulos se carguen desde cero cada vez que llame al `RunPython` mando. Sin embargo, una vez que su código sea estable, es posible que desee activar la casilla de verificación "RunPython: Usar servidor UDF" que solo está disponible en Windows. Esto usará el mismo servidor Python que usan las funciones definidas por el usuario (el tema de [Capítulo 12](#)) y mantener la sesión de Python ejecutándose entre llamadas, que serán

mucho más rápido. Sin embargo, tenga en cuenta que debe hacer clic en el botón Reiniciar servidor UDF en la cinta después de los cambios de código.

## xlwings PRO

Si bien este libro solo utiliza la versión gratuita y de código abierto de xlwings, también hay un paquete PRO comercial disponible para financiar el mantenimiento y desarrollo continuo del paquete de código abierto. Algunas de las funcionalidades adicionales que ofrece xlwings PRO son:

- El código Python se puede incrustar en Excel, eliminando así los archivos fuente externos.
- El paquete de informes le permite convertir sus libros de trabajo en plantillas con marcadores de posición. Esto les da a los usuarios sin conocimientos técnicos el poder de editar la plantilla sin tener que cambiar el código Python.
- Los instaladores se pueden construir fácilmente para deshacerse de cualquier problema de implementación: los usuarios finales pueden instalar Python incluyendo todas las dependencias con un solo clic, dándoles la sensación de trabajar con libros de Excel normales sin tener que configurar nada manualmente.

Para obtener más detalles sobre xlwings PRO y solicitar una licencia de prueba, consulte la [página de inicio de xlwings](#).

## Conclusión

Este capítulo comenzó mostrándole lo fácil que es ejecutar código Python desde Excel: con Anaconda instalado, solo necesita ejecutar Instalación del complemento xlwings seguido por xlwings inicio rápido myproject, y está listo para hacer clic en el botón Ejecutar principal en el complemento xlwings o usar el RunPython Función VBA. La segunda parte presentó algunas configuraciones que facilitan la implementación de su herramienta xlwings para sus usuarios finales. El hecho de que xlwings venga preinstalado con Anaconda ayuda mucho a reducir las barreras de entrada para nuevos usuarios.

En este capítulo, simplemente usamos el ejemplo de Hello World para aprender cómo funciona todo. El siguiente capítulo toma estos fundamentos para construir Python Package Tracker, una aplicación empresarial completa.

## **El rastreador de paquetes de Python**

En este capítulo, crearemos una aplicación comercial típica que descarga datos de Internet y los almacena en una base de datos antes de visualizarlos en Excel. Esto le ayudará a comprender qué papel juega xlwings en dicha aplicación y le permitirá ver lo fácil que es conectarse a sistemas externos con Python. En un intento por construir un proyecto que se parezca a una aplicación del mundo real pero relativamente simple de seguir, se me ocurrió la *Rastreador de paquetes de Python*, una herramienta de Excel que muestra la cantidad de lanzamientos por año para un paquete de Python determinado. A pesar de ser un estudio de caso, es posible que la herramienta le resulte útil para comprender si un determinado paquete de Python se está desarrollando activamente o no.

Después de familiarizarnos con la aplicación, repasaremos algunos temas que debemos comprender para poder seguir su código: veremos cómo podemos descargar datos de Internet y cómo podemos interactuar con bases de datos antes de aprender sobre el manejo de excepciones en Python, un concepto importante cuando se trata del desarrollo de aplicaciones. Una vez que hayamos terminado con estos preliminares, repasaremos los componentes del Rastreador de paquetes de Python para ver cómo encaja todo. Para concluir este capítulo, veremos cómo funciona la depuración del código xlwings. Al igual que los dos últimos capítulos, este capítulo requiere que tenga una instalación de Microsoft Excel en Windows o macOS.  
¡Comencemos tomando Python Package Tracker para una prueba de manejo!

### **Qué construiremos**

Dirígete al repositorio complementario, donde encontrarás el *rastreador de paquetes* carpeta. Hay un par de archivos en esa carpeta, pero por ahora solo abra el archivo de Excel *package-tracker.xlsmy* diríjase a la hoja de la base de datos: primero necesitamos obtener algunos datos en la base de datos para tener algo con lo que trabajar. Como se muestra en Figura 11-1, escriba un

nombre del paquete, por ejemplo, "xlwings", luego haga clic en Agregar paquete. Puede elegir cualquier nombre de paquete que exista en el [Índice de paquetes de Python](#) (PyPI).



macOS: confirmar el acceso a la carpeta

Cuando agregue el primer paquete en macOS, tendrá que confirmar una ventana emergente para que la aplicación pueda acceder al *rastreador de paquetes* carpeta. Esta es la misma ventana emergente que ya encontramos en [Capítulo 9](#).

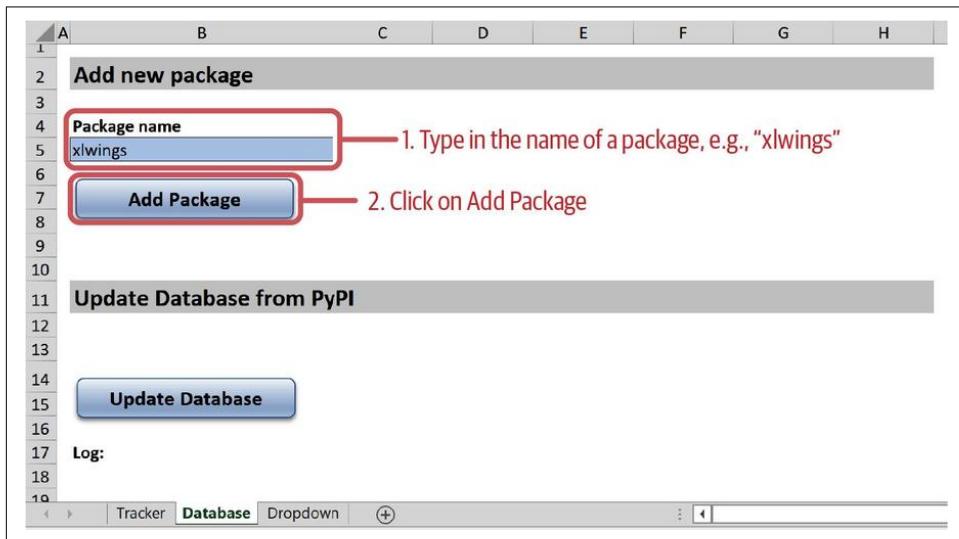


Figura 11-1. El rastreador de paquetes de Python (hoja de base de datos)

Si todo funciona de acuerdo con el plan, verá el mensaje "Añadido xlwings correctamente" a la derecha de donde escribió el nombre del paquete. Además, verá una marca de tiempo de Última actualización en la sección Actualizar base de datos, así como una sección de Registro actualizada donde dice que descargó xlwings correctamente y lo almacenó en la base de datos. Hagamos esto una vez más y agreguemos el paquete pandas para que tengamos más datos con los que jugar. Ahora, cambie a la hoja Rastreador y seleccione xlwings del menú desplegable en la celda B5 antes de hacer clic en Mostrar historial. Su pantalla ahora debería verse similar a [Figura 11-2](#), que muestra la última versión del paquete, así como un cuadro con el número de versiones a lo largo de los años.

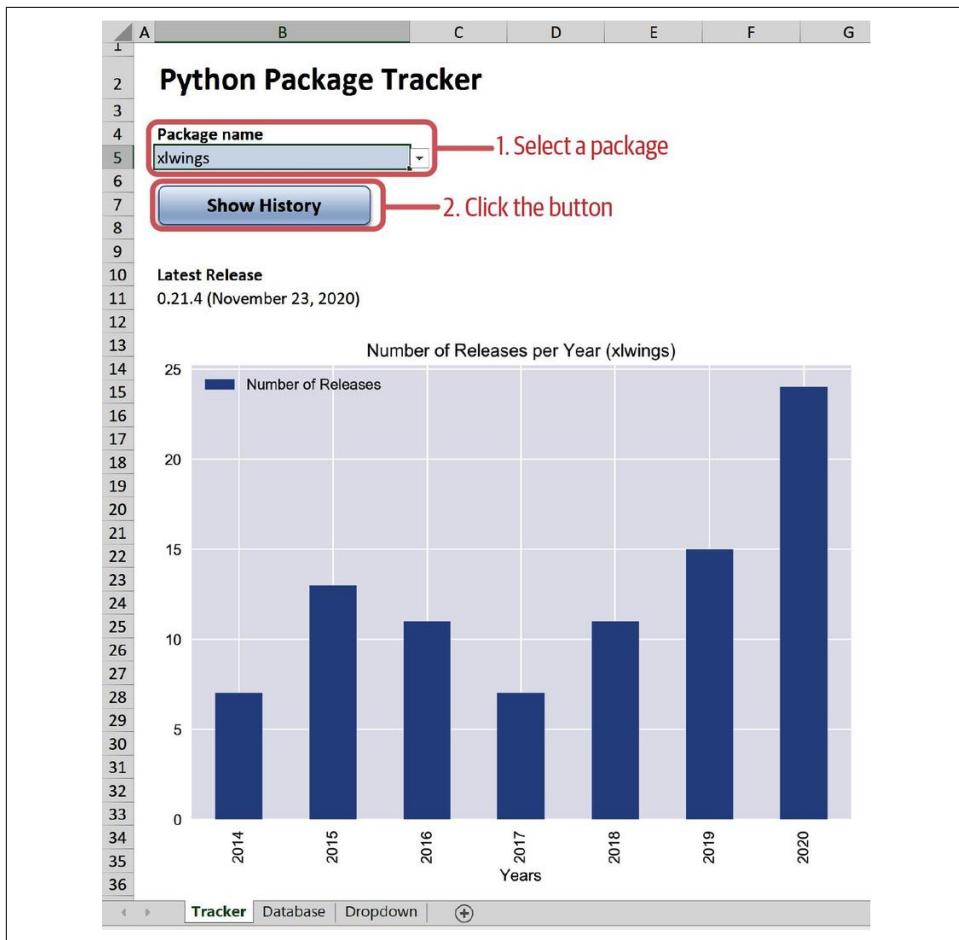


Figura 11-2. El rastreador de paquetes de Python (hoja de seguimiento)

Ahora puede volver a la hoja Base de datos y agregar paquetes adicionales. Siempre que desee actualizar la base de datos con la información más reciente de PyPI, haga clic en el botón Actualizar base de datos: esto sincronizará su base de datos con los datos más recientes de PyPI.

Después de ver cómo funciona Python Package Tracker desde la perspectiva de un usuario, ahora vamos a presentar su funcionalidad principal.

## Funcionalidad central

En esta sección, le presentaré la funcionalidad principal del Rastreador de paquetes de Python: cómo obtener datos a través de API web y cómo consultar bases de datos. También le mostraré cómo manejar las excepciones, un tema que inevitablemente surgirá al escribir código de aplicación. ¡Comencemos con las API web!

## API web

Las API web son una de las formas más populares para que una aplicación obtenga datos de Internet: API significa *Interfaz de programación de aplicaciones* y define cómo interactúa con una aplicación mediante programación. Una API web, por lo tanto, es una API a la que se accede a través de una red, generalmente Internet. Para comprender cómo funcionan las API web, retrocedamos un paso y veamos qué sucede (en términos simplificados) cuando abre una página web en su navegador: después de ingresar una URL en la barra de direcciones, su navegador envía un *OBTENER solicitud* al servidor, solicitando la página web que deseé. Una solicitud GET es un método del protocolo HTTP que utiliza su navegador para comunicarse con el servidor. Cuando el servidor recibe la solicitud, responde devolviendo el documento HTML solicitado, que muestra su navegador: *voilà*, su página web se ha cargado. El protocolo HTTP tiene varios otros métodos; el más común, aparte de la solicitud GET, es el *Solicitud POST*, que se utiliza para enviar datos al servidor (por ejemplo, cuando completa un formulario de contacto en una página web).

Si bien tiene sentido que los servidores envíen una página HTML con un formato agradable para interactuar con humanos, las aplicaciones no se preocupan por el diseño y solo están interesadas en los datos. Por lo tanto, una solicitud GET a una API web funciona como solicitar una página web, pero normalmente recupera los datos en *JSON* en lugar de formato HTML. JSON significa *Notación de objetos de Java- Script* y es una estructura de datos que casi todos los lenguajes de programación entienden, lo que la hace ideal para intercambiar datos entre diferentes sistemas. Aunque la notación utiliza la sintaxis de JavaScript, se parece mucho a cómo se utilizan los diccionarios y las listas (anidados) en Python. Las diferencias son las siguientes:

- JSON solo acepta comillas dobles para cadenas
- Usos de JSON nulo donde Python usa Ninguno
- JSON usa minúsculas cierto y falso mientras están en mayúsculas en Python
- JSON solo acepta cadenas como claves, mientras que los diccionarios de Python aceptan una amplia gama de objetos como claves

los json El módulo de la biblioteca estándar le permite convertir un diccionario de Python en una cadena JSON y viceversa:

En [1]: importar json

En [2]: # Un diccionario de Python ...

```
user_dict = {"nombre": "Fulano de tal",
             "la edad": 23,
             "casado": Falso,
             "niños": Ninguno,
             "aficiones": ["senderismo", "leyendo"]}
```

En [3]: # ...convertido a una cadena JSON

```
# por json.dumps ("cadena de volcado"). El parámetro "sangría" es
# opcional y embellece la impresión.user_json =
json.deshecho(user_dict, sangrar=4)impresión(
user_json)
```

```
{
    "nombre": "Jane Doe",
    "edad": 23,
    "casado": falso,
    "niños": nulo,
    "aficiones": [
        "senderismo",
        "leyendo"
    ]
}
```

En [4]: # Convierta la cadena JSON de nuevo a una estructura de datos nativa de Python

```
json.cargas(user_json)
```

Fuera [4]: {'nombre': 'Jane Doe',
 'edad': 23,
 'casado': Falso,
 'niños': Ninguno,
 'hobbies': ['senderismo', 'lectura']}

## API REST

En lugar de API web, a menudo verá el término *DESCANSAR* o *Sosegado* API. REST significa *Transferencia de estado representacional* y define una API web que se adhiere a ciertas restricciones. En esencia, la idea de REST es que acceda a la información en forma de *recursos apátridas*. Sin estado significa que cada solicitud a una API REST es completamente independiente de cualquier otra solicitud y siempre debe proporcionar el conjunto completo de información que solicita. Tenga en cuenta que el término *API REST* a menudo se usa incorrectamente para referirse a cualquier tipo de API web, incluso si no se adhiere a las restricciones REST.

El consumo de API web suele ser realmente sencillo (veremos cómo funciona esto con Python en un momento), y casi todos los servicios ofrecen uno. Si desea descargar su lista de reproducción favorita de Spotify, debe emitir la siguiente solicitud GET (consulte la [Referencia de la API web de Spotify](#)):

OBTENGA [https://api.spotify.com/v1/playlists/playlist\\_id](https://api.spotify.com/v1/playlists/playlist_id)

O, si desea obtener información sobre sus últimos viajes con Uber, ejecute la siguiente solicitud GET (consulte la [API REST de Uber](#)):

```
OBTENER https://api.uber.com/v1.2/history
```

Sin embargo, para usar estas API, debe estar autenticado, lo que generalmente significa que necesita una cuenta y un token que puede enviar junto con sus solicitudes. Para el Rastreador de paquetes de Python, necesitaremos obtener datos de PyPI para obtener información sobre las versiones de un paquete específico. Afortunadamente, la API web de PyPI no requiere autenticación, por lo que tenemos una cosa menos de la que preocuparnos. Cuando miras el [Documentos de la API JSON de PyPI](#), verás que solo quedan dos *puntos finales*, es decir, fragmentos de URL que se adjuntan a la *URL base*, <https://pypi.org/pypi>:

```
OBTENER /nombre del proyecto/ json OBTENER /  
nombre del proyecto/versión/ json
```

El segundo punto final le proporciona la misma información que el primero, pero solo para una versión específica. Para Python Package Tracker, el primer punto final es todo lo que necesitamos para obtener los detalles sobre las versiones anteriores de un paquete, así que veamos cómo funciona. En Python, una forma sencilla de interactuar con una API web es mediante el paquete Requests que viene preinstalado con Anaconda. Ejecute los siguientes comandos para obtener datos de PyPI sobre pandas:

```
En [5]: importar peticiones
```

```
En [6]: respuesta = peticiones.obtener("https://pypi.org/pypi/pandas/json")  
respuesta.código de estado
```

```
Fuera [6]: 200
```

Cada respuesta viene con un código de estado HTTP: por ejemplo, 200 medio *OK* y 404 medio *Extraviado*. Puede buscar la lista completa de códigos de estado de respuesta HTTP en el [Documentos web de Mozilla](#). Puede que esté familiarizado con el código de estado 404 ya que su navegador a veces lo muestra cuando hace clic en un enlace inactivo o escribe una dirección que no existe. Del mismo modo, también obtendrá un 404 código de estado si ejecuta una solicitud GET con un nombre de paquete que no existe en PyPI. Para ver el contenido de la respuesta, es más fácil llamar al método json del objeto de respuesta, que transformará la cadena JSON de la respuesta en un diccionario de Python:

```
En [7]: respuesta.json()
```

La respuesta es muy larga, por lo que estoy imprimiendo un breve subconjunto aquí para permitirle comprender la estructura:

```
Fuera [7]: {
```

```
    'info': {  
        'bugtrack_url': Ninguno,  
        'licencia': 'BSD',  
        'mantenedor': 'El equipo de desarrollo de PyData',  
        'mantenedor_email': 'pydata@googlegroups.com',  
        'nombre': 'pandas'
```

```

    },
    'lanzamientos': {
        '0.1': [
            {
                'nombre de archivo': 'pandas-0.1.tar.gz',
                'tamaño': 238458,
                'upload_time': '2009-12-25T23: 58: 31'
            },
            {
                'nombre de archivo': 'pandas-0.1.win32-py2.5.exe',
                'tamaño': 313639,
                'upload_time': '2009-12-26T17: 14: 35'
            }
        ]
    }
}

```

Para obtener una lista con todos los lanzamientos y sus fechas, algo que necesitamos para el Rastreador de paquetes de Python, podemos ejecutar el siguiente código para recorrer el lanzamientos diccionario:

```

En [8]: lanzamientos = []
por versión, archivos en respuesta.json() ['lanzamientos'].elementos():
    lanzamientos.adjunтар(F"{{versión}}: {{archivos [0] ['upload_time']}}")
lanzamientos[:3] # mostrar los primeros 3 elementos de la lista

```

```

Fuera [8]: ['0.1: 2009-12-25T23: 58: 31',
            '0.10.0: 2012-12-17T16: 52: 06',
            '0.10.1: 2013-01-22T05: 22: 09']

```

Tenga en cuenta que estamos eligiendo arbitrariamente la marca de tiempo de lanzamiento del paquete que aparece primero en la lista. Una versión específica a menudo tiene varios paquetes para dar cuenta de diferentes versiones de Python y sistemas operativos. Para concluir este tema, es posible que recuerde de [Capítulo 5](#) que los pandas tienen un `read_json` método para devolver un marco de datos directamente desde una cadena JSON. Sin embargo, esto no nos ayudaría aquí, ya que la respuesta de PyPI no está en una estructura que pueda transformarse directamente en un DataFrame.

Esta fue una breve introducción a las API web para comprender su uso en la base de código de Python Package Tracker. ¡Vemos ahora cómo podemos comunicarnos con las bases de datos, el otro sistema externo que utilizamos en nuestra aplicación!

## Bases de datos

Para poder utilizar los datos de PyPI incluso cuando no está conectado a Internet, debe almacenarlos después de la descarga. Si bien puede almacenar sus respuestas JSON como archivos de texto en el disco, una solución mucho más cómoda es usar una base de datos: esto le permite consultar sus datos de una manera fácil. El rastreador de paquetes de Python está usando [SQLite](#), *abase de datos relacional*. Los sistemas de bases de datos relacionales obtienen su nombre de *relación*, que se refiere a la tabla de la base de datos en sí (y no a la relación entre tablas, que es un concepto erróneo común): su objetivo más alto es la integridad de los datos, que logran al

dividir los datos en diferentes tablas (un proceso llamado *normalización*) y aplicando restricciones para evitar datos incoherentes y redundantes. Las bases de datos relacionales utilizan SQL (lenguaje de consulta estructurado) para realizar consultas de bases de datos y se encuentran entre los sistemas de bases de datos relacionales basados en servidor más populares.[servidor SQL](#), [Oráculo](#), [PostgreSQL](#), y [MySQL](#). Como usuario de Excel, es posible que también esté familiarizado con las [acceso Microsoft](#) base de datos.

### Bases de datos NoSQL

En estos días, las bases de datos relacionales tienen una fuerte competencia de *NoSQL* bases de datos que almacenan datos redundantes en un intento de lograr las siguientes ventajas:

#### *Ninguna mesa se une*

Dado que las bases de datos relacionales dividen sus datos en varias tablas, a menudo es necesario combinar la información de dos o más tablas por *unión* ellos, lo que a veces puede ser lento. Esto no es necesario con las bases de datos NoSQL, lo que puede resultar en un mejor rendimiento para ciertos tipos de consultas.

#### *Sin migraciones de bases de datos*

Con los sistemas de bases de datos relacionales, cada vez que realiza un cambio en la estructura de la tabla, por ejemplo, al agregar una nueva columna a una tabla, debe ejecutar una base de datos *migración*. Una migración es un script que trae la base de datos a la nueva estructura deseada. Esto hace que la implementación de nuevas versiones de una aplicación sea más compleja, lo que puede resultar en un tiempo de inactividad, algo que es más fácil de evitar con las bases de datos NoSQL.

#### *Más fácil de escalar*

Las bases de datos NoSQL son más fáciles de distribuir en varios servidores, ya que no hay tablas que dependan entre sí. Esto significa que una aplicación que usa una base de datos NoSQL puede escalar mejor cuando su base de usuarios se dispara.

Las bases de datos NoSQL vienen en muchos sabores: algunas bases de datos son almacenes de valores clave simples, es decir, funcionan de manera similar a un diccionario en Python (por ejemplo, [Redis](#)); otros permiten el almacenamiento de documentos, a menudo en formato JSON (p. ej., [MongoDB](#)). Algunas bases de datos incluso combinan los mundos relacional y NoSQL: PostgreSQL, que resulta ser una de las bases de datos más populares en la comunidad de Python, es tradicionalmente una base de datos relacional, pero también le permite almacenar datos en formato JSON, sin perder la capacidad de consultarlos a través de SQL.

SQLite, la base de datos que vamos a utilizar, es una base de datos basada en archivos como Microsoft Access. Sin embargo, a diferencia de Microsoft Access, que solo funciona en Windows, SQLite funciona en todas las plataformas compatibles con Python. Por otro lado, SQLite no le permite crear una interfaz de usuario como Microsoft Access, pero Excel es útil para esta parte.

Ahora echemos un vistazo a la estructura de la base de datos de Package Tracker antes de descubrir cómo podemos usar Python para conectarnos a bases de datos y realizar consultas SQL. Luego, para concluir esta introducción sobre bases de datos, veremos las inyecciones de SQL, una vulnerabilidad popular de las aplicaciones controladas por bases de datos.

#### **La base de datos de Package Tracker**

La base de datos de Python Package Tracker no podría ser más simple ya que solo tiene dos tablas: la tabla paquetes almacena el nombre del paquete y la tabla package\_versions almacena las cadenas de la versión y la fecha de carga. Las dos mesas se pueden unir en el package\_id: en lugar de almacenar el Nombre del paquete con cada fila en el package\_versions mesa, ha sido *normalizado* en el paquetes mesa. Esto elimina los datos redundantes; los cambios de nombre, por ejemplo, solo deben realizarse en un solo campo en toda la base de datos. Para tener una mejor idea de cómo se ve la base de datos con los paquetes xlwings y pandas cargados, eche un vistazo a Tablas 11-1 y 11-2.

*Tabla 11-1. La tabla de paquetes*

package_id	Nombre del paquete
1	xlwings
2	pandas

*Tabla 11-2. La tabla package\_versions (primeras tres filas de cada package\_id)*

package_id	version_string	uploaded_at
1	0.1.0	2014-03-19 18: 18: 49.000000
1	0.1.1	2014-06-27 16: 26: 36.000000
1	0.2.0	2014-07-29 17: 14: 22.000000...
...	...	
2	0,1	2009-12-25 23: 58: 31.000000
2	0.2beta	2010-05-18 15: 05: 11.000000
2	0.2b1	2010-05-18 15: 09: 05.000000...
...	...	

Figura 11-3 es un diagrama de base de datos que muestra las dos tablas nuevamente de forma esquemática. Puede leer los nombres de las tablas y columnas y obtener información sobre las claves primarias y externas:

#### **Clave primaria**

Las bases de datos relacionales requieren que cada tabla tenga un *Clave primaria*. Una clave primaria es una o más columnas que identifican de forma única una fila (una fila también se llama *registro*). En el caso de la paquetes tabla, la clave principal es package\_id y en el caso

de El package\_versions tabla, la clave primaria es un llamado *clave compuesta*, es decir, una combinación de package\_id y version\_string.

#### Clave externa

La columna package\_id en el package\_versions la mesa es una *clave externa* a la misma columna en el paquetes tabla, simbolizada por la línea que conecta las tablas: una clave externa es una restricción que, en nuestro caso, asegura que cada package\_id en el package\_versions La tabla existe en el paquetes mesa - esta garantiza la integridad de los datos. Las ramas en el extremo derecho de la línea de conexión simbolizan la naturaleza de la relación: un paquete puede tener muchos package\_versions, que se llama un *uno a muchos* relación.

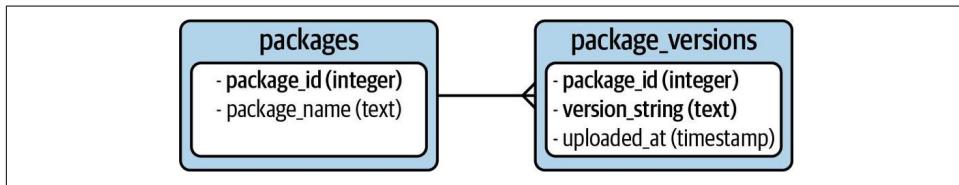


Figura 11-3. Diagrama de la base de datos (las claves principales están en negrita)

Para ver el contenido de las tablas de la base de datos y ejecutar consultas SQL, puede instalar una extensión de VS Code llamada SQLite (consulte la [Documentos de extensión SQLite](#) para obtener más detalles) o utilice un software de administración SQLite dedicado, del cual hay muchos. Sin embargo, usaremos Python para ejecutar consultas SQL. Antes que nada, veamos cómo podemos conectarnos a una base de datos!

#### Conecciones de base de datos

Para conectarse a una base de datos de Python, necesita un *conductor*, es decir, un paquete de Python que sabe cómo comunicarse con la base de datos que está utilizando. Cada base de datos requiere un controlador diferente y cada controlador usa una sintaxis diferente, pero afortunadamente, hay un paquete poderoso llamado [SQLAlchemy](#) que abstrae la mayoría de las diferencias entre las distintas bases de datos y controladores. SQLAlchemy se utiliza principalmente como un *mapeador relacional de objetos* (ORM) que traduce los registros de su base de datos en objetos Python, un concepto con el que muchos desarrolladores, aunque no todos, encuentran más natural trabajar con él. Para simplificar las cosas, ignoramos la funcionalidad ORM y solo usamos SQLAlchemy para facilitar la ejecución de consultas SQL sin procesar. SQLAlchemy también se usa detrás de escena cuando usa pandas para leer y escribir tablas de bases de datos en forma de DataFrames. La ejecución de una consulta de base de datos desde pandas implica tres niveles de paquetes: pandas, SQLAlchemy y el controlador de la base de datos, como se muestra en [Figura 11-4](#). Puede ejecutar consultas a la base de datos desde cada uno de estos tres niveles.

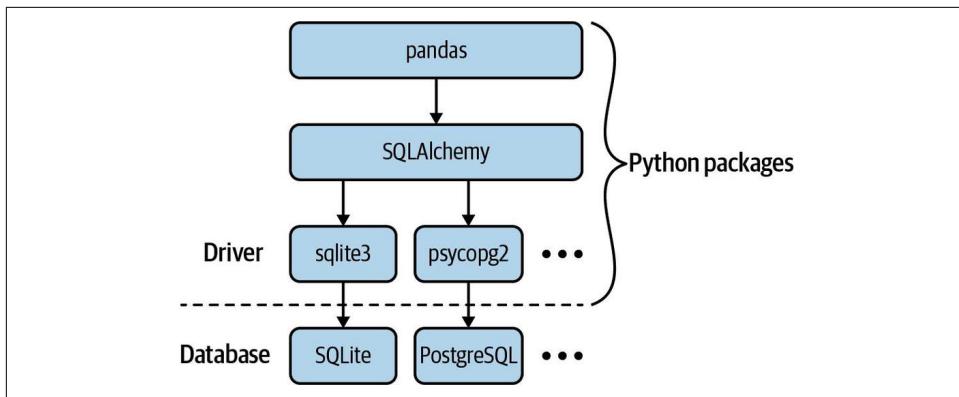


Figura 11-4. Accediendo a bases de datos desde Python

**Tabla 11-3** muestra qué controlador usa SQLAlchemy por defecto (algunas bases de datos se pueden usar con más de un controlador). También le da el formato de la cadena de conexión de la base de datos; usaremos la cadena de conexión en un momento en el que ejecutemos consultas SQL reales.

Tabla 11-3. Cadenas de conexión y controladores predeterminados de SQLAlchemy

Base de datos	Controlador predeterminado	Cadena de conexión
SQLite	sqlite3	sqlite:///ruta de archivo
PostgreSQL	psycopg2	postgresql://nombre de usuario:contraseña@anfitrión:Puerto/base de datos
MySQL	mysql-python	mysql://nombre de usuario:contraseña@anfitrión:Puerto/base de datos
Oráculo	cx_oracle	nombre de usuario:contraseña@anfitrión:Puerto/base de datos
servidor SQL	pyodbc	nombre de usuario:contraseña@anfitrión:Puerto/base de datos

A excepción de SQLite, generalmente necesita una contraseña para conectarse a una base de datos. Y dado que las cadenas de conexión son URL, tendrá que usar la versión codificada de URL de sus contraseñas si tiene caracteres especiales en ellas. Así es como puede imprimir la versión codificada en URL de su contraseña:

En [9]: `importar urllib.parse`

```
En [10]: urllib.analizar gramaticalmente.quote_plus("pa $$
palabra") Fuerá [10]: 'pa% 24% 24word'
```

Habiendo presentado pandas, SQLAlchemy y el controlador de la base de datos como los tres niveles desde los cuales podemos conectarnos a las bases de datos, ¡veamos cómo se comparan en la práctica haciendo algunas consultas SQL!

## Consultas SQL

Incluso si es nuevo en SQL, no debería tener problemas para comprender las pocas consultas SQL que usaré en los siguientes ejemplos y en el Rastreador de paquetes de Python. SQL es un *lenguaje declarativo*, lo que significa que le dices a la base de datos *Lo que quieras en lugar de qué hacer*. Algunas consultas casi se leen como en inglés simple:

**SELECCIONE \* DE paquetes**

Esto le dice a la base de datos que desea *seleccione todas las columnas de la tabla de paquetes*. En el código de producción, no querrá usar el comodín \*, lo que significa *todas las columnas* sin que especifique cada columna explícitamente, ya que esto hace que su consulta sea menos propensa a errores:

**SELECCIONE package\_id, Nombre del paquete DE paquetes**



Consultas de base de datos frente a pandas DataFrames

SQL es un *basado en conjuntos* language, lo que significa que opera en un conjunto de filas en lugar de recorrer filas individuales. Esto es muy similar a cómo trabaja con pandas DataFrames. La consulta SQL:

**SELECCIONE package\_id, Nombre del paquete DE paquetes**

corresponde a la siguiente expresión de pandas (asumiendo que paquetes es un DataFrame):

`paquetes.loc[:, ["package_id", "Nombre del paquete"]]`

Los siguientes ejemplos de código usan el *packagetracker.db* archivo que encontrarás en *el rastreador de paquetes* carpeta del repositorio complementario. Los ejemplos esperan que ya haya agregado xlwings y pandas a la base de datos a través de la interfaz de Excel de Python Package Tracker como hicimos al principio de este capítulo; de lo contrario, solo obtendría resultados vacíos. Sigue la Figura 11-4. De abajo hacia arriba, primero haremos nuestra consulta SQL directamente desde el controlador, luego usaremos SQLAlchemy y finalmente pandas:

En [11]: # Empecemos por las importaciones.

```
importar sqlite3  
de sqlalchemy importar create_engine  
importar pandas como pd
```

En [12]: # Nuestra consulta SQL: "seleccione todas las columnas de la tabla de paquetes"  
`sql = "SELECCIONAR * DE paquetes"`

En [13]: # Opción 1: controlador de base de datos (sqlite3 es parte de la biblioteca estándar)

```
# El uso de la conexión como administrador de contexto confirma automáticamente  
# la transacción o la retrotrae en caso de error.con sqlite3.connect(  
"packagetracker / packagetracker.db") como estafa:  
cursor = estafa.cursor() # Necesitamos un cursor para ejecutar consultas SQL  
resultado = cursor.ejecutar(sql).buscar_todo() # Devolver todos los registros resultado
```

Fuera [13]: [(1, 'xlwings'), (2, 'pandas')]

En [14]: # Opción 2: SQLAlchemy

```
# "create_engine" espera la cadena de conexión de su base de datos.
# Aquí, podemos ejecutar una consulta como método del objeto de
conexión.motor = create_engine("sqlite:///packagetracker/
packagetracker.db")con motor.conectar() como estafa:
    resultado = estafa.ejecutar(sql).buscar_todo()
resultado
```

Fuera [14]: [(1, 'xlwings'), (2, 'pandas')]

En [15]: # Opción 3: pandas

```
# Proporcionar un nombre de tabla a "read_sql" lee la tabla completa.
# Pandas requiere un motor SQLAlchemy que reutilizamos
# el ejemplo anterior.
df = pd.read_sql("paquetes", motor, index_col="package_id")df
```

Fuera [15]:

	Nombre del paquete
package_id	
1	xlwings
2	pandas

En [dieciséis]: # "read\_sql" también acepta una consulta SQL

```
pd.read_sql(sql, motor, index_col="package_id")
```

Fuera [16]:

	Nombre del paquete
package_id	
1	xlwings
2	pandas

En [17]: # El método DataFrame "to\_sql" escribe DataFrames en tablas

```
# "if_exists" tiene que ser "fail", "añadir" o "reemplazar"
# y define qué sucede si la tabla ya existió df.to_sql(
"paquetes2", estafa=motor, if_exists="adjuntar")
```

En [18]: # El comando anterior creó una nueva tabla "paquetes2" y

```
# insertamos los registros del DataFrame df como podemos
# verificar leyéndolo
pd.read_sql("paquetes2", motor, index_col="package_id")
```

Fuera [18]:

	Nombre del paquete
package_id	
1	xlwings
2	pandas

En [19]: # Deshagámonos de la tabla de nuevo ejecutando el

```
# comando "soltar tabla" a través de SQLAlchemy
con motor.conectar() como estafa:
    estafa.ejecutar("DROP TABLE paquetes2")
```

Si debe usar el controlador de la base de datos, SQLAlchemy o pandas para ejecutar sus consultas, depende en gran medida de sus preferencias: personalmente, me gusta el control detallado que obtiene al usar SQLAlchemy y disfruto de poder usar la misma sintaxis con diferentes bases de datos. Por otro lado, los pandasread\_sql Es conveniente obtener el resultado de una consulta en forma de DataFrame.



### Claves externas con SQLite

Sorprendentemente, SQLite no respeta las claves externas de forma predeterminada al ejecutar consultas. Sin embargo, si usa SQLAlchemy, puede aplicar fácilmente claves externas; ver el [Documentos de SQLAlchemy](#). Esto también funcionará si ejecuta las consultas desde pandas. Encontrará el código respectivo en la parte superior de la `database.py` módulo en el *rastreador de paquetes* carpeta del repositorio complementario.

Ahora que sabe cómo ejecutar consultas SQL simples, terminemos esta sección examinando las inyecciones SQL, que pueden representar un riesgo de seguridad para su aplicación.

### inyección SQL

Si no protege sus consultas SQL correctamente, un usuario malintencionado puede ejecutar código SQL arbitrario inyectando declaraciones SQL en los campos de entrada de datos: por ejemplo, en lugar de seleccionar un nombre de paquete como xlwings en el menú desplegable del Rastreador de paquetes de Python, podrían enviar una declaración SQL que cambie su consulta deseada. Esto puede exponer información confidencial o realizar acciones destructivas como eliminar una tabla. ¿Cómo puedes prevenir esto? Primero echemos un vistazo a la siguiente consulta de la base de datos, que ejecuta Package Tracker cuando selecciona xlwings y hace clic en Mostrar historial:<sup>1</sup>

```
SELECCIONE v.uploaded_at, v.version_string  
DE paquetes p  
UNIR INTERNAMENTE package_versions v SOBRE pag.package_id = v.package_id  
DÓNDE pag.package_id = 1
```

Esta consulta une las dos tablas y solo devuelve aquellas filas donde el `package_id` es 1. Para ayudarlo a comprender esta consulta en función de lo que aprendimos en [Capítulo 5](#), si paquetes y `package_versions` eran pandas DataFrames, podría escribir:

```
df = paquetes.unir(package_versions, cómo="interno", sobre="package_id")  
df.loc[df["package_id"] == 1, ["subido_en", "version_string"]]
```

Es obvio que el `package_id` debe ser una variable donde ahora tenemos un código 1 para devolver las filas correctas según el paquete seleccionado. Conociendo las cuerdas f de [Capítulo 3](#), podría tener la tentación de cambiar la última línea de la consulta SQL de esta manera:

```
F"DONDE p.package_id = {package_id}"
```

Si bien esto funcionaría técnicamente, nunca debe hacer esto, ya que abre la puerta a la inyección SQL: por ejemplo, alguien podría enviar '1 O VERDADERO' en lugar de un número entero que representa el `package_id`. La consulta resultante devolvería las filas del conjunto

---

<sup>1</sup> En realidad, la herramienta utiliza Nombre del paquete en lugar de `package_id` para simplificar el código.

mesa en lugar de solo aquellos donde el package\_id es 1. Por lo tanto, utilice siempre la sintaxis que le ofrece SQLAlchemy para los marcadores de posición (comienzan con dos puntos):

En [20]: # Comencemos importando la función de texto de SQLAlchemy  
de **sqlalchemy.sql importar** texto

En [21]: # ":package\_id" es el marcador de posición  
sql = """  
SELECCIONE v.uploaded\_at, v.version\_string  
FROM paquetes p  
INNER JOIN package\_versions v ON p.package\_id = v.package\_id  
DONDE p.package\_id =: package\_id  
ORDEN POR v.uploaded\_at  
""""

En [22]: # Vía SQLAlchemy  
con motor.conectar() como estafa:  
    resultado = estafa.ejecutar(texto(sql), package\_id=1).buscar todo()  
resultado[:3] # Imprime los primeros 3 registros

Fuera [22]: [('2014-03-19 18: 18: 49.000000', '0.1.0'),  
              ('2014-06-27 16: 26: 36.000000', '0.1.1'),  
              ('2014-07-29 17: 14: 22.000000', '0.2.0')]

En [23]: # Vía pandas  
pd.read\_sql(texto(sql), motor, parse\_dates=["subido\_en"],  
              params={"package\_id": 1}, index\_col=[  
              "subido\_en"]).cabeza(3)

Fuera [23]:

	version_string
uploaded_at	
2014-03-19 18:18:49	0.1.0
2014-06-27 16:26:36	0.1.1
2014-07-29 17:14:22	0.2.0

Envolviendo la consulta SQL con SQLAlchemy's texto La función tiene la ventaja de que puede usar la misma sintaxis para marcadores de posición en diferentes bases de datos. De lo contrario, tendría que usar el marcador de posición que usa el controlador de la base de datos:sqlite3 usos ? y psycopg2 usos %s, por ejemplo.

Puede argumentar que la inyección de SQL no es un gran problema cuando sus usuarios tienen acceso directo a Python y podrían ejecutar código arbitrario en la base de datos de todos modos. Pero si toma su prototipo de xlwings y lo transforma en una aplicación web algún día, se convertirá en un gran problema, por lo que es mejor hacerlo correctamente desde el principio.

Además de las API web y las bases de datos, hay otro tema sobre el que hemos saltado hasta ahora que es indispensable para el desarrollo de aplicaciones sólidas: el manejo de excepciones. ¡Vamos a ver cómo funciona!

## Excepciones

Mencioné el manejo de excepciones en [Capítulo 1](#) como un ejemplo de donde VBA con su *Ir a el* mecanismo se ha quedado atrás. En esta sección, le muestro cómo Python usa el *intentar / excepto* mecanismo para manejar errores en sus programas. Siempre que algo esté fuera de su control, los errores pueden ocurrir y sucederán. Por ejemplo, el servidor de correo electrónico puede estar inactivo cuando intenta enviar un correo electrónico, o puede faltar un archivo que su programa espera; en el caso de Python Package Tracker, este podría ser el archivo de la base de datos. Tratar con la entrada del usuario es otra área en la que debe prepararse para las entradas que no tienen sentido. Practiquemos un poco, si se llama a la siguiente función con un cero, obtendrás un ZeroDivisionError:

```
En [24]: def print_reciprocal(número):
    resultado = 1 / número
    impresión(F"El recíproco es: {resultado}")
```

```
En [25]: print_reciprocal(0) # Esto generará un error
```

```
-----
ZeroDivisionError Traceback (última llamada más reciente) <ipython-
input-25-095f19ebb9e9> en <module>
----> 1 print_reciprocal (0) # Esto generará un error
```

```
<ipython-input-24-88fd8a4711> en print_reciprocal (número)
  1 def print_reciprocal (número):
----> 2      resultado = 1 / número
      3      print (f "El recíproco es: {resultado}")
```

ZeroDivisionError: división por cero

Para permitir que su programa reaccione con gracia a tales errores, use las declaraciones try / except (esto es el equivalente al ejemplo de VBA en [Capítulo 1](#)):

```
En [26]: def print_reciprocal(número):
    tratar:
        resultado = 1 / número
    excepto Excepción como mi:
        # "as e" hace que el objeto Exception esté disponible como variable "e"
        # "repr" significa "representación imprimible" de un objeto
        # y te devuelve una cadena con el mensaje de error
        impresión(F"Hubo un error: {repr (e)}")resultado = "N / A"

    demás:
        impresión("¡No hubo ningún error!")
    finalmente:
        impresión(F"El recíproco es: {resultado}")
```

Siempre que ocurra un error en el tratar bloque, la ejecución del código pasa al excepto bloque donde puede manejar el error: esto le permite dar al usuario retroalimentación útil o escribir el error en un archivo de registro. Los demás La cláusula solo se ejecuta si no se produce ningún error durante la tratar bloque y el finalmente El bloque se ejecuta siempre, ya sea que

se generó un error. A menudo, te saldrás con la tuya solotrar y excepto bloques. Veamos la salida de la función dadas diferentes entradas:

En [27]: `print_reciprocal(10)`

¡No hubo ningún error!  
El recíproco es: 0.1

En [28]: `print_reciprocal("a")`

Hubo un error: TypeError ("tipos de operandos no admitidos para /: 'int' y 'str'"")

El recíproco es: N / A

En [29]: `print_reciprocal(0)`

Hubo un error: ZeroDivisionError ('división por cero') El recíproco  
es: N / A

La forma en que he usado la declaración `except` significa que cualquier excepción que ocurra en el tratar bloque hará que la ejecución del código continúe en el excepto cuadra. Normalmente, eso no es lo que quieres. Desea verificar un error lo más específico posible y manejar solo aquellos que espera. De lo contrario, su programa puede fallar por algo completamente inesperado, lo que dificulta la depuración. Para solucionar esto, reescriba la función de la siguiente manera, verificando explícitamente los dos errores que esperamos (estoy dejando de lado el demás y finalmente declaraciones):

En [30]: `def print_reciprocal(número):`

```
    tratar:  
        resultado = 1 / número  
        impresión(F"El recíproco es: {resultado}")  
    excepto (Error de tecleado, ZeroDivisionError):  
        impresión("Escriba cualquier número excepto 0.")
```

Ejecutemos el código de nuevo:

En [31]: `print_reciprocal("a")` Escriba

cualquier número excepto 0.

Si desea manejar un error de manera diferente según la excepción, hágalo por separado:

En [32]: `def print_reciprocal(número):`

```
    tratar:  
        resultado = 1 / número  
        impresión(F"El recíproco es: {resultado}")  
    excepto Error de tecleado:  
        impresión("Por favor ingrese un número").  
    excepto ZeroDivisionError:  
        impresión("El recíproco de 0 no está definido".)
```

En [33]: `print_reciprocal("a")` Por

favor ingrese un número.

En [34]: `print_reciprocal(0)` El recíproco de 0 no está definido.

Ahora que conoce el manejo de errores, las API web y las bases de datos, está listo para pasar a la siguiente sección, donde veremos cada componente del Rastreador de paquetes de Python.

## Estructura de la aplicación

En esta sección, veremos detrás de escena del Rastreador de paquetes de Python para comprender cómo funciona todo. Primero, recorreremos el frontend de la aplicación, es decir, el archivo de Excel, antes de mirar su backend, es decir, el código Python. Para concluir esta sección, veremos cómo funciona la depuración de un proyecto xlwings, una habilidad útil con proyectos del tamaño y complejidad del Package Tracker.

En el *rastreador de paquetes* directorio en el repositorio complementario, encontrará cuatro archivos. ¿Recuerdas cuando hablé de *separación de intereses* en [Capítulo 1](#)? Ahora podemos asignar estos archivos a las diferentes capas como se muestra en [Cuadro 11-4](#):

*Cuadro 11-4. Separación de intereses*

Capa	Expediente	Descripción
Capa de presentación	<code>packagetracker.xlsx</code>	Esta es la interfaz y, como tal, el único archivo con el que interactúa el usuario final.
Capa empresarial	<code>packagetracker.py</code>	Este módulo maneja la descarga de datos a través de la API web y procesa los números con pandas.
Capa de datos	<code>database.py</code>	Este módulo maneja todas las consultas de la base de datos.
Base de datos	<code>packagetracker.db</code>	Este es un archivo de base de datos SQLite.

En este contexto, vale la pena mencionar que la capa de presentación, es decir, el archivo de Excel, no contiene una fórmula de celda única, lo que hace que la herramienta sea mucho más fácil de auditar y controlar.

### Controlador de vista de modelo (MVC)

La separación de preocupaciones tiene muchas caras y el desglose como se muestra en [Cuadro 11-4](#) es solo una posibilidad. Otro patrón de diseño popular con el que puede encontrarse con relativa rapidez se llama *controlador de vista de modelo* (MVC). En el mundo MVC, el núcleo de la aplicación es el *modelo* donde se manejan todos los datos y, por lo general, la mayor parte de la lógica empresarial. Mientras que la *vista* corresponde a la capa de presentación, el *controlador* es solo una capa delgada que se encuentra entre el modelo y la vista para asegurarse de que siempre estén sincronizados. Para simplificar las cosas, no estoy usando el patrón MVC en este libro.

Ahora que sabe de qué es responsable cada archivo, ¡sigamos adelante y observemos más de cerca cómo se ha configurado la interfaz de Excel!

## Interfaz

Cuando construye una aplicación web, diferencia entre los *Interfaz*, que es la parte de la aplicación que se ejecuta en su navegador, y el *backend*, que es el código que se ejecuta en el servidor. Podemos aplicar la misma terminología con las herramientas xlwings: el frontend es el archivo de Excel y el backend es el código Python al que llamas a través de RunPython. Si desea construir la interfaz desde cero, comience ejecutando el siguiente comando en un indicador de Anaconda (asegúrese de CD primero en el directorio de su elección):

(base)> rastreador de paquetes de inicio rápido de xlwings

Navega al *rastreador de paquetes* directorio y abrir *packagetracker.xlsm* en Excel. Comience agregando las tres pestañas, Tracker, Database y Dropdown, como se muestra en Figura 11-5.

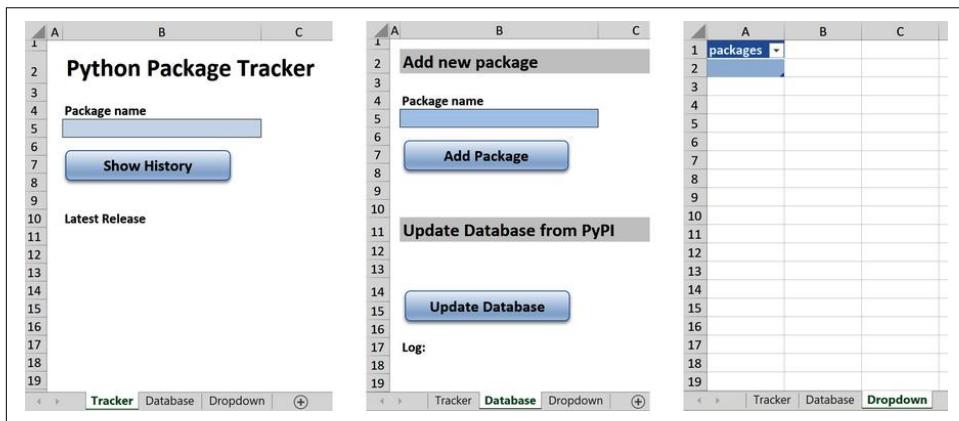


Figura 11-5. Construyendo la interfaz de usuario

Si bien debería poder hacerse cargo del texto y el formato de Figura 11-5, Necesito darte algunos detalles más sobre las cosas que no son visibles:

### Botones

Para que la herramienta se parezca un poco menos a Windows 3.1, no utilicé los botones de macro estándar que usamos en el capítulo anterior. En su lugar, fui a Insertar> Formas e inserté un Rectángulo redondeado. Si desea utilizar el botón estándar, también está bien, pero en este punto, no asigne una macro todavía.

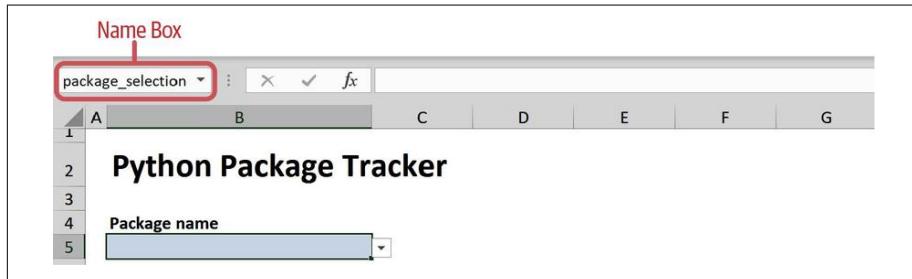
#### Rangos con nombre

Para que la herramienta sea un poco más fácil de mantener, usaremos rangos con nombre en lugar de direcciones de celda en el código Python. Por lo tanto, agregue los rangos con nombre como se muestra en [Tabla 11-5.](#)

*Tabla 11-5. Rangos con nombre*

Hoja	Celda	Nombre
Rastreador	B5	package_selection
Rastreador	B11	último lanzamiento
Base de datos	B5	nuevo paquete
Base de datos	B13	updated_at
Base de datos	B18	Iniciar sesión

Una forma de agregar rangos con nombre es seleccionar la celda, luego escribir el nombre en el Cuadro de nombre y finalmente confirmar presionando Enter, como en [Figura 11-6..](#)



*Figura 11-6. El cuadro de nombre*

#### Mesas

En la hoja desplegable, después de escribir "paquetes" en la celda A1, seleccione A1, luego vaya a Insertar> Tabla y asegúrese de activar la casilla de verificación junto a "Mi tabla tiene encabezados". Para finalizar, con la tabla seleccionada, vaya a la pestaña de la cinta Diseño de tabla (Windows) o Tabla (macOS) y cambie el nombre de la tabla deTabla 1 para drop down\_content, como se muestra en [Figura 11-7..](#)

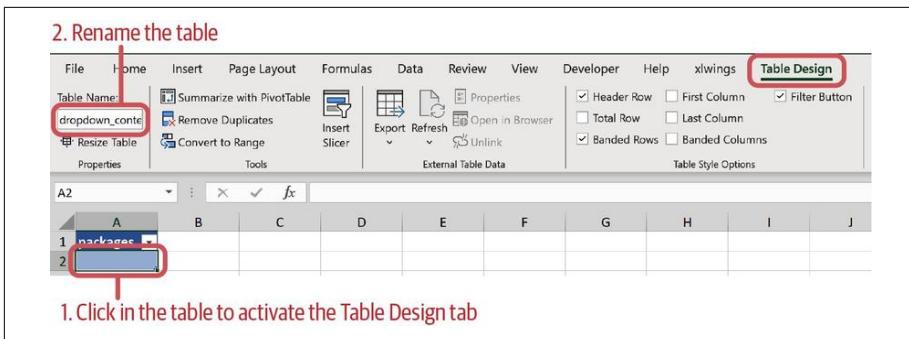


Figura 11-7. Cambiar el nombre de una tabla de Excel

### Validación de datos

Usamos la validación de datos para proporcionar el menú desplegable en la celda B5 en la hoja de seguimiento. Para agregarlo, seleccione la celda B5, luego vaya a Datos > Validación de datos y en Permitir, seleccione Lista. En fuente, establezca la siguiente fórmula:

= INDIRECTO ("dropdown\_content [paquetes]")

Luego, confirme con OK. Esto es solo una referencia al cuerpo de la tabla, pero como Excel no acepta una referencia de tabla directamente, tenemos que envolverla en un INDIRECTO fórmula, que resuelve la tabla en su dirección. Aún así, al usar una tabla, redimensionará correctamente el rango que se muestra en el menú desplegable cuando agreguemos más paquetes.

### Formato condicional

Cuando agrega un paquete, puede haber algunos errores que nos gustaría mostrarle al usuario: el campo podría estar vacío, el paquete puede que ya exista en la base de datos o puede faltar en PyPI. Para mostrar el error en rojo y otros mensajes en negro, usaremos un truco simple basado en el formato condicional: queremos una fuente roja siempre que el mensaje contenga la palabra "error". En la hoja Base de datos, seleccione la celda C5, que es donde escribiremos el mensaje. Luego vaya a Inicio > Formato condicional > Resaltar reglas de celdas > Texto que contiene. Ingrese el valor

**error** y seleccione Texto rojo en el menú desplegable como se muestra en Figura 11-8., luego haga clic en Aceptar. Aplique el mismo formato condicional a la celda C5 en la hoja de seguimiento.

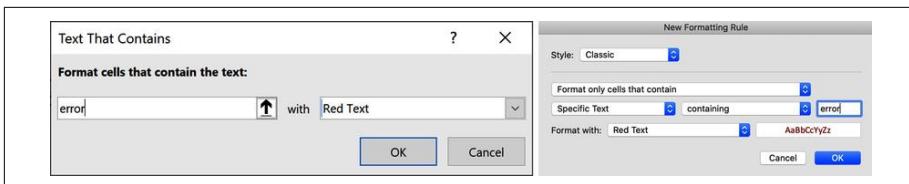


Figura 11-8. Formateo condicional en Windows (izquierda) y macOS (derecha)

En las hojas de Seguimiento y Base de datos, las líneas de cuadrícula se han ocultado al desmarcar la casilla de verificación Ver en Diseño de página> Líneas de cuadrícula.

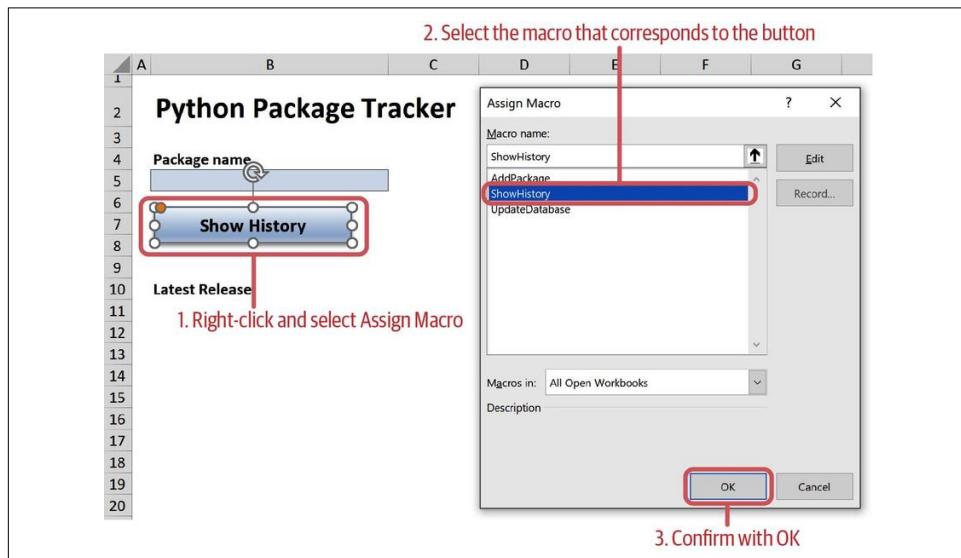
En este punto, la interfaz de usuario está completa y debería verse como [Figura 11-5](#). Ahora necesitamos agregar el RunPython llamadas en el editor de VBA y conectarlas con los botones. Haga clic en Alt + F11 (Windows) u Opción-F11 (macOS) para abrir el editor de VBA, luego, debajo del VBAProject de *packagetracker.xlsm*, haga doble clic en Module1 en el lado izquierdo debajo de Módulos para abrirlo. Eliminar el existente SampleCall código y reemplácelo con las siguientes macros:

```
Sub AddPackage()
    RunPython "importar packagetracker; packagetracker.add_package ()"End Sub

Sub ShowHistory()
    RunPython "importar packagetracker; packagetracker.show_history ()"
End Sub

Sub Actualizar base de datos()
    RunPython "importar packagetracker; packagetracker.update_database ()"End Sub
```

A continuación, haga clic con el botón derecho en cada botón, seleccione Asignar macro y seleccione la macro que corresponda al botón. [Figura 11-9](#). muestra el botón Mostrar historial, pero funciona igual para los botones Agregar paquete y Actualizar base de datos.



*Figura 11-9. Asignar la macro ShowHistory al botón Mostrar historial*

La interfaz ya está lista y podemos continuar con la interfaz de Python.

## Backend

El código de los dos archivos Python *packagetracker.py* y *database.py* es demasiado largo para mostrarse aquí, por lo que deberá abrirlos desde el repositorio complementario en VS Code. Sin embargo, me referiré a un par de fragmentos de código en esta sección para explicar algunos conceptos clave. Veamos qué sucede cuando hace clic en el botón Agregar paquete en la hoja de la base de datos. El botón tiene asignada la siguiente macro VBA:

```
Sub AddPackage()
    RunPython "importar packagetracker; packagetracker.add_package ()"End Sub
```

Como ves, el RunPython la función llama al *add\_package* Función de Python en el rastreador de paquetes módulo como se muestra en [Ejemplo 11-1](#).



Sin código de producción

La aplicación se mantiene lo más simple posible para que sea más fácil de seguir; no comprueba todos los posibles errores. En un entorno de producción, querría hacerlo más robusto: por ejemplo, mostraría un error fácil de usar si no puede encontrar el archivo de base de datos.

*Ejemplo 11-1. Los add\_package función en packagetracker.py (sin comentarios)*

```
def add_package():
    db_sheet = xw.Libro.llamador().hojas["Base de datos"]
    Nombre del paquete =
    db_sheet["nuevo paquete"].valor
    feedback_cell = db_sheet["nuevo paquete"].compensar(column_offset=1)

    feedback_cell.contenidos claros()

    si no Nombre del paquete:
        feedback_cell.valor = "Error: ¡proporcione un nombre!"      ①
        regreso
    si peticiones.obtener(F"{BASE_URL} / {package_name} / json",
        se acabó el tiempo=6).código de estado != 200:            ②
        feedback_cell.valor = "Error: ¡Paquete no encontrado!"
        regreso

    error = base de datos.store_package(Nombre del
        paquete)db_sheet["nuevo paquete"].contenidos claros()      ③

    si error:
        feedback_cell.valor = F"Error: {error}"demás
        :
        feedback_cell.valor = F"Se agregó {package_name} correctamente."
```

actualizar base de datos) ④  
refresh\_dropdown() ⑤

- ① El "error" en el mensaje de retroalimentación activará la fuente roja en Excel a través del formato condicional.
- ② De forma predeterminada, Requests está esperando eternamente una respuesta que podría hacer que la aplicación se "cuelgue" en los casos en que PyPI tenga un problema y esté respondiendo lentamente. Es por eso que para el código de producción, siempre debe incluir un código explícito.~~se acabó el tiempo~~ parámetro.
- ③ Los store\_package devuelve la función Ninguno si la operación fue exitosa y una cadena con el mensaje de error en caso contrario.
- ④ Para simplificar las cosas, se actualiza toda la base de datos. En un entorno de producción, solo agregaría los registros del nuevo paquete.
- ⑤ Esto actualizará la tabla en la hoja desplegable con el contenido de la paquetes mesa. Junto con la validación de datos que hemos configurado en Excel, esto asegura que todos los paquetes aparezcan en el menú desplegable de la hoja Tracker. Debería darles a los usuarios una forma de llamar a esta función directamente si permite que la base de datos se llene desde fuera de su archivo de Excel. Este es el caso tan pronto como tenga varios usuarios usando la misma base de datos desde diferentes archivos de Excel.

Debería poder seguir las otras funciones en el *packagetracker.py* archivo con la ayuda de los comentarios en el código. Dirijamos ahora nuestra atención a la *database.py* expediente. Las primeras líneas se muestran en [Ejemplo 11-2](#).

*Ejemplo 11-2. database.py (extracto con las importaciones relevantes)*

```
de pathlib importar Sendero

importar sqlalchemy
importar pandas como pd

...

# Queremos que el archivo de la base de datos se ubique junto a este archivo.
# Aquí, estamos convirtiendo el camino en un camino absoluto.
this_dir = Sendero(__expediente__).resolver().padre ①_path =
this_dir / "packagetracker.db"

# Motor de base de datos
motor = sqlalchemy.create_engine(F"sqlite:/// {db_path}")
```

- ① Si necesita un repaso de lo que hace esta línea, eche un vistazo al comienzo de [Capítulo 7](#), donde lo explico en el código del informe de ventas.

Si bien este fragmento se ocupa de armar la ruta del archivo de la base de datos, también le muestra cómo evitar un error común cuando trabaja con cualquier tipo de archivo, ya sea una imagen, un archivo CSV o, como en este caso, un archivo de base de datos. Cuando crea una secuencia de comandos rápida de Python, puede usar una ruta relativa como lo he hecho en la mayoría de las muestras de cuadernos de Jupyter:

```
motor = sqlalchemy.create_engine("sqlite:///packagetracker.db")
```

Esto funciona siempre que su archivo esté en su directorio de trabajo. Sin embargo, cuando ejecuta este código desde Excel a través de RunPython, el directorio de trabajo puede ser diferente, lo que hará que Python busque el archivo en la carpeta incorrecta; obtendrá un Archivo no encontrado error. Puede resolver este problema proporcionando una ruta absoluta o creando una ruta de la forma en que lo hacemos en [Ejemplo 11-2](#). Esto asegura que Python esté buscando el archivo en el mismo directorio que el archivo fuente, incluso si ejecuta el código desde Excel a través de RunPython.

Si desea crear Python Package Tracker desde cero, deberá crear la base de datos manualmente: ejecute el *database.py* archivo como un script, por ejemplo, haciendo clic en el botón Ejecutar archivo en VS Code. Esto creará el archivo de base de datos *packagetracker.db* con las dos mesas. El código que crea la base de datos se encuentra en la parte inferior de *database.py*.

```
si __nombre__ == "__principal__":
    create_db()
```

Mientras que la última línea llama al *create\_db* función, el significado de la anterior si La declaración se explica en el siguiente consejo.



```
if __name__ == "__main__"
```

Verás esto si declaración en la parte inferior de muchos archivos de Python. Se asegura de que este código solo se ejecute cuando ejecuta el archivo *como un guión*, por ejemplo, desde un indicador de Anaconda ejecutando `python database.py` o haciendo clic en el botón Ejecutar archivo en VS Code. Sin embargo, no se activará cuando ejecute el archivo *importándolo como módulo*, es decir, haciendo importar base de datos en su código. La razón de esto es que Python asigna el nombre `__principal__` al archivo si lo ejecuta directamente como script, mientras que será llamado por su nombre de módulo (base de datos) cuando lo ejecutas a través del importar declaración. Dado que Python rastrea el nombre del archivo en una variable llamada `__name__`, los si declaración evaluará a Cierto solo cuando lo ejecuta como script; no se activará cuando lo importe desde el *packagetracker.py* expediente.

El resto de base de datos El módulo ejecuta sentencias SQL tanto a través de SQLAlchemy como de pandas to\_sql y read\_sql métodos para que tenga una idea de ambos enfoques.

## Pasar a PostgreSQL

Si desea reemplazar SQLite con PostgreSQL, una base de datos basada en servidor, solo hay algunas cosas que necesita cambiar. Primero que nada, necesitas correr para instalar psycopg2 (o pip instalar psycopg2-binary si no está utilizando la distribución Anaconda) para instalar el controlador PostgreSQL. Entonces, en `endatabase.py`, cambie la cadena de conexión en el `create_engine` función a la versión de PostgreSQL como se muestra en [Tabla 11-3](#). Finalmente, para crear las tablas, necesitaría cambiar el `ENTERO` tipo de datos de `packages.package_id` a la notación específica de PostgreSQL de DE SERIE. La creación de una clave primaria de incremento automático es un ejemplo de dónde difieren los dialectos SQL.

Cuando crea herramientas de la complejidad del Rastreador de paquetes de Python, probablemente se encuentre con algunos problemas en el camino: por ejemplo, es posible que haya cambiado el nombre de un rango con nombre en Excel y haya olvidado ajustar el código de Python en consecuencia. ¡Este es un buen momento para ver cómo funciona la depuración!

## Depuración

Para depurar fácilmente sus scripts xlwings, ejecute sus funciones directamente desde VS Code, en lugar de ejecutarlas haciendo clic en un botón en Excel. Las siguientes líneas en la parte inferior de `lapackagetracker.py` archivo le ayudará a depurar el `add_package` función (este es el mismo código que también encontrará en la parte inferior de una Inicio rápido proyecto):

```
si __nombre__ == "__principal__": ❶
    xw.Libro("packagetracker.xlsm").set_mock_caller() ❷
    add_package()
```

- ❶ Acabamos de ver cómo funciona esta declaración if cuando estábamos mirando el *base de datos.py* código; ver el consejo anterior.
- ❷ Como este código solo se ejecuta cuando ejecuta el archivo directamente desde Python como un script, el `set_mock_caller ()` El comando solo está destinado a fines de depuración: cuando ejecuta el archivo en VS Code o desde un indicador de Anaconda, establece el `xw.Book.caller ()` para `xw.Book ("packagetracker.xlsm")`. El único propósito de hacer esto es poder ejecutar su script desde ambos lados, Python y Excel, sin tener que cambiar el objeto del libro dentro del `add_package` función de ida y vuelta entre `xw.Book ("packagetracker.xlsm")` (cuando lo llamas desde VS Code) y `xw.Book.caller ()` (cuando lo llama desde Excel).

Abierto *packagetracker.py* en VS Code y establezca un punto de interrupción en cualquier línea dentro del *add\_package* función haciendo clic a la izquierda de los números de línea. Luego presione F5 y seleccione "Archivo Python" en el cuadro de diálogo para iniciar el depurador y hacer que su código se detenga en el punto de interrupción. Asegúrese de presionar F5 en lugar de usar el botón Ejecutar archivo, ya que el botón Ejecutar archivo ignora los puntos de interrupción.



### Depurando con VS Code y Anaconda

En Windows, cuando ejecuta el depurador de VS Code por primera vez con código que usa pandas, es posible que reciba un error: "Se ha producido una excepción: ImportError, no se pueden importar las dependencias requeridas: numpy". Esto sucede porque el depurador está en funcionamiento antes de que el entorno Conda se haya activado correctamente. Como solución alternativa, detenga el depurador haciendo clic en el ícono de detención y presione F5 nuevamente; funcionará la segunda vez.

Si no está familiarizado con cómo funciona el depurador en VS Code, eche un vistazo a [el apéndice B](#) donde explico todas las funciones y botones relevantes. También retomaremos el tema en la sección respectiva del próximo capítulo. Si desea depurar una función diferente, detenga la sesión de depuración actual, luego ajuste el nombre de la función en la parte inferior de su archivo. Por ejemplo, para depurar el *show\_history* función, cambie la última línea en *packagetracker.py* de la siguiente manera antes de presionar F5 nuevamente:

```
si __nombre__ == "__principal__":
    xw.Libro("packagetracker.xlsm").set_mock_caller()
    show_history()
```

En Windows, también puede activar la casilla de verificación Mostrar consola en el complemento xlwings, que mostrará un símbolo del sistema mientras el RunPython la llamada se está ejecutando.<sup>2</sup> Esto le permite imprimir información adicional para ayudarlo a depurar el problema. Por ejemplo, puede imprimir el valor de una variable para inspeccionarla en el símbolo del sistema. Sin embargo, una vez que se haya ejecutado el código, el símbolo del sistema se cerrará. Si necesita mantenerlo abierto un poco más, hay un truco fácil: agregue *aporte()* como la última línea de su función. Esto hace que Python espere la entrada del usuario en lugar de cerrar el símbolo del sistema de inmediato. Cuando haya terminado de inspeccionar la salida, presione Enter en el símbolo del sistema para cerrarla, solo asegúrese de eliminar la *aporte()* línea de nuevo antes de desmarcar la opción Mostrar consola!

---

<sup>2</sup> En el momento de escribir este artículo, esta opción aún no está disponible en macOS.

## Conclusión

Este capítulo le mostró que es posible crear aplicaciones razonablemente complejas con un mínimo de esfuerzo. Poder aprovechar los poderosos paquetes de Python como Requests o SQLAlchemy hace toda la diferencia para mí cuando comparo esto con VBA, donde hablar con sistemas externos es mucho más difícil. Si tiene casos de uso similares, le recomiendo encarecidamente que mire más de cerca tanto las solicitudes como la alquimia de SQL; ser capaz de tratar de manera eficiente las fuentes de datos externas le permitirá hacer que copiar / pegar sea cosa del pasado.

En lugar de hacer clic en los botones, algunos usuarios prefieren crear sus herramientas de Excel utilizando fórmulas de celda. El siguiente capítulo le muestra cómo xlwings le permite escribir funciones definidas por el usuario en Python, lo que le permite reutilizar la mayoría de los conceptos de xlwings que hemos aprendido hasta ahora.

## Funciones definidas por el usuario (UDF)

Los tres capítulos anteriores le mostraron cómo automatizar Excel con una secuencia de comandos de Python y cómo ejecutar dicha secuencia de comandos desde Excel con solo hacer clic en un botón. Este capítulo presenta las funciones definidas por el usuario (UDF) como otra opción para llamar al código Python desde Excel con xlwings. Las UDF son funciones de Python que usa en celdas de Excel de la misma manera que usa funciones integradas comoSUMA o PROMEDIO. Como en el capítulo anterior, comenzaremos con el Inicio rápido comando que nos permite probar una primera UDF en poco tiempo. Luego pasamos a un estudio de caso sobre la obtención y procesamiento de datos de Google Trends como una excusa para trabajar con UDF más complejas: aprenderemos cómo trabajar con pandas DataFrames y gráficos, así como cómo depurar UDF. Para concluir este capítulo, profundizaremos en algunos temas avanzados con un enfoque en el desempeño.

Desafortunadamente, xlwings no admite UDF en macOS, lo que hace que este capítulo sea el único que requiere que ejecute las muestras en Windows.<sup>1</sup>



### Una nota para los usuarios de macOS y Linux

Incluso si no está en Windows, es posible que desee echar un vistazo al estudio de caso de Tendencias de Google, ya que podría adaptarlo fácilmente para trabajar con un RunPython llamar a macOS. También puede producir un informe utilizando una de las bibliotecas de escritor deCapítulo 8, que incluso funciona en Linux.

---

<sup>1</sup> La implementación de Windows utiliza un servidor COM (he introducido la tecnología COM brevemente en Capítulo 9).

Dado que COM no existe en macOS, las UDF tendrían que volver a implementarse desde cero, lo cual es mucho trabajo y simplemente aún no se ha hecho.

# Introducción a las UDF

Esta sección comienza con los requisitos previos para escribir UDF antes de que podamos usar la Inicio rápido comando para ejecutar nuestra primera UDF. Para seguir los ejemplos de este capítulo, necesitará instalar el complemento xlwings y tener habilitada la opción de Excel "Confiar en el acceso al modo de objeto del proyecto VBA":

#### Complemento

Supongo que tiene instalado el complemento xlwings como se explica en [Capítulo 10](#). Sin embargo, este no es un requisito estricto: si bien facilita el desarrollo, especialmente para hacer clic en el botón Importar funciones, no es necesario para la implementación y se puede reemplazar configurando el libro de trabajo en el modo independiente; para obtener más detalles, consulte [Capítulo 10](#).

#### Confiar en el acceso al modelo de objetos del proyecto de VBA

Para poder escribir sus primeras UDF, deberá cambiar una configuración en Excel: vaya a Archivo> Opciones> Centro de confianza> Configuración del centro de confianza> Configuración de macros y active la casilla de verificación para "Confiar en el acceso al modelo de objetos del proyecto VBA". como en [Figura 12-1](#). Esto permite que xlwings inserte automáticamente un módulo VBA en su libro de trabajo cuando hace clic en el botón Importar funciones en el complemento, como veremos en breve. Dado que solo confía en esta configuración durante el proceso de importación, debe considerarla como una configuración de desarrollador por la que los usuarios finales no deben preocuparse.

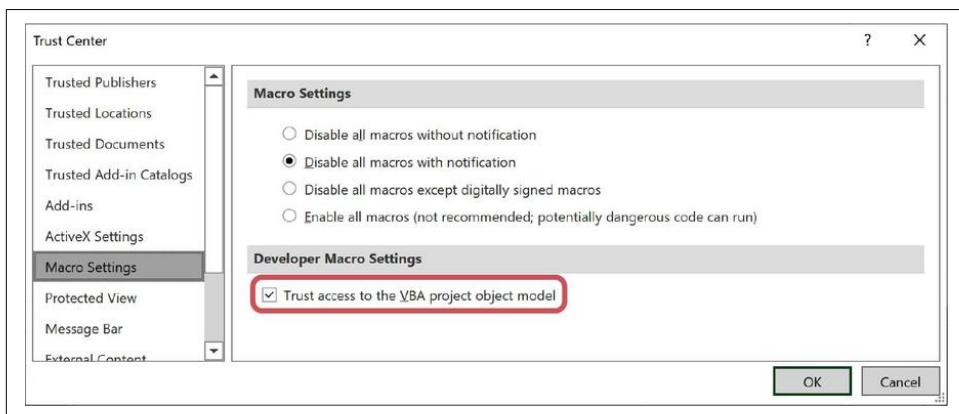


Figura 12-1. Confiar en el acceso al modelo de objetos del proyecto de VBA

Con estos dos requisitos previos en su lugar, ¡está listo para ejecutar su primera UDF!

## Inicio rápido de UDF

Como de costumbre, la forma más fácil de despegar es utilizar el Inicio rápido mando. Antes de ejecutar lo siguiente en un indicador de Anaconda, asegúrese de cambiar al

directorio de su elección a través del CD mando. Por ejemplo, si está en su directorio de inicio y desea cambiar al escritorio, ejecute cd de escritorio primero:

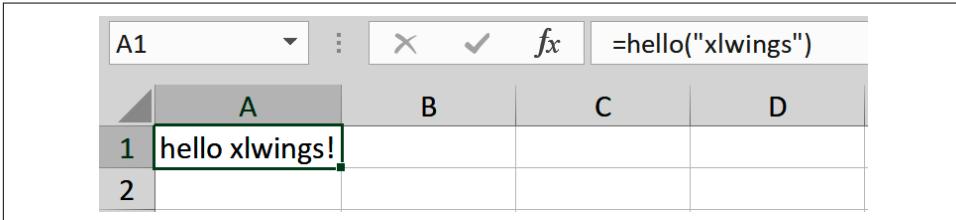
(base)> **Inicio rápido de xlwings first\_udf**

Navega al *first\_udf* carpeta en el Explorador de archivos y abra *first\_udf.xlsxm* en Excel y *first\_udf.py* en VS Code. Luego, en el complemento de cinta de xlwings, haga clic en el botón Importar funciones. De forma predeterminada, esta es una acción silenciosa, es decir, solo verá algo en caso de error. Sin embargo, si activa la casilla de verificación Mostrar consola en el complemento de Excel y vuelve a hacer clic en el botón Importar funciones, se abre un símbolo del sistema e imprime lo siguiente:

```
servidor xlwings en ejecución [...]
Funciones importadas de los siguientes módulos: first_udf
```

Sin embargo, la primera línea imprime algunos detalles más que podemos ignorar; lo importante es que una vez que se imprime esta línea, Python está listo y funcionando. La segunda línea confirma que importó las funciones de *first\_udf* módulo correctamente. Ahora escribe

=**hola ("xlwings")** en la celda A1 de la hoja activa en *first\_udf.xlsxm* y después de presionar Enter, verá la fórmula evaluada como se muestra en [Figura 12-2](#).



The screenshot shows a Microsoft Excel interface. The formula bar at the top contains the formula =hello("xlwings"). Below the formula bar is the standard ribbon with tabs A, B, C, and D. The main workspace shows a single row of data in the first column. Cell A1 contains the value "hello xlwings!". Cell A2 is empty. The cell A1 is highlighted with a green border, indicating it is the active cell.

A	B	C	D
1	hello xlwings!		
2			

Figura 12-2. *first\_udf.xlsxm*

Analicemos esto para ver cómo funciona todo: comience por mirar el Hola función en *first\_udf.py* ([Ejemplo 12-1](#)), que es parte del Inicio rápido código que hemos ignorado hasta ahora.

*Ejemplo 12-1. first\_udf.py (extracto)*

```
importar xlwings como xw
```

```
@xw.func
def Hola(nombre):
    regreso F"¡Hola, {nombre}!"
```

Cada función que marques con @xw.func se importará a Excel cuando haga clic en Importar funciones en el complemento xlwings. La importación de una función la hace disponible en Excel para que pueda usarla en sus fórmulas de celda; llegaremos a los detalles técnicos en un momento. @xw.func es un *decorador*, lo que significa que debes colocarlo directamente

encima de la definición de función. Si quieres saber un poco más sobre cómo funcionan los decoradores, echa un vistazo a la barra lateral.

### Decoradores de funciones

Un decorador es un nombre de función que se coloca encima de una definición de función, comenzando con el signo @. Es una forma sencilla de cambiar el comportamiento de una función y xlwings la utiliza para reconocer qué funciones desea que estén disponibles en Excel. Para ayudarlo a comprender cómo funciona un decorador, el siguiente ejemplo muestra la definición de un decorador llamadoverboso que imprimirá algún texto antes y después de la función print\_hello se ejecuta. Técnicamente, el decorador toma la función (print\_hello) y lo proporciona como argumento func al verboso función. La función interna llamada envoltura luego puede hacer lo que sea necesario; en este caso, imprime un valor antes y después de llamar alprint\_hello función. El nombre de la función interna no importa:

En [1]: # Esta es la definición del decorador de funciones.

```
def verboso(func):
    def envoltura():
        impresión("Antes de llamar a la función.")
        func()
        impresión("Después de llamar a la función.")
    regreso envoltura
```

En [2]: # Usando un decorador de funciones

```
@verboso
def print_hello():
    impresión("¡Hola!")
```

En [3]: # Efecto de llamar a la función decorada

```
print_hello()
```

Antes de llamar a la función.

¡Hola!

Después de llamar a la función.

Al final de este capítulo, encontrará [Tabla 12-1](#) con un resumen de todos los decoradores que ofrece xlwings.

De forma predeterminada, si los argumentos de la función son rangos de celdas, xlwings le entrega los valores de estos rangos de celdas en lugar de los xlwings distancia objeto. En la gran mayoría de los casos, esto es muy conveniente y le permite llamar alHola función con una celda como argumento. Por ejemplo, podría escribir "xlwings" en la celda A2, luego cambiar la fórmula en A1 a lo siguiente:

=Hola(A2)

El resultado será el mismo que en [Figura 12-2](#). Te mostraré en la última sección de este capítulo cómo cambiar este comportamiento y hacer que los argumentos lleguen como xlwingsdistancia

objetos en su lugar, como veremos a continuación, hay ocasiones en las que lo necesitará. En VBA, el equivalenteHola La función se vería así:

```
Función Hola(nombre Como Cuerda) Como Cuerda
    Hola = "Hola " Y nombre Y "!"
Función final
```

Cuando hace clic en el botón Importar funciones en el complemento, xlwings inserta un módulo VBA llamado xlwings\_udfs en su libro de Excel. Contiene una función VBA para cada función de Python que importe: estas funciones de envoltura VBA se encargan de ejecutar la función respectiva en Python. Si bien nadie te impide mirar el xlwings\_udfs El módulo VBA al abrir el editor VBA con Alt + F11, puede ignorarlo ya que el código se genera automáticamente y cualquier cambio se perderá al hacer clic en el botón Importar funciones nuevamente. Juguemos ahora con nuestroHola funcionar en first\_udf.py y reemplazar Hola en el valor de retorno con Adiós:

```
@ xw.func
def Hola(nombre):
    regreso F"\Adiós {nombre}!"
```

Para volver a calcular la función en Excel, haga doble clic en la celda A1 para editar la fórmula (o seleccione la celda y presione F2 para activar el modo de edición), luego presione Enter.

Alternativamente, escriba el atajo de teclado Ctrl + Alt + F9: esto fuerza el recálculo de todas las hojas de trabajo en todos los libros abiertos, incluido el Hola fórmula. Tenga en cuenta que F9 (volver a calcular todas las hojas de trabajo en todos los libros abiertos) o Shift + F9 (volver a calcular la hoja de trabajo activa) no volverán a calcular la UDF ya que Excel solo activa un nuevo cálculo de las UDF si cambia una celda dependiente. Para cambiar este comportamiento, puede hacer que la función *volátil* agregando el argumento respectivo al func decorador:

```
@ xw.func(volátil=Cierto)
def Hola(nombre):
    regreso F"\Adiós {nombre}!"
```

Las funciones volátiles se evalúan cada vez que Excel realiza un recálculo, ya sea que las dependencias de la función hayan cambiado o no. Algunas de las funciones integradas de Excel son volátiles como =ALEATORIO () o =AHORA() y usar muchos de ellos hará que su libro de trabajo sea más lento, así que no se exceda. Cuando cambia el nombre o los argumentos de una función o el func decorador como acabamos de hacer, deberá volver a importar su función haciendo clic de nuevo en el botón Importar funciones: esto reiniciará el intérprete de Python antes de importar la función actualizada. Cuando ahora cambie la función deAdiós para Hola, basta con usar los atajos de teclado Shift + F9 o F9 para hacer que la fórmula se recalcule, ya que la función ahora es volátil.



#### Guarde el archivo Python después de cambiarlo

Un error común es olvidar guardar el archivo fuente de Python después de realizar cambios. Por lo tanto, siempre verifique dos veces que el archivo Python esté guardado antes de presionar el botón Importar funciones o volver a calcular las UDF en Excel.

De forma predeterminada, xlwings importa funciones de un archivo de Python en el mismo directorio con el mismo nombre que el archivo de Excel. Cambiar el nombre y mover su archivo fuente de Python requiere cambios similares a los de [Capítulo 10](#), cuando estábamos haciendo lo mismo con RunPython llamadas: continúe y cambie el nombre del archivo de `first_udf.py` para `hola.py`. Para informar a xlwings sobre ese cambio, agregue el nombre del módulo, es decir, `Hola` (sin el `.py` extensión!) a los módulos UDF en el complemento xlwings, como se muestra en [Figura 12-3](#).



Figura 12-3. La configuración de los módulos UDF

Haga clic en el botón Importar funciones para volver a importar la función. Luego, vuelva a calcular la fórmula en Excel para asegurarse de que todo sigue funcionando.



#### Importar funciones de varios módulos de Python

Si desea importar funciones de varios módulos, use un punto y coma entre sus nombres en la configuración de Módulos UDF, por ejemplo, `hola; otro_módulo`.

Ahora adelante y muévete `hola.py` a su escritorio: esto requiere que agregue la ruta de su escritorio a la PYTHONPATH en el complemento xlwings. Como se vio en [Capítulo 10](#), puede usar variables de entorno para lograr esto, es decir, puede establecer el PYTHONPATH configuración en el complemento de `%PERFIL DE USUARIO% \ Escritorio`. Si todavía tienes el camino al `pyscripts` carpeta allí desde [Capítulo 10](#), sobrescríbalo o déjelo ahí, separando las rutas con un punto y coma. Después de estos cambios, vuelva a hacer clic en el botón Importar funciones, luego vuelva a calcular la función en Excel para verificar que todo sigue funcionando.



### Configuración e implementación

En este capítulo, siempre me refiero a cambiar una configuración en el complemento; sin embargo, todo desde [Capítulo 10](#) con respecto a la configuración y el despliegue también se puede aplicar a este capítulo. Esto significa que también se puede cambiar una configuración en la hoja xlwings.conf o en un archivo de configuración ubicado en el mismo directorio que el archivo de Excel. Y en lugar de usar el complemento xlwings, puede usar un libro de trabajo que se haya configurado en modo independiente. Con las UDF, también tiene sentido crear su propio complemento personalizado; esto le permite compartir sus UDF entre todos los libros de trabajo sin tener que importarlos en cada libro de trabajo. Para obtener más información sobre cómo crear su propio complemento personalizado, consulte la [xlwings docs](#).

Si cambia el código Python de su UDF, xlwings automáticamente recoge los cambios cada vez que guarda el archivo Python. Como se mencionó, solo necesita volver a importar sus UDF si cambia algo en el nombre, los argumentos o los decoradores de la función. Sin embargo, si su archivo fuente importa código de otros módulos y cambia algo en estos módulos, la forma más fácil de permitir que Excel recoja todos los cambios es hacer clic en Reiniciar servidor UDF.

En este punto, sabe cómo escribir una UDF simple en Python y cómo usarla en Excel. El estudio de caso de la siguiente sección le presentará UDF más realistas que utilizan pandas DataFrames.

### Estudio de caso: Tendencias de Google

En este caso de estudio, usaremos datos de Google Trends para aprender a trabajar con marcos de datos de pandas y matrices dinámicas, una de las características nuevas más interesantes en Excel que Microsoft lanzó oficialmente en 2020. Luego creamos una UDF que conecta directamente a Google Trends, así como a uno que utiliza un DataFrame trama método. Para concluir esta sección, veremos cómo funciona la depuración con UDF. ¡Comencemos con una breve introducción a Tendencias de Google!

### Introducción a Tendencias de Google

[Tendencias de Google](#) es un servicio de Google que le permite analizar la popularidad de las consultas de búsqueda de Google a lo largo del tiempo y en todas las regiones. [Figura 12-4](#), muestra Google Trends después de agregar algunos lenguajes de programación populares, seleccionando Todo el mundo como la región y 1/1/16 - 12/26/20 como el intervalo de tiempo. Cada término de búsqueda ha sido seleccionado con el *Lenguaje de programación* contexto que aparece en un menú desplegable después de escribir el término de búsqueda. Esto asegura que ignoremos Python, la serpiente y Java, la isla. Google indexa los datos dentro del período de tiempo y la ubicación seleccionados con 100 que representa el interés de búsqueda máximo. En nuestra muestra, significa que dentro del período de tiempo dado

y ubicación, el mayor interés de búsqueda se dio en Java en febrero de 2016. Para obtener más detalles sobre Tendencias de Google, eche un vistazo a su [entrada en el blog](#).

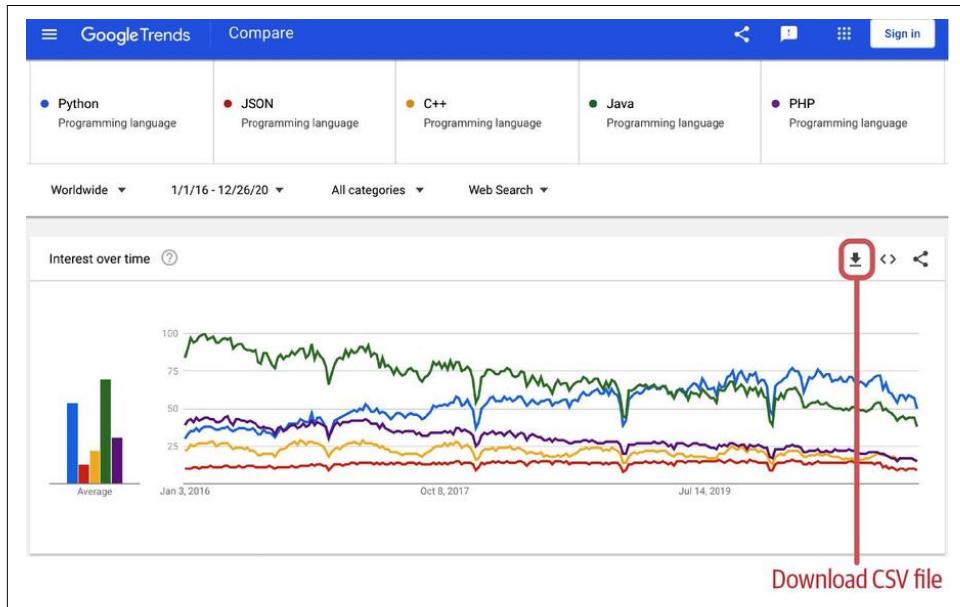


Figura 12-4. Interés a lo largo del tiempo; fuente de datos [Tendencias de Google](#)



#### Muestras aleatorias

Los números de Tendencias de Google se basan en muestras aleatorias, lo que significa que es posible que vea una imagen ligeramente diferente de [Figura 12-4](#), incluso si usa la misma ubicación, período de tiempo y términos de búsqueda que en la captura de pantalla.

Pulsé el botón de descarga que ves en [Figura 12-4](#). para obtener un archivo CSV desde donde copié los datos en el libro de Excel de un Inicio rápido proyecto. En la siguiente sección, le mostraré dónde encontrar este libro de trabajo; lo usaremos para analizar los datos con una UDF directamente desde Excel.

## Trabajar con DataFrames y Matrices dinámicas

Habiendo llegado tan lejos en el libro, no debería sorprenderse de que los marcos de datos de pandas también sean los mejores amigos de las UDF. Para ver cómo los DataFrames y las UDF funcionan juntos y para aprender acerca de las matrices dinámicas, navegue a la `describe` carpeta en el `udfs` directorio del repositorio complementario y abrir `describe.xlsx` en Excel y `describe.py` en VS Code. El archivo de Excel contiene los datos de Google Trends y en el archivo de Python, encontrará una función simple para comenzar, como se muestra en [Ejemplo 12-2](#).

### Ejemplo 12-2. *describe.py*

```
importar xlwings como xw
importar pandas como pd

@ xw.func
@ xw.arg("df", pd.Marco de datos, índice=Cierto, encabezamiento=Cierto)
)def describir(df):
    regreso df.describir()
```

En comparación con el Hola función de la Inicio rápido proyecto, notarás un segundo decorador:

```
@ xw.arg("df", pd.Marco de datos, índice=Cierto, encabezamiento=Cierto)
```

arg es la abreviatura de *argumento* y le permite aplicar los mismos convertidores y opciones que estaba usando en [Capítulo 9](#) cuando estaba introduciendo la sintaxis xlwings. En otras palabras, el decorador ofrece la misma funcionalidad para UDF que el opción método para xlwings distancia objetos. Formalmente, esta es la sintaxis del argumento del decorador:

```
@ xw.arg("nombre_argumento", convertir=Ninguno, Opción 1=valor1, opcion 2=valor2, ...)
```

Para ayudarlo a establecer la conexión con [Capítulo 9](#), el equivalente de la describir La función en forma de un script se ve así (esto supone que *describe.xlsm* está abierto en Excel y que la función se aplica al rango A3: F263):

```
importar xlwings como xw
importar pandas como pd
```

```
rango de datos = xw.Libro("describe.xlsm").hojas[0]["A3: F263"]
df = rango de datos.opciones(pd.Marco de datos, índice=Cierto, encabezamiento=Cierto).valor
df.describir()
```

Las opciones índice y encabezamiento no sería necesario ya que están usando los argumentos predeterminados, pero los incluí para mostrarle cómo se aplican con las UDF. Con *describe.xlsm* como su libro de trabajo activo, haga clic en el botón Importar funciones, luego escriba **= describir (A3: F263)** en una celda libre, en H3, por ejemplo. Lo que sucede cuando presiona Enter depende de su versión de Excel, más específicamente si su versión de Excel es lo suficientemente reciente como para admitir *matrices dinámicas*. Si es así, verá la situación como se muestra en [Figura 12-5](#), es decir, la salida del describir función en las celdas H3: M11 está rodeado por un borde azul delgado. Solo podrá ver el borde azul si su cursor está dentro de la matriz, y es tan sutil que puede tener problemas para verlo claramente si mira la captura de pantalla en una versión impresa del libro. Veremos cómo se comportan las matrices dinámicas en un momento y también puedes aprender más sobre ellas en la barra lateral. ["Matrices dinámicas"](#) en la página 263.

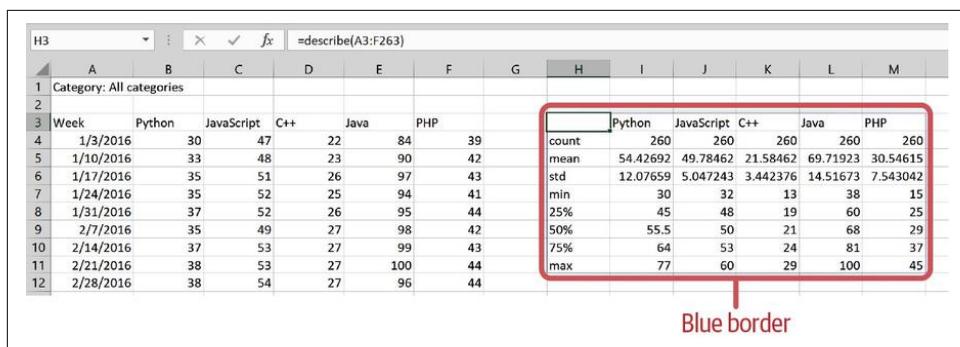


Figura 12-5. Los describir función con matrices dinámicas

Sin embargo, si está utilizando una versión de Excel que no admite matrices dinámicas, parecerá que no sucede nada: de forma predeterminada, la fórmula solo devolverá la celda superior en H3, que está vacía. Para solucionar este problema, utilice lo que Microsoft llama hoy en día legado. *Matrices CSE*. Las matrices CSE deben confirmarse escribiendo la combinación de teclas Ctrl + Shift + Enter en lugar de presionar solo Enter, de ahí su nombre. Veamos cómo funcionan en detalle:

- Asegúrese de que H3 sea una celda vacía seleccionándola y presionando la tecla Suprimir.
- Seleccione el rango de salida comenzando en la celda H3 y luego seleccione todas las celdas en el camino a M11.
- Con el rango H3: M11 seleccionado, escriba la fórmula **=describir (A3: F263)** y luego confirme presionando Ctrl + Shift + Enter.

Ahora debería ver casi la misma imagen que en Figura 12-5 con estas diferencias:

- No hay borde azul alrededor del rango H3: M11.
- La fórmula muestra llaves alrededor para marcarla como una matriz CSE: **{= describir (A3: F263)}**.
- Mientras elimina matrices dinámicas yendo a la celda superior izquierda y presionando la tecla Eliminar, con las matrices CSE, siempre debe seleccionar primero la matriz completa para poder eliminarla.

Hagamos ahora nuestra función un poco más útil introduciendo un parámetro opcional llamado selección que nos permitirá especificar qué columnas queremos incluir en nuestra salida. Si tiene muchas columnas y solo desea incluir un subconjunto en el describir función, esto puede convertirse en una característica útil. Cambie la función de la siguiente manera:

```

@ xw.func
@ xw.arg("df", pd.Marco de datos)1
describir(df, selección=Ninguno):2

```

```

si selección no es Ninguno:
    regreso df.loc[:, selección].describir() ③
demás:
    regreso df.describir()

```

- ①** Dejé el índice y encabezamiento argumentos, ya que utilizan los valores predeterminados, pero siéntase libre de dejarlos en.
- ②** Agregar el parámetro selección y hacerlo opcional asignando Ninguno como su valor predeterminado.
- ③** Si selección se proporciona, filtre las columnas del DataFrame en función de él.

Una vez que haya cambiado la función, asegúrese de guardarla y luego presione el botón Importar funciones en el complemento xlwings; esto es necesario ya que hemos agregado un nuevo parámetro. Escribir **Selección** en la celda A2 y **CIERTO** en las celdas B2: F2. Finalmente, ajuste su fórmula en la celda H3 dependiendo de si tiene matrices dinámicas o no:

#### *Con matrices dinámicas*

Seleccione H3, luego cambie la fórmula a **=describir (A3: F263, B2: F2)** y presione Enter.

#### *Sin matrices dinámicas*

Comenzando en la celda H3, seleccione H3: M11, y luego presione F2 para activar el modo de edición de la celda H3 y cambie la fórmula a **=describir (A3: F263, B2: F2)**. Para finalizar, presione Ctrl + Shift + Enter.

Para probar la función mejorada, cambiemos Java CIERTO en la celda E2 para FALSO y vea qué sucede: con las matrices dinámicas, verá que la tabla se reduce mágicamente en una columna. Sin embargo, con las matrices CSE heredadas, terminará con una columna fea llena de #N / A valores, como se muestra en [Figura 12-6..](#)

Para solucionar este problema, xlwings puede cambiar el tamaño de las matrices CSE heredadas haciendo uso del decorador de retorno. Agréguelo cambiando su función de esta manera:

```

@ xw.func
@ xw.arg("df", pd.Marco de datos)
@ xw.ret(expandir="mesa") ①
def describir(df, selección=Ninguno):
    si selección no es Ninguno:
        regreso df.loc[:, selección].describir()
    demás:
        regreso df.describir()

```

- ①** Añadiendo el decorador de devoluciones con la opción expand = "tabla", xlwings cambiará el tamaño de la matriz CSE para que coincida con las dimensiones del DataFrame devuelto.

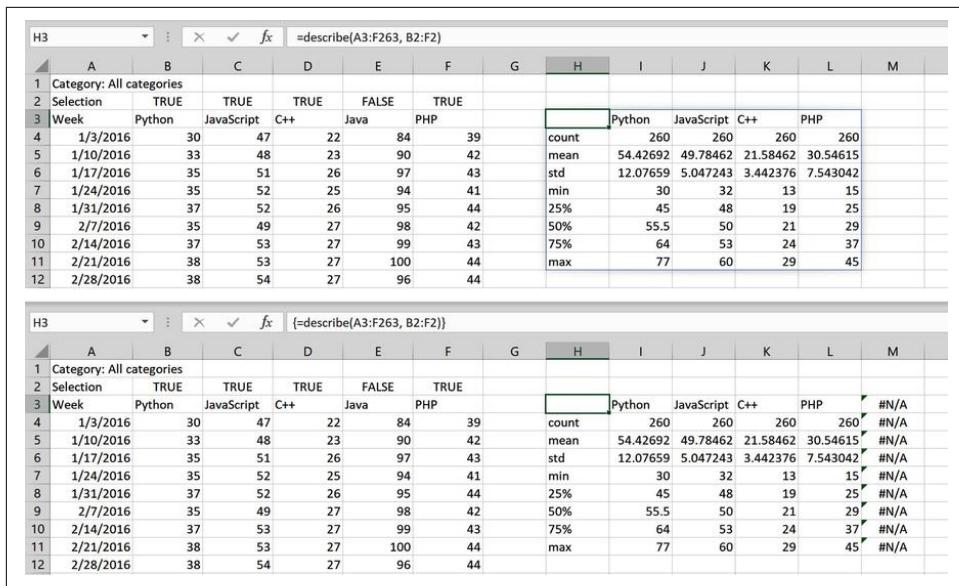


Figura 12-6. Matrices dinámicas (superior) frente a matrices CSE (inferior) después de excluir una columna

Después de agregar el decorador de retorno, guarde el archivo fuente de Python, cambie a Excel y presione Ctrl + Alt + F9 para volver a calcular: esto cambiará el tamaño de la matriz CSE y eliminará el #N / A columna. Dado que esta es una solución alternativa, le recomiendo encarecidamente que haga todo lo que esté a su alcance para tener en sus manos una versión de Excel que admita matrices dinámicas.



#### Orden de decoradores de funciones

Asegúrese de colocar el `xw.func` decorador en la parte superior del `xw.arg` `xw.ret` decoradores; tenga en cuenta que el orden `dexw.arg` y `xw.ret` importa.

los *decorador de retorno* funciona conceptualmente de la misma manera que el decorador de argumentos, con la única diferencia de que no es necesario especificar el nombre de un argumento. Formalmente, su sintaxis se ve así:

```
@xw.ret(convertir=Ninguno, Opción 1=valor1, opción 2=valor2, ...)
```

Por lo general, no es necesario que proporcione una convertir argumento como xlwings reconoce el tipo de valor de retorno automáticamente; ese es el mismo comportamiento que vimos en [Capítulo 9](#) con el opciones método al escribir valores en Excel.

Como ejemplo, si desea suprimir el índice del DataFrame que devuelve, use este decorador:

```
@xw.ret(indice=False)
```

### Matrices dinámicas

Habiendo visto cómo funcionan las matrices dinámicas en el contexto de la describir función, estoy bastante seguro de que estará de acuerdo en que son una de las adiciones más fundamentales y emocionantes a Excel que Microsoft ha creado en mucho tiempo. Se presentaron oficialmente en 2020 a los suscriptores de Microsoft 365 que utilizan la versión más reciente de Excel. Para ver si su versión es lo suficientemente reciente, verifique la existencia de la nueva ÚNICO función: empezar a escribir =ÚNICO en una celda y si Excel sugiere el nombre de la función, se admiten matrices dinámicas. Si usa Excel con una licencia permanente en lugar de como parte de la suscripción de Microsoft 365, es probable que lo obtenga con la versión que se anunció para su lanzamiento en 2021 y que presumiblemente se llamará Office 2021. Aquí hay algunas notas técnicas sobre el comportamiento de las matrices dinámicas:

- Si las matrices dinámicas sobrescriben una celda con un valor, obtendrá un #;DERRAMAR! error. Después de hacer espacio para la matriz dinámica eliminando o moviendo la celda que está en el camino, la matriz se escribirá. Tenga en cuenta que el decorador de retorno xlwings con expandir = "tabla" es menos inteligente y sobrescribirá los valores de celda existentes sin previo aviso.
- Puede hacer referencia al rango de una matriz dinámica utilizando la celda superior izquierda seguida de un signo #. Por ejemplo, si su matriz dinámica está en el rango A1: B2 y desea resumir todas las celdas, escriba=SUMA(A1 #).
- Si alguna vez desea que sus matrices se comporten como las matrices CSE heredadas nuevamente, comience su fórmula con un signo @, por ejemplo, para que una multiplicación de matrices devuelva una matriz CSE heredada, use = @MMULT () .

Descargar un archivo CSV y copiar / pegar los valores en un archivo de Excel funcionó bien para este ejemplo introductorio de DataFrame, pero copiar / pegar es un proceso tan propenso a errores que querrá deshacerse de él siempre que pueda. Con Google Trends, de hecho puede hacerlo, ¡y la siguiente sección le muestra cómo hacerlo!

### Obtención de datos de Tendencias de Google

Los ejemplos anteriores eran todos muy simples, básicamente envolviendo una sola función pandas. Para tener en nuestras manos un caso más real, creamos una UDF que descargue los datos directamente desde Google Trends para que ya no tenga que conectarse a Internet y descargar un archivo CSV manualmente. Google Trends no tiene una API oficial (interfaz de programación de aplicaciones), pero hay un paquete de Python llamado [pytrends](#) que llena el vacío. No ser una API oficial significa que Google puede cambiarla en cualquier momento.

quieren, por lo que existe el riesgo de que los ejemplos de esta sección dejen de funcionar en algún momento. Sin embargo, dado que Pytrends ha existido durante más de cinco años en el momento de escribir este artículo, también existe una posibilidad real de que se actualice para reflejar los cambios y hacer que funcione nuevamente. En cualquier caso, sirve de buen ejemplo para mostrarte que *hay un paquete de Python para casi cualquier cosa*—Un reclamo que hice en [Capítulo 1](#). Si estuviera restringido a usar Power Query, probablemente necesitaría invertir mucho más tiempo para que algo funcione; yo, al menos, no pude encontrar una solución plug-and-play que esté disponible de forma gratuita . Dado que pytrends no es parte de Anaconda y tampoco tiene un paquete oficial de Conda, instalémoslo con pip, si aún no lo ha hecho:

```
(base)> pip instalar pytrends
```

Para replicar el caso exacto de la versión en línea de Tendencias de Google como se muestra en [Figura 12-4.](#), necesitaremos encontrar los identificadores correctos para los términos de búsqueda con el contexto "Lenguaje de programación". Para hacer esto, pytrends puede imprimir los diferentes contextos de búsqueda *otipos* que sugiere Google Trends en el menú desplegable. En el siguiente ejemplo de código, medio representa *Identificador de máquina*, que es el ID que estamos buscando:

En [4]: **de pytrends.request importar TrendReq**

En [5]: *# Primero, creamos una instancia de un objeto TrendRequest*  
tendencia = TrendReq()

En [6]: *# Ahora podemos imprimir las sugerencias como aparecerían.*  
*# en línea en el menú desplegable de Tendencias de Google después de escribir "Python"*  
tendencia.sugerencias("Pitón")

Fuera [6]: `[{'mid': '/m/05z1_1', 'title': 'Python', 'type': 'Programming language'}, {'mid': '/m/05tb5', 'title': 'Familia Python', 'type': 'Snake'}, {'mid': '/m/0cv6_m', 'title': 'Pythons', 'type': 'Snake'}, {'mid': '/m/06bxrb', 'title': 'CPython', 'type': 'Topic'}, {'mid': '/g/1q6j3gsvm', 'título': 'python', 'tipo': 'Tema'}]`

Repetir esto para los otros lenguajes de programación nos permite recuperar el correcto medio para todos ellos, y podemos escribir la UDF como se muestra en [Ejemplo 12-3](#). Encontrarás el código fuente en el *Tendencias de Google* directorio dentro del *udfs* carpeta del repositorio complementario.

*Ejemplo 12-3. Losget\_interest\_over\_time función en google\_trends.py (extracto con las declaraciones de importación relevantes)*

```
importar pandas como pd  
de pytrends.request importar TrendReq  
importar xlwings como xw
```

```
@ xw.func(call_in_wizard=False) ❶  
@ xw.arg("medios", Doc="ID de máquina: un rango de 5 celdas como máximo")@ ❷  
xw.arg("fecha de inicio", Doc="Una celda con formato de fecha")@ xw.arg("fecha  
final", Doc="Una celda con formato de fecha")
```

```

def get_interest_over_time(medios, fecha de inicio, fecha final):
    """ Consultar Tendencias de Google: reemplaza el ID de máquina (medio) de
    los lenguajes de programación comunes con su equivalente legible por
    humanos en el valor de retorno, por ejemplo, en lugar de "/m/05z1_",
    devuelve "Python".
    """
    # Verificar y transformar parámetros
    afirmar len(medios) <= 5, "Demasiados medios (máx.: 5)"      ④
    fecha de inicio = fecha de inicio.fecha().isoformat()fecha      ⑤
    final = fecha final.fecha().isoformat()

    # Realice la solicitud de Tendencias de Google y devuelva el DataFrame
    tendencia = TrendReq(se acabó el tiempo⑥)⑩tendencia.build_payload(kw_list=
    medios,
                           periodo de tiempo=F"(fecha de inicio Fecha de término)")      ⑦
    df = tendencia.interes_sobre_tiempo()      ⑧

    # Reemplaza "mid" de Google por una palabra legible por humanos.
    medios = {"// m / 05z1_": "Pitón", "// m / 02p97": "JavaScript",
              "// m / 0jgqq": "C ++", "// m / 07sbkfb": "Java", "// m / 060kv": "PHP"}df = df.
    rebautizar(columnas=medios)      ⑨

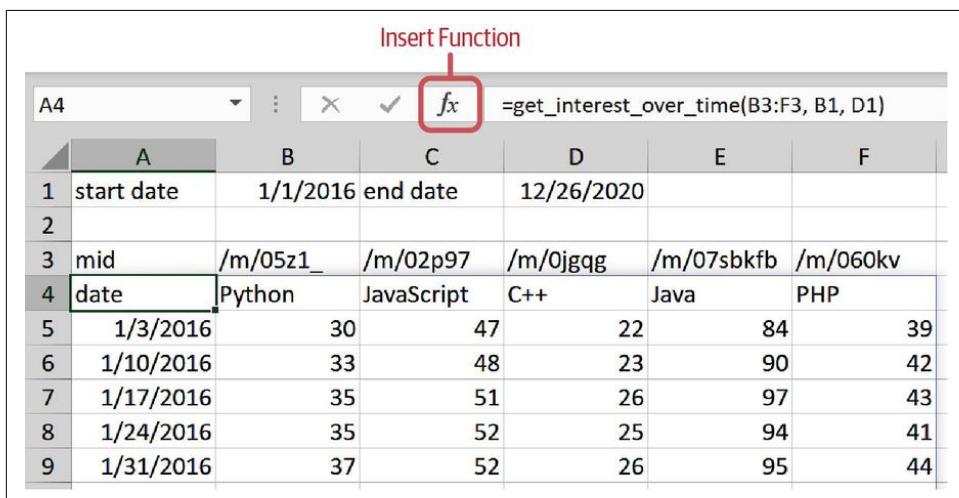
    # Suelta la columna isPartialregreso df.
    soltar(columnas="isPartial")      ⑩

```

- ➊ De forma predeterminada, Excel llama a la función cuando la abre en el Asistente para funciones. Como esto puede hacer que sea más lento, especialmente con las solicitudes de API involucradas, lo apagaremos.
- ➋ Opcionalmente, agregue una cadena de documentos para el argumento de la función, que se mostrará en el Asistente de funciones cuando edite el argumento respectivo, como en [Figura 12-8..](#)
- ➌ La cadena de documentación de la función se muestra en el Asistente de funciones, como en [Figura 12-8..](#)
- ➍ la afirmar declaración es una manera fácil de generar un error en caso de que el usuario proporcione demasiados medios. Google Trends permite un máximo de cinco medios por consulta.
- ➎ pytrends espera las fechas de inicio y finalización como una sola cadena en el formulario AAAA-MM-DD AAAA-MM-DD. Como proporcionamos las fechas de inicio y finalización como celdas con formato de fecha, llegarán como fecha y hora objetos. Llamando alfecha y isoformato los métodos en ellos los formatearán correctamente.
- ➏ Estamos creando una instancia de pytrends solicitud objeto. Al establecer else acabó el tiempo a diez segundos, reducimos el riesgo de ver una request.exceptions.ReadTimeout error, que ocurre ocasionalmente si Google Trends tarda un poco más en responder. Si aún ve este error, simplemente ejecute la función nuevamente o aumente el tiempo de espera.

- 7 Proporcionamos el kw\_list y periodo de tiempo argumentos al objeto de la solicitud.
- 8 Realizamos la solicitud real llamando interes\_sobre\_tiempo, que devolverá un DataFrame de pandas.
- 9 Cambiamos el nombre del medios con su equivalente legible por humanos.
- 10 La última columna se llama isPartial. Ciertamente indica que el intervalo actual, por ejemplo, una semana, todavía está en curso y, por lo tanto, aún no tiene todos los datos. Para mantener las cosas simples y estar en línea con la versión en línea, descartamos esta columna al devolver el DataFrame.

Ahora abierto *google\_trends.xlsxm* desde el repositorio complementario, haga clic en Importar funciones en el complemento xlwings, y luego llame al get\_interest\_over\_time función de la celda A4, como se muestra en [Figura 12-7..](#)



*Figura 12-7. google\_trends.xlsxm*

Para obtener ayuda con respecto a los argumentos de la función, haga clic en el botón Insertar función a la izquierda de la barra de fórmulas mientras la celda A4 está seleccionada: esto abrirá el Asistente de funciones donde encontrará sus UDF bajo el xlwings categoría. Después de seleccionar get\_interest\_over\_time, verá el nombre de los argumentos de la función, así como la cadena de documentos como descripción de la función (restringida a los primeros 256 caracteres): consulte [Figura 12-8..](#) Alternativamente, comience a escribir= **get\_interest\_over\_time** ( en la celda A4 (incluido el paréntesis de apertura) antes de presionar el botón Insertar función; esto lo llevará directamente a la vista que se muestra en [Figura 12-8..](#) Tenga en cuenta que las UDF devuelven las fechas sin formato. Para solucionar este problema, haga clic con el botón derecho en la columna con las fechas, seleccione Formato de celdas y luego seleccione el formato de su elección en la categoría Fecha.

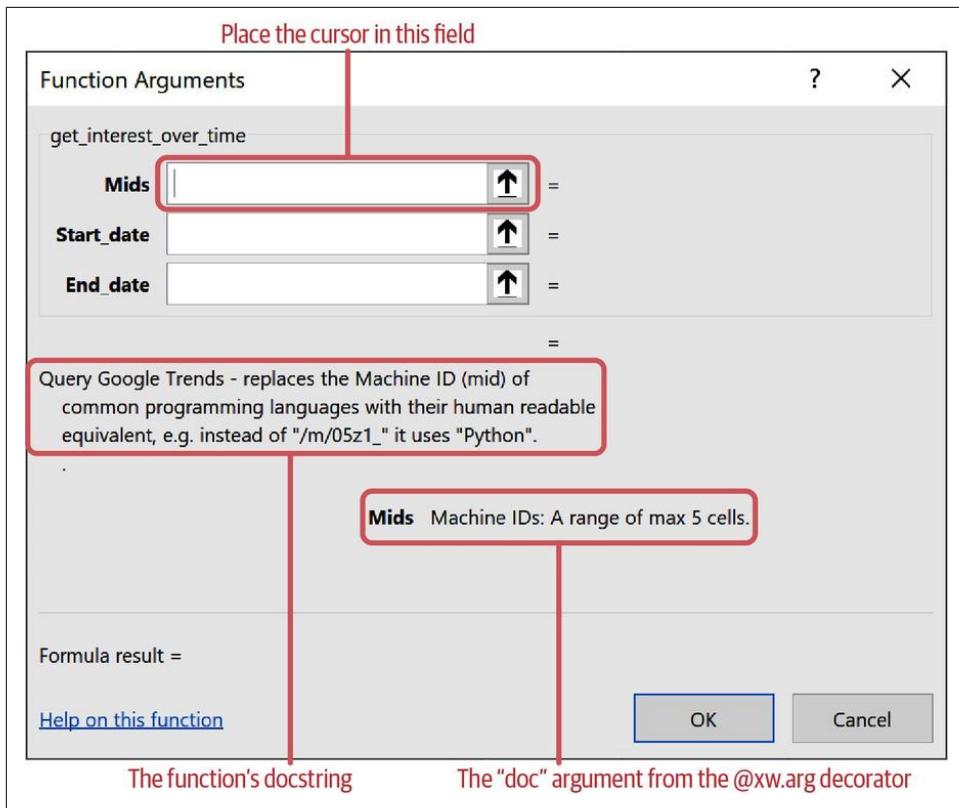


Figura 12-8. El asistente de funciones

Si miras de cerca Figura 12-7., puedes ver por el borde azul alrededor de la matriz de resultados que estoy usando matrices dinámicas nuevamente. Como la captura de pantalla se recorta en la parte inferior y la matriz comienza en el extremo izquierdo, solo verás los bordes superior y derecho que comienzan en la celda A4, e incluso pueden ser difíciles de reconocer en la captura de pantalla. Si tu versión de Excel no admite matrices dinámicas, utilice la solución agregando el siguiente decorador de retorno a la función get\_interest\_over\_time (debajo de los decoradores existentes):

```
@xw.ret(expandir="mesa")
```

Ahora que sabe cómo trabajar con UDF más complicadas, ¡veamos cómo podemos usar gráficos con UDF!

## Trazado con UDF

Como recordarás de Capítulo 5, llamando a un DataFrame trama El método devuelve un gráfico Matplotlib de forma predeterminada. En capítulos 9 y 11, ya hemos visto cómo agregar un gráfico como una imagen a Excel. Cuando se trabaja con UDF, existe una forma sencilla de

producir parcelas: eche un vistazo a la segunda función en *google\_trends.py*, se muestra en la Ejemplo 12-4.

Ejemplo 12-4. *la trama función en google\_trends.py (extracto con las declaraciones de importación relevantes)*

```
importar xlwings como xw
importar pandas como pd
importar matplotlib.pyplot como plt
```

```
@ xw.func
@ xw.arg("df", pd.Marco de datos)
def trama(df, nombre, llamador): ①
    plt.estilo.usar("marinero") si ②
        no df.vacío: ③
            llamador.hoja.imágenes.agregar(df.trama().get_figure(), ④
                cima=llamador.compensar(row_offset=1).cima, ⑤
                izquierda=llamador.izquierda,nombre=nombre,
                actualizar=Cierto) ⑥
    regreso F"<Parcela: {nombre}>" ⑦
```

- ① los llamador El argumento es un argumento especial reservado por xlwings: este argumento no se expondrá cuando llame a la función desde una celda de Excel. En lugar de, llamador será proporcionado por xlwings detrás de escena y corresponde a la celda desde la cual está llamando a la función (en forma de xlwings distancia objeto). Tener el distancia El objeto de la celda que realiza la llamada hace que sea fácil de colocar el gráfico mediante el uso de cima y izquierda argumentos de imágenes.add. los nombre El argumento definirá el nombre de la imagen en Excel.
- ② Establecemos el marinero estilo para hacer la trama visualmente más atractiva.
- ③ Solo llame al trama método si el DataFrame no está vacío. Llamando altrama en un DataFrame vacío generaría un error.
- ④ get\_figure () devuelve el objeto de figura Matplotlib de un diagrama de DataFrame, que es lo que pictures.add espera.
- ⑤ Los argumentos cima y izquierda sólo se utilizan cuando inserta el gráfico por primera vez. Los argumentos proporcionados colocarán la gráfica en un lugar conveniente, una celda debajo de aquella desde donde llama a esta función.
- ⑥ El argumento actualización = Verdadero se asegura de que las llamadas a funciones repetidas actualizarán la imagen existente con el nombre proporcionado en Excel, sin cambiar su

posición o tamaño. Sin este argumento, xlwings se quejaría de que ya existe una imagen con ese nombre en Excel.

- 7 Si bien no es estrictamente necesario devolver nada, su vida será mucho más fácil si devuelve una cadena: esto le permite reconocer en qué parte de su hoja se encuentra su función de trazado.

En *google\_trends.xlsm*, en la celda H3, llame al trama funcionar así:

```
=trama(A4:F263, "Historia")
```

Si su versión de Excel admite matrices dinámicas, use A4 # en lugar de A4: F263 para hacer que la fuente sea dinámica como se muestra en **Figura 12-9..**

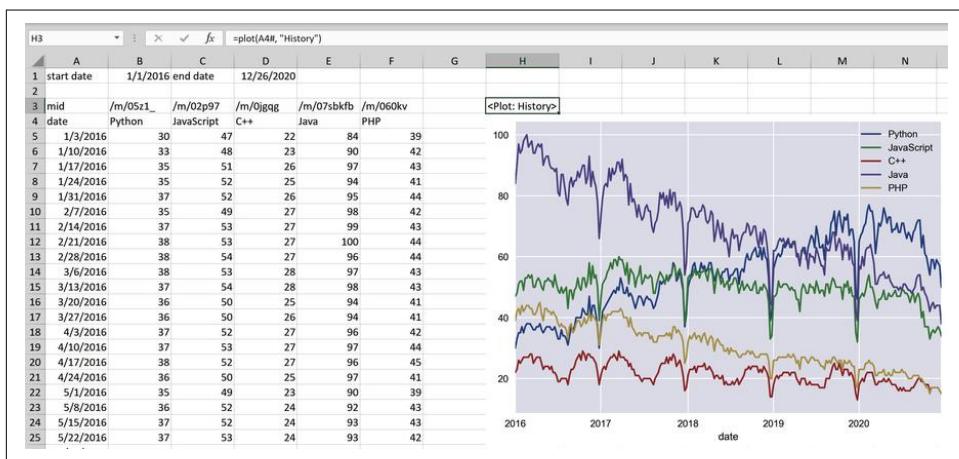


Figura 12-9. *la trama función en acción*

Supongamos que está un poco confundido por la forma en que `get_interest_over_time` la función funciona. Una opción para comprender mejor es depurar el código; la siguiente sección le muestra cómo funciona esto con las UDF.

## Depurar UDF

Una forma sencilla de depurar una UDF es utilizar la impresión función. Si tiene habilitada la configuración Mostrar consola en el complemento xlwings, podrá imprimir el valor de una variable en el símbolo del sistema que aparece cuando llama a su UDF. Una opción un poco más cómoda es utilizar el depurador de VS Code, que le permitirá hacer una pausa en los puntos de interrupción y recorrer el código línea por línea. Para usar el depurador de VS Code (o el depurador de cualquier otro IDE), deberá hacer dos cosas:

1. En el complemento de Excel, active la casilla de verificación Depurar UDF. Esto evita que Excel inicie Python automáticamente, lo que significa que debe hacerlo manualmente como se explica en el siguiente punto.
2. La forma más fácil de ejecutar el servidor Python UDF manualmente es agregando las siguientes líneas al final del archivo que está intentando depurar. Ya agregué estas líneas en la parte inferior de *lagoogle\_trends.py* archivo en el repositorio complementario:

```
si __nombre__ == "__principal__":
    xv.atender()
```

Como puede recordar de [Capítulo 11](#), esta si La declaración se asegura de que el código solo se ejecute cuando se ejecuta el archivo como un script; no se ejecuta cuando se importa el código como un módulo. Con el comando agregado, ejecutar *google\_trends.py* en VS Code en modo de depuración presionando F5 y seleccionando "Archivo Python" —asegúrese de no ejecutar el archivo haciendo clic en el botón Ejecutar archivo, ya que esto ignoraría los puntos de interrupción.

Establezcamos un punto de interrupción en la línea 29 haciendo clic a la izquierda del número de línea. Si no está familiarizado con el uso del depurador de VS Code, eche un vistazo a [apéndice B](#) donde lo presento con más detalle. Cuando vuelva a calcular la celda A4, la llamada a la función se detendrá en el punto de interrupción y podrá inspeccionar las variables. Lo que siempre es útil durante la depuración es ejecutar `df.info()`. Active la pestaña Consola de depuración, escriba `df.info()` en el mensaje en la parte inferior, y confirme presionando Enter, como se muestra en [Figura 12-10](#).



### Depurando con VS Code y Anaconda

Esta es la misma advertencia que en [Capítulo 11](#): en Windows, cuando ejecuta el depurador de VS Code por primera vez con código que usa pandas, es posible que reciba un error: "Se ha producido una excepción: ImportError, no se pueden importar las dependencias requeridas: numpy". Esto sucede porque el depurador está en funcionamiento antes de que el entorno Conda se haya activado correctamente. Como solución alternativa, detenga el depurador haciendo clic en el icono de detención y presione F5 nuevamente; funcionará la segunda vez.

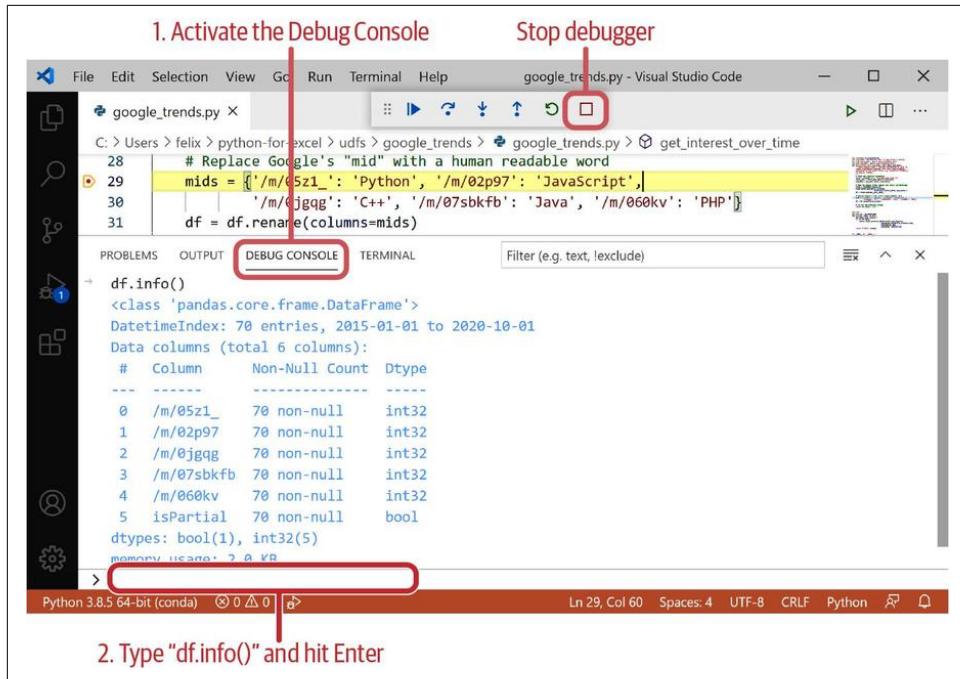


Figura 12-10. Usar la consola de depuración mientras el código está en pausa en un punto de interrupción

Si mantiene su programa en pausa durante más de noventa segundos en un punto de interrupción, Excel le mostrará una ventana emergente que dice que "Microsoft Excel está esperando que otra aplicación complete una acción OLE". Esto no debería tener un impacto en su experiencia de depuración más que tener que confirmar la ventana emergente para que desaparezca una vez que haya terminado con la depuración. Para finalizar esta sesión de depuración, haga clic en el botón Detener en VS Code (consulte [Figura 12-10](#)) y asegúrese de desmarcar la configuración Depurar UDF nuevamente en el complemento de cinta de xlwings. Si olvida desmarcar la configuración de depuración de UDF, sus funciones devolverán un error la próxima vez que las recalcule.

Esta sección le mostró la funcionalidad UDF más comúnmente utilizada trabajando con el estudio de caso de Tendencias de Google. La siguiente sección abordará algunos temas avanzados, incluido el rendimiento de UDF y laxw.sub decorador.

## Temas avanzados de UDF

Si usa muchas UDF en su libro de trabajo, el rendimiento puede convertirse en un problema. Esta sección comienza mostrándole las mismas optimizaciones de rendimiento básicas que hemos visto en [Capítulo 9](#), pero se aplica a las UDF. La segunda parte trata del almacenamiento en caché, una técnica adicional de optimización del rendimiento que podemos usar con las UDF. En el camino, también aprenderemos cómo hacer que los argumentos de las funciones lleguen como xlwingsdistanza objetos en lugar de

como valores. Al final de esta sección, le presentaré lasxw.sub decorador que puede utilizar como alternativa al RunPython llame si trabaja exclusivamente en Windows.

## Optimización básica del rendimiento

Esta parte analiza dos técnicas de optimización del rendimiento: cómo minimizar las llamadas de aplicaciones cruzadas y cómo utilizar el convertidor de valores sin procesar.

### Minimice las llamadas entre aplicaciones

Como probablemente recuerde de [Capítulo 9](#), las llamadas entre aplicaciones, es decir, las llamadas entre Excel y Python, son relativamente lentas, por lo que cuantas menos UDF tenga, mejor. Por lo tanto, debe trabajar con matrices siempre que pueda; tener una versión de Excel que admita matrices dinámicas definitivamente facilita esta parte. Cuando trabaja con pandas DataFrames, no hay mucho que pueda salir mal, pero hay ciertas fórmulas en las que es posible que no piense en usar matrices automáticamente. Considere el ejemplo de [Figura 12-11](#) que calcula los ingresos totales como la suma de una tarifa base dada más una tarifa variable determinada por los usuarios multiplicado por el precio.

The figure shows two side-by-side Excel spreadsheets. The left spreadsheet, titled 'Single-cell formulas', contains a single formula =revenue(\$B\$5, \$A9, B\$8) in cell B9. The right spreadsheet, titled 'Array-based formulas', contains a formula =revenue2(H5, G9:G13, H8:K8) in cell H9. Both spreadsheets show a table of data from rows 9 to 13, with columns for Users (1 to 20), Price (30 to 700), and Total Revenue (130 to 580). Red boxes highlight the formula cells in both cases, and red arrows point from these formulas to their respective ranges in the arrays below them.

=revenue(\$B\$5, \$A9, B\$8)											
A	B	C	D	E	F	G	H	I	J	K	
1 Total Revenue											
2											
3 Single-cell formulas						Array-based formulas					
4											
5 Base Fee	100					Base Fee	100				
6											
7 Price						Price					
8 Users	30	28	26	24		Users	30	28	26	24	
9 1	130	128	126	124		1	130	128	126	124	
10 2	160	156	152	148		2	160	156	152	148	
11 5	250	240	230	220		5	250	240	230	220	
12 10	400	380	360	340		10	400	380	360	340	
13 20	700	660	620	580		20	700	660	620	580	

Figura 12-11. Fórmulas de celda única (izquierda) frente a fórmulas basadas en matrices (derecha)

### Fórmulas de celda única

La mesa de la izquierda en [Figura 12-11](#) usa la fórmula =ingresos (\$ B \$ 5, \$ A9, B \$ 8) en la celda B9. Esta fórmula se aplica luego a todo el rango B9: E13. Esto significa que tiene 20 fórmulas de una sola celda que llaman a la función ingresos.

### Fórmulas basadas en matrices

La mesa de la derecha en [Figura 12-11](#) usa la fórmula =ingresos2 (H5, G9: G13, H8: K8). Si no tiene matrices dinámicas en su versión de Excel, deberá agregar el decorador xw.ret (expandir = "tabla") al ingresos2 función o convierta la matriz en una matriz CSE heredada seleccionando H9: K13, presionando F2 para editar la fórmula y confirmando con Ctrl + Shift + Enter. A diferencia de la fórmula de celda única, esta versión solo llama a la función ingresos2 una vez.

Puede ver el código Python para las dos UDF en [Ejemplo 12-5](#), y encontrará el archivo fuente en el *ingresos* carpeta dentro de la *udfs* directorio del repositorio complementario.

### *Ejemplo 12-5. ingresos.py*

```
importar numpy como notario público
importar xlwings como xw

@ xw.func
def ingresos(tarifa_base, usuarios, precio):
    regreso tarifa_base + usuarios * precio

@ xw.func
@ xw.arg("usuarios", notario público.formación, ndim=2)@
xw.arg("precio", notario público.formación)
def ingresos2(tarifa_base, usuarios, precio):
    regreso tarifa_base + usuarios * precio
```

Cuando cambie la tarifa base en la celda B5 o H5 respectivamente, verá que el ejemplo de la derecha será mucho más rápido que el de la izquierda. La diferencia en las funciones de Python es mínima y solo difiere en los decoradores de argumentos: la versión basada en matrices se lee en usuarios y precios como matriz NumPy, la única advertencia aquí es leer en usuarios como un vector de columna bidimensional estableciendo `ndim = 2` en el decorador de argumentos. Probablemente recuerde que las matrices NumPy son similares a DataFrames pero sin índice o encabezado y con un solo tipo de datos, pero si desea una actualización más detallada, eche otro vistazo a [Capítulo 4](#).

#### **Usando valores brutos**

El uso de valores sin procesar significa que está omitiendo los pasos de preparación y limpieza de datos que xlwings realiza además de pywin32, la dependencia de xlwings en Windows. Esto, por ejemplo, significa que ya no puede trabajar con DataFrames directamente ya que pywin32 no los comprende, pero eso puede no ser un problema si trabaja con listas o matrices NumPy. Para usar UDF con valores brutos, use la cadena `cruento` como el convertir argumento en el argumento o decorador de retorno. Este es el equivalente a usar el convertidor `cruento` a través del método de un xlwings distancia objeto como lo hicimos en [Capítulo 9](#). De acuerdo con lo que vimos en ese entonces, obtendrá la mayor velocidad durante las operaciones de escritura. Por ejemplo, llamar a la siguiente función sin el decorador de retorno sería aproximadamente tres veces más lento en mi computadora portátil:

```
importar numpy como notario público
importar xlwings como xw
```

```

@ xw.func
@ xw.ret("crudo")
def randn(I=1000, j=1000):
    """ Devuelve una matriz con dimensiones (i, j) con números pseudoaleatorios
    distribuidos normalmente proporcionados por random.randn de NumPy
    """
    regreso notario público.aleatorio.randn(I, j)

```

Encontrará la muestra respectiva en el repositorio complementario en el *raw\_values* carpeta dentro de la *udfs* directorio. Cuando trabaja con UDF, tiene otra opción fácil para mejorar el rendimiento: evitar cálculos repetidos de funciones lentas almacenando en caché sus resultados.

#### Almacenamiento en caché

Cuando llamas a un *determinista* función, es decir, una función que dadas las mismas entradas, siempre devuelve la misma salida, puede almacenar el resultado en un *cache*: las llamadas repetidas de la función ya no tienen que esperar por el cálculo lento, pero pueden tomar el resultado de la caché donde ya está precalculado. Esto se explica mejor con un breve ejemplo. Un mecanismo de almacenamiento en caché muy básico se puede programar con un diccionario:

En [7]: **importar tiempo**

En [8]: **cache = {}**

```

def suma_lenta(a, B):
    llave = (a, B)
    si llave en cache:
        regreso cache[llave]
    demás:
        tiempo.dormir(2) # dormir por 2 segundos
        resultado = a + B
        cache[llave] = resultado
    regreso resultado

```

Cuando llama a esta función por primera vez, el cache está vacío. Por lo tanto, el código ejecutarádemás cláusula con la pausa artificial de dos segundos que imita un cálculo lento. Después de realizar el cálculo, agregará el resultado al cache dictionario antes de devolver el resultado. Cuando llame a esta función por segunda vez con los mismos argumentos y durante la misma sesión de Python, la encontrará en el cache y devuélvalo de inmediato, sin tener que volver a realizar el cálculo lento. El almacenamiento en caché de un resultado basado en sus argumentos también se llama *memorización*. En consecuencia, verá la diferencia horaria cuando llame a la función por primera y segunda vez:

En [9]: **%%tiempo**  
**suma\_lenta(1, 2)**

Tiempo de pared: 2,01 s

Salida [9]: 3

```
En [10]: %%tiempo
          suma_lenta(1, 2)
```

Tiempo de pared: 0 ns

Fuera [10]: 3

Python tiene un decorador incorporado llamado `lru_cache` que puede hacer tu vida realmente fácil y que importas desde el `functools` módulo que forma parte de la biblioteca estándar. `lru` representa *menos usado recientemente* caché y significa que contiene un número máximo de resultados (por defecto 128) antes de deshacerse de los más antiguos. Podemos usar esto con nuestro ejemplo de Tendencias de Google de la última sección. Siempre que solo estemos consultando valores históricos, podemos almacenar en caché el resultado de forma segura. Esto no solo hará que varias llamadas sean más rápidas, sino que también disminuirá la cantidad de solicitudes que enviamos a Google, lo que reducirá la posibilidad de que Google nos bloquee, algo que podría suceder si envía demasiadas solicitudes en poco tiempo.

Aquí están las primeras líneas del `get_interest_over_time` función con los cambios necesarios para aplicar el almacenamiento en caché:

```
de functools importar lru_cache ①

importar pandas como pd
de pytrends.request importar TrendReq
importar matplotlib.pyplot como pltimportar
xlwings como xw

@lru_caché ②
@ xw.func(call_in_wizard=False)
@ xw.arg("medios", xw.Distancia, Doc="ID de máquina: un rango de 5 celdas como máximo")@ ③
xw.arg("fecha de inicio", Doc="Una celda con formato de fecha")@ xw.arg("fecha final", Doc=
"Una celda con formato de fecha")def get_interest_over_time(medios, fecha de inicio, fecha
final):
    """ Consultar Tendencias de Google: reemplaza el ID de máquina (medio) de
    los lenguajes de programación comunes con su equivalente legible por
    humanos en el valor de retorno, por ejemplo, en lugar de"/m/05z1_",
    devuelve" Python ".
    """
    medios = medios.valor ④
```

- ① Importar el `lru_cache` decorador.
- ② Usa el decorador. El decorador tiene que estar encima del `xw.func` decorador.
- ③ Por defecto, `medios` es una lista. Esto crea un problema en este caso, ya que las funciones con listas como argumentos no se pueden almacenar en caché. El problema subyacente es que las listas son objetos mutables que no se pueden usar como claves en los diccionarios; ver [Apéndice C](#) para obtener más información sobre objetos mutables frente a inmutables. Utilizando el `xw.Range` convertidor

nos permite recuperar medios como xlwings distancia objeto en lugar de como lista, lo que resuelve nuestro problema.

- ④ Para que el resto del código vuelva a funcionar, ahora necesitamos obtener los valores a través del valor propiedad de los xlwings distancia objeto.



#### Almacenamiento en caché con diferentes versiones de Python

Si está usando una versión de Python por debajo de 3.8, tendrá que usar el decorador con paréntesis así: `@lru_caché ()`. Si está utilizando Python 3.9 o posterior, reemplace `@lru_cache` con `@cache`, que es lo mismo que `@lru_cache (maxsize = Ninguno)`, es decir, la caché nunca se deshace de los valores más antiguos. También necesita importar el `cache` decorador de `functools`.

los `xw.Range` El convertidor también puede ser útil en otras circunstancias, por ejemplo, si necesita acceder a las fórmulas de las celdas en lugar de a los valores en su UDF. En el ejemplo anterior, podría escribir `mids.formula` para acceder a las fórmulas de las celdas. Encontrarás el ejemplo completo en la `elgoogle_trends_cache` carpeta dentro de la `udfs` directorio del repositorio complementario.

Ahora que sabe cómo modificar el rendimiento de las UDF, terminemos esta sección presentando la `xw.sub` decorador.

### El decorador secundario

En [Capítulo 10](#), Te mostré cómo acelerar el `RunPython` llamar activando la configuración Usar servidor UDF. Si vive en un mundo solo de Windows, existe una alternativa a la `RunPython` / Usar servidor UDF combinación en forma de `xw.sub` decorador. Esto le permitirá importar sus funciones de Python como subprocedimientos en Excel, sin tener que escribir manualmente ninguna `RunPython` llamadas. En Excel, necesitará un procedimiento Sub para poder adjuntarlo a un botón, una función de Excel, como la obtiene al usar el `xw.func` decorador, no funcionará.

Para probar esto, cree un nuevo Inicio rápido proyecto llamado `importacionesub`. Como de costumbre, asegúrese de CD primero en el directorio donde desea que se cree el proyecto:

```
(base)> Importacionesub de inicio rápido de xlwings
```

En el Explorador de archivos, navegue hasta el `importacionesub` carpeta y abrir `importacionesub.xlsxm` en Excel y `importacionesub.py` en VS Code, luego decora el principal función con `@xw.sub` como se muestra en [Ejemplo 12-6](#).

*Ejemplo 12-6. importacionesub.py (extracto)*

```
importar xlwings como xw

@ xw.sub
def principal():
    wb = xw.Libro.llamador
    ()hoja = wb.hojas[0]
    si hoja["A1"].valor == "¡Hola xlwings!":
        hoja["A1"].valor = "¡Adiós xlwings!"
    demás:
        hoja["A1"].valor = "¡Hola xlwings!"
```

En el complemento xlwings, haga clic en Importar funciones antes de presionar Alt + F8 para ver las macros disponibles: además de SampleCall que usa RunPython, ahora también verá una macro llamada principal. Selecciónelo y haga clic en el botón Ejecutar; verá el saludo familiar en la celda A1. Ahora puede seguir adelante y asignar elprincipal macro a un botón como hicimos en [Capítulo 10](#). Mientras que laxw.sub Decorator puede hacer tu vida más fácil en Windows, ten en cuenta que al usarlo, pierdes la compatibilidad multiplataforma. Con xw.sub, hemos conocido a todos los decoradores de xlwings; los he resumido de nuevo en[Tabla 12-1](#).

*Tabla 12-1. decoradores xlwings*

Decorador	Descripción
xw.func	Coloque este decorador encima de todas las funciones que deseé importar a Excel como una función de Excel.
xw.sub	Coloque este decorador encima de todas las funciones que deseé importar a Excel como un procedimiento Sub de Excel.
xw.arg	Aplique convertidores y opciones a los argumentos, por ejemplo, agregue una cadena de documentos a través del Doc argumento o puede hacer que un rango llegue como DataFrame proporcionando pd.DataFrame como primer argumento (esto supone que ha importado pandas como pd).
xw.ret	Aplique convertidores y opciones para devolver valores, por ejemplo, suprima el índice de un DataFrame proporcionando índice = Falso.

Para obtener más detalles sobre estos decoradores, eche un vistazo a la [documentación xlwings](#).

## Conclusión

Este capítulo trata sobre cómo escribir funciones de Python e importarlas a Excel como UDF, lo que le permite llamarlas a través de fórmulas de celda. Al trabajar con el estudio de caso de Google Trends, aprendió cómo influir en el comportamiento de los argumentos de la función y los valores de retorno mediante el uso de arg y retirado decoradores, respectivamente. La última parte le mostró algunos trucos de rendimiento e introdujo el xw.sub decorador, que puedes usar como RunPython reemplazo si trabaja exclusivamente con Windows. Lo bueno de escribir UDF en Python es que esto le permite reemplazar fórmulas de celdas largas y complejas con código Python que será más fácil de

comprender y mantener. Mi forma preferida de trabajar con UDF es definitivamente usar pandas DataFrames con las nuevas matrices dinámicas de Excel, una combinación que facilita el trabajo con el tipo de datos que obtenemos de Google Trends, es decir, DataFrames con un número dinámico de filas.

Y eso es todo, ¡hemos llegado al final del libro! ¡Muchas gracias por su interés en mi interpretación de un entorno moderno de automatización y análisis de datos para Excel! Mi idea era presentarle el mundo de Python y sus poderosos paquetes de código abierto, permitiéndole escribir código Python para su próximo proyecto en lugar de tener que lidiar con las propias soluciones de Excel como VBA o Power Query, manteniendo así una puerta abierta para aléjese de Excel si es necesario. Espero poder darte algunos ejemplos prácticos para facilitar el inicio. Después de leer este libro, ahora sabe cómo:

- Reemplazar un libro de trabajo de Excel con un cuaderno de Jupyter y código pandas
- Procese por lotes libros de trabajo de Excel leyéndolos con OpenPyXL, xlrd, pyxlsb o xlwings y luego consolidelos con pandas
- Produzca informes de Excel con OpenPyXL, XlsxWriter, xlwt o xlwings
- Use Excel como interfaz y conéctelo a prácticamente cualquier cosa que desee a través de xlwings, ya sea haciendo clic en un botón o escribiendo una UDF

Sin embargo, pronto querrá ir más allá del alcance de este libro. Te invito a que revises el[página de inicio del libro](#) de vez en cuando para obtener actualizaciones y material adicional. Con este espíritu, aquí hay algunas ideas que podría explorar por su cuenta:

- Programe la ejecución periódica de una secuencia de comandos de Python utilizando el Programador de tareas en Windows o un trabajo cron en macOS o Linux. Por ejemplo, podría crear un informe de Excel todos los viernes en función de los datos que consume de una API REST o una base de datos.
- Escriba una secuencia de comandos de Python que envíe alertas por correo electrónico siempre que los valores en sus archivos de Excel satisfagan una determinada condición. Tal vez sea entonces cuando el saldo de su cuenta, consolidado a partir de varios libros de trabajo, cae por debajo de cierto valor, o cuando muestra un valor diferente al esperado según su base de datos interna.
- Escriba código que encuentre errores en los libros de Excel: busque errores de celda como #¡ÁRBITRO! o #¡VALOR! o errores lógicos como asegurarse de que una fórmula incluya todas las celdas que debería. Si comienza a rastrear sus libros de trabajo de misión crítica con un sistema de control de versiones profesional como Git, incluso puede ejecutar estas pruebas automáticamente cada vez que confirma una nueva versión.

Si este libro lo inspira a automatizar su rutina diaria o semanal de descargar datos y copiarlos / pegarlos en Excel, no podría estar más feliz. La automatización no solo le devuelve el tiempo, también reduce drásticamente la posibilidad de cometer errores. Si

Si tiene algún comentario, ¡hágamelo saber! Puede ponerse en contacto conmigo a través de O'Reilly, abriendo un número en el[repositorio complementario](#) o en Twitter en [@felixzumstein](#).



## APÉNDICE A

# Ambientes Conda

En [Capítulo 2](#), Presenté los entornos de Conda explicando que (base) al comienzo de un Anaconda Prompt significa el entorno Conda actualmente activo con el nombre base. Anaconda requiere que trabajes siempre en un entorno activado, pero la activación se realiza automáticamente para el base entorno cuando inicia Anaconda Prompt en Windows o la Terminal en macOS. Trabajar con entornos Conda le permite separar adecuadamente las dependencias de sus proyectos: si desea probar una versión más nueva de un paquete como pandas sin cambiar subbase entorno, puede hacerlo en un entorno Conda separado. En la primera parte de este apéndice, lo guiaré a través del proceso de creación de un entorno Conda llamado xl38 donde instalaremos todos los paquetes en la versión que los usé para escribir este libro. Esto le permitirá ejecutar las muestras de este libro tal como están, incluso si algunos paquetes han lanzado nuevas versiones con cambios importantes mientras tanto. En la segunda parte, le mostraré cómo deshabilitar la activación automática delbase entorno si no le gusta el comportamiento predeterminado.

### Cree un nuevo entorno de Conda

Ejecute el siguiente comando en su Anaconda Prompt para crear un nuevo entorno con el nombre xl38 que usa Python 3.8:

```
(base)> conda crear --nombre xl38 python = 3.8
```

Al presionar Enter, Conda imprimirá lo que va a instalar en el nuevo entorno y le pedirá que confirme:

Continuar ([y] / n)?

Presione Enter para confirmar o escribir **norte** si quieras cancelar. Una vez realizada la instalación, active su nuevo entorno así:

```
(base)> conda activar xl38  
(xl38)>
```

El nombre del entorno cambió de base para xl38 y ahora puede usar Conda o pip para instalar paquetes en este nuevo entorno sin afectar a ninguno de los otros entornos (como recordatorio: solo use pip si el paquete no está disponible a través de Conda). Sigamos e instalaremos todos los paquetes de este libro en la versión que los estaba usando. Primero, verifique que esté en el xl38 entorno, es decir, Anaconda Prompt muestra (xl38), luego instale los paquetes de Conda así (el siguiente comando debe escribirse como un solo comando; los saltos de línea son solo para fines de visualización):

```
(xl38)> conda install lxml = 4.6.1 matplotlib = 3.3.2 cuaderno = 6.1.4 openpyxl = 3.0.5  
          pandas = 1.1.3 almohada = 8.0.1 plotly = 4.14.1 flake8 = 3.8.4  
          python-dateutil = 2.8.1 solicitudes = 2.24.0 sqlalchemy = 1.3.20  
          xlrd = 1.2.0 xlsxwriter = 1.3.7 xlutils = 2.0 .0 xlwings = 0.20.8 xlwt  
          = 1.3.0
```

Confirme el plan de instalación y finalice el entorno instalando las dos dependencias restantes con pip:

```
(xl38)> pip install pyxlsb == 1.0.7 pytrends == 4.7.3  
(xl38)>
```



### Cómo utilizar el entorno xl38

Si desea utilizar el xl38 medio ambiente en lugar del baseentorno para trabajar con los ejemplos de este libro, asegúrese de tener siempre su xl38 entorno activado al ejecutar:

```
(base)> conda activar xl38
```

Es decir, siempre que muestre el mensaje Anaconda como (base)>, querrás que se muestre xl38> en lugar de.

Para volver a desactivar el entorno y volver a la base medio ambiente, tipo:

```
(xl38)> conda desactivar  
(base)>
```

Si desea eliminar el entorno por completo, ejecute el siguiente comando:

```
(base)> conda env remove --name xl38
```

En lugar de seguir los pasos manualmente para crear el xl38 entorno, también puede aprovechar el archivo de entorno *xl38.ym* que incluí en el *condacarpeta* del repositorio complementario. Ejecutar los siguientes comandos se encarga de todo:

```
(base)> cd C:\ Usuarios \nombre de usuario\ python-para-excel  
\ conda(base)> conda env create -f xl38.yml(base)> conda  
activar xl38(xl38)>
```

Por defecto, Anaconda siempre activa el base entorno cuando abre un terminal en macOS o Anaconda Prompt en Windows. Si no le gusta esto, puede desactivar la activación automática como le mostraré a continuación.

#### Deshabilitar la activación automática

Si no quieras el base entorno para que se active automáticamente cada vez que inicie un Anaconda Prompt, puede deshabilitarlo: esto requerirá que escriba **conda activar base** manualmente en un símbolo del sistema (Windows) o una terminal (macOS) antes de poder usar Python.

##### Ventanas

En Windows, deberá usar el símbolo del sistema normal en lugar del símbolo de Anaconda. Los siguientes pasos permitirán ejecutar el comando en un símbolo del sistema normal. Asegúrese de reemplazar la ruta en la primera línea con la ruta donde Anaconda está instalado en su sistema:

```
> cd C:\ Usuarios \nombre de usuario\ Anaconda3\ condabin  
> conda init cmd.exe
```

Su símbolo del sistema regular ahora está configurado con Conda, por lo que en el futuro puede activar el base entorno como este:

```
> conda activar base  
(base)>
```

##### Mac OS

En macOS, simplemente ejecute el siguiente comando en su Terminal para deshabilitar la activación automática:

```
(base)> conda config --set auto_activate_base false
```

Si alguna vez desea revertir, ejecute el mismo comando nuevamente con cierto en lugar de falso. Los cambios entrarán en vigor después de reiniciar la Terminal. De cara al futuro, deberá activar el base entorno como este antes de que pueda utilizar el siguiente comando de nuevo:

```
> conda activar base  
(base)>
```



## Funcionalidad avanzada del código VS

Este apéndice le muestra cómo funciona el depurador en VS Code y cómo puede ejecutar los cuadernos de Jupyter directamente desde VS Code. Los temas son independientes entre sí, por lo que puede leerlos en cualquier orden.

### Depurador

Si alguna vez usó el depurador de VBA en Excel, tengo buenas noticias para usted: depurar con VS Code es una experiencia muy similar. Empecemos abriendo el archivo `debugging.py` desde el repositorio complementario en VS Code. Haga clic en el margen a la izquierda de la línea número 4 para que aparezca un punto rojo; este es su punto de interrupción donde se pausará la ejecución del código. Ahora presione F5 para comenzar a depurar: aparecerá el Panel de comando con una selección de configuraciones de depuración. Elija "Archivo Python" para depurar el archivo activo y ejecutar el código hasta que llegue al punto de interrupción. La línea se resaltará y la ejecución del código se pausará, consulte [Figura B-1](#). Mientras depura, la barra de estado se vuelve naranja.

Si la sección Variables no se muestra automáticamente a la izquierda, asegúrese de hacer clic en el menú Ejecutar para ver los valores de las variables. Alternativamente, también puede colocar el cursor sobre una variable en el código fuente y obtener una información sobre herramientas con su valor. En la parte superior, verá la barra de herramientas de depuración que le da acceso a los siguientes botones de izquierda a derecha): Continuar, Pasar por alto, Pasar a, Salir, Reiniciar y Detener. Cuando pase el cursor sobre ellos, también verá los atajos de teclado.

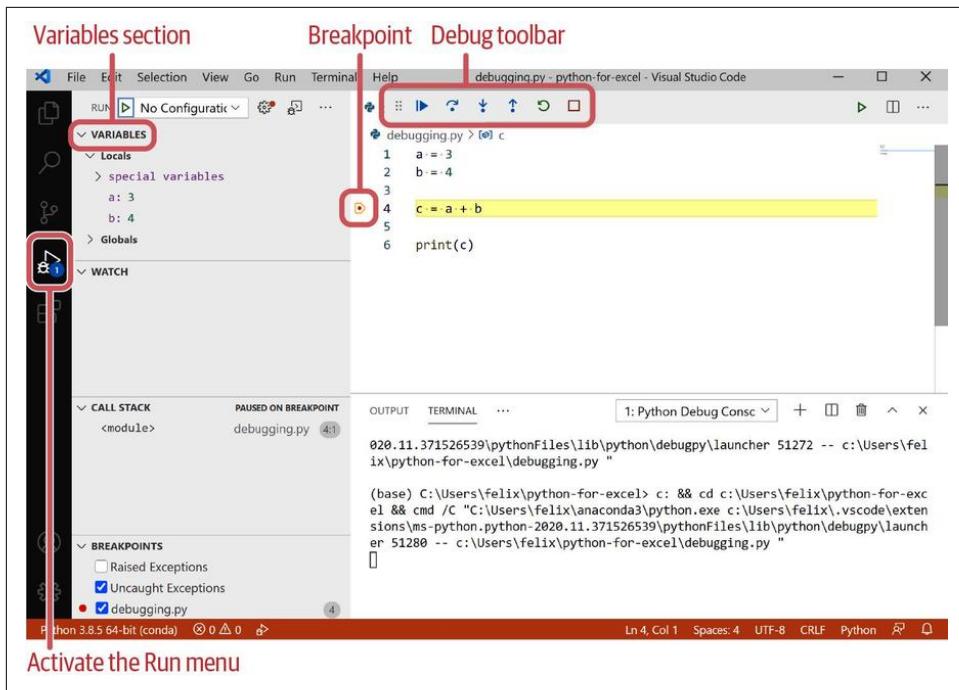


Figura B-1. VS Code con el depurador detenido en el punto de interrupción

Veamos qué hace cada uno de estos botones:

#### *Continuar*

Esto continúa ejecutando el programa hasta que llega al siguiente punto de interrupción o al final del programa. Si llega al final del programa, el proceso de depuración se detendrá.

#### *Paso sobre*

El depurador avanzará una línea. *Paso sobre* significa que el depurador no recorrerá visualmente las líneas de código que no forman parte de su alcance actual. Por ejemplo, no entrará en el código de una función a la que llame línea por línea, ¡pero la función seguirá llamándose!

#### *Entrar en*

Si tiene un código que llama a una función o clase, etc., *Entrar en* hará que el depurador entre en esa función o clase. Si la función o clase está en un archivo diferente, el depurador abrirá este archivo por usted.

### *Salir*

Si ingresó a una función con Step Into, *Salir* hace que el depurador vuelva al siguiente nivel superior hasta que, finalmente, volverá al nivel más alto desde donde llamó a Step Into inicialmente.

### *Reiniciar*

Esto detendrá el proceso de depuración actual y comenzará uno nuevo desde el principio.

### *Parada*

Esto detendrá el proceso de depuración actual.

Ahora que sabe lo que hace cada botón, haga clic en Pasar por alto para avanzar una línea y ver cómo variable C aparece en la sección Variables, luego finalice este ejercicio de depuración haciendo clic en Continuar.

Si guarda la configuración de depuración, el Panel de comando no aparecerá y le preguntará acerca de la configuración cada vez que presione F5: haga clic en el ícono Ejecutar en la barra de actividad, luego haga clic en "crear un archivo launch.json". Esto hará que el Panel de comandos se muestre nuevamente y cuando seleccione "Archivo Python", crea el `launch.json` archivo en un directorio llamado `.vscode`. Cuando presione F5, el depurador se iniciará de inmediato. Si necesita cambiar la configuración o desea volver a abrir la ventana emergente del Panel de comando, edite o elimine el `launch.json` archivo en el `.vscode` directorio.

## Cuadernos Jupyter en VS Code

En lugar de ejecutar sus cuadernos de Jupyter en un navegador web, también puede ejecutarlos directamente con VS Code. Además de eso, VS Code ofrece un conveniente explorador de variables, así como opciones para transformar el portátil en archivos estándar de Python sin perder la funcionalidad de la celda. Esto facilita el uso del depurador o copiar / pegar celdas entre diferentes cuadernos. ¡Comencemos ejecutando un cuaderno en VS Code!

### Ejecutar cuadernos de Jupyter

Haga clic en el ícono del Explorador en la barra de actividades y abra `ch05.ipynb` desde el repositorio complementario. Para continuar, deberá hacer que el cuaderno sea de confianza haciendo clic en Confiar en la notificación que aparece. El diseño del portátil se ve un poco diferente al del navegador para que coincida con el resto de VS Code, pero por lo demás, es la misma experiencia, incluidos todos los atajos de teclado. Ejecutemos las primeras tres celdas a través de Shift + Enter. Esto iniciará el servidor de la computadora portátil Jupyter si aún no se está ejecutando (verá el estado en la parte superior derecha de la computadora portátil). Después de ejecutar las celdas, haga clic en el botón de la calculadora en el menú en la parte superior del cuaderno: esto abrirá el Explorador de variables, en el que puede ver los valores de todas las variables que existen actualmente, como en Figura B-2. Es decir, solo encontrará variables de celdas que se hayan ejecutado.



#### Guardar los cuadernos de Jupyter en VS Code

Para guardar cuadernos en VS Code, debe usar el botón Guardar en la parte superior del cuaderno o presionar Ctrl + S en Windows o Command-S en macOS. Archivo > Guardar no funcionará.

The screenshot shows the Jupyter Notebook interface within VS Code. At the top, there's a toolbar with 'Save notebook', 'Variable explorer' (highlighted in red), 'Export as', and 'Jupyter server status' (highlighted in red). Below the toolbar, the 'Variables' section lists two items: 'data' (list, size 4, value: [[Mark, 55, 'Italy', 4.5, 'Europe']]) and 'df' (DataFrame, size (4, 5), columns: name age country score continent, value: 1001). A code cell at the bottom contains the following Python code:

```
[1] import pandas as pd  
import numpy as np
```

Figura B-2. Explorador de variables del cuaderno Jupyter

Si usa estructuras de datos como listas anidadas, matrices NumPy o DataFrames, puede hacer doble clic en la variable: esto abrirá el Visor de datos y le dará una vista familiar similar a una hoja de cálculo. [Figura B-3](#) muestra el Visor de datos después de hacer doble clic en el df variable.

The screenshot shows the 'Data Viewer - df' window. At the top, there's a 'Filter Rows' button. Below it is a table with the following data:

index	name	age	country	score	contin...
1001	Mark	55	Italy	4.5	Europe
1000	John	33	USA	6.7	America
1002	Tim	41	USA	3.9	America
1003	Jenny	12	Germany	9	Europe

Figura B-3. Visor de datos del cuaderno Jupyter

Si bien VS Code le permite ejecutar archivos de cuaderno estándar de Jupyter, también le permite transformar los cuadernos en archivos normales de Python, sin perder sus celdas. ¡Vamos a ver cómo funciona!

### Secuencias de comandos de Python con celdas de código

Para usar celdas de cuaderno de Jupyter en archivos estándar de Python, VS Code usa un comentario especial para denotar celdas: `# %%`. Para convertir un cuaderno Jupyter existente, ábralo y presione el botón Exportar como en el menú en la parte superior del cuaderno; ver Figura B-2. Esto le permitirá seleccionar "Archivo Python" de la paleta de comandos. Sin embargo, en lugar de convertir un archivo existente, creamos un nuevo archivo que llamamos `cell.py` con el siguiente contenido:

```
# %%
3 + 4
# %% [rebaja]
## Este es un título
#
# Contenido de rebajas
```

Las celdas de Markdown deben comenzar con `# %% [reducción]` y requieren que toda la celda se marque como comentario. Si desea ejecutar un archivo como un cuaderno, haga clic en el enlace "Ejecutar a continuación" que aparece cuando se desplaza sobre la primera celda. Esto abrirá el *Ventana interactiva de Python* a la derecha, como se muestra en Figura B-4.

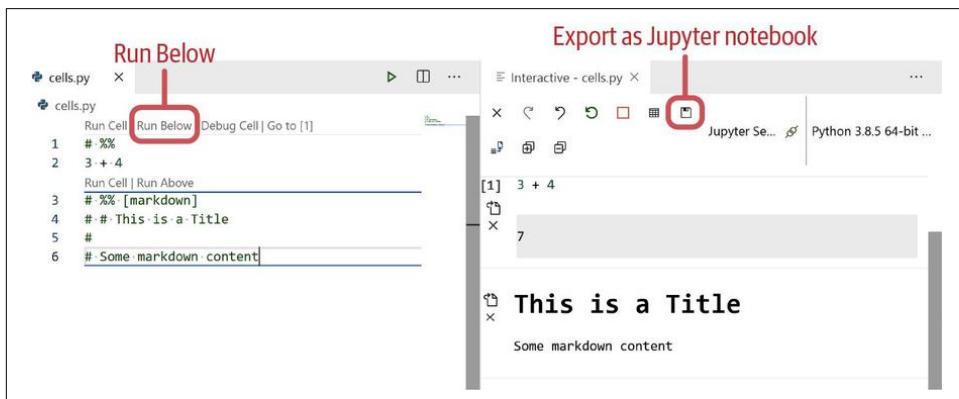


Figura B-4. Ventana interactiva de Python

La ventana interactiva de Python se muestra nuevamente como cuaderno. Para exportar su archivo en el `ipynb` formato, haga clic en el ícono Guardar (Exportar como cuaderno de Jupyter) en la parte superior de la ventana interactiva de Python. La ventana interactiva de Python también le ofrece una celda en la parte inferior desde donde puede ejecutar código de forma interactiva. El uso de archivos Python normales en lugar de los cuadernos de Jupyter le permite usar el depurador de VS Code y facilita el trabajo con el control de versiones, ya que se ignoran las celdas de salida, que generalmente agregan mucho ruido entre versiones.



## Conceptos avanzados de Python

En este apéndice, examinaremos más de cerca los siguientes tres temas: clases y objetos, objetos de fecha y hora que reconocen la zona horaria y objetos mutables frente a inmutables. Los temas son independientes entre sí, por lo que puede leerlos en cualquier orden.

### Clases y objetos

En esta sección, escribiremos nuestra propia clase para comprender mejor cómo se relacionan las clases y los objetos. Las clases definen nuevos tipos de objetos: una clase se comporta como un springform que usas para hornear un pastel. Dependiendo de los ingredientes que uses, obtienes un pastel diferente, por ejemplo, un pastel de chocolate o un pastel de queso. El proceso de sacar un pastel (el objeto) del springform (la clase) se llama *instanciación*, razón por la cual los objetos también se denominan *instancias de clase*. Ya sea chocolate o tarta de queso, ambos son un *escribe of cake*: las clases le permiten definir nuevos tipos de datos que mantienen los datos relacionados (*atributos*) y funciones (*métodos*) juntos y, por lo tanto, le ayudarán a estructurar y organizar su código. Permítanme volver ahora al ejemplo del juego de carreras de coches de [Capítulo 3](#) para definir nuestra propia clase:

En [1]: clase Coche:

```
def __en eso__(uno mismo, color, velocidad=0):
    uno mismo.color = coloruno
    mismo.velocidad = velocidad

def acelerar(uno mismo, mph):
    uno mismo.velocidad += mph
```

Esta es una clase de automóvil simple con dos métodos. Los métodos son funciones que forman parte de una definición de clase. Esta clase tiene un método regular llamado *acelerar*. Este método cambiará los datos (*velocidad*) de una instancia de esta clase. También tiene un método especial que comienza y termina con guiones bajos dobles llamados *\_en eso\_*. Python lo llamará automáticamente cuando un objeto sea *inicializado* para adjuntar algunos datos iniciales al

objeto. El primer argumento de cada método representa la instancia de la clase y se llama uno mismo por convención. Esto se aclarará cuando vea cómo usa elCoche clase. Primero, creamos dos autos. Está haciendo esto de la misma manera que está llamando a una función: llame a la clase agregando paréntesis y proporcionando los argumentos de \_\_en\_ eso\_\_ método. Nunca proporcionas nada para uno mismo, ya que Python se encargará de eso. En esta muestra, uno mismo estarán coche1 o coche2, respectivamente:

```
En [2]: # Creamos una instancia de dos objetos de automóvil  
coche1 = Coche("rojo")coche2 =  
Coche(color="azul")
```

Cuando llamas a una clase, realmente estás llamando al \_\_en\_ eso\_\_ función, razón por la cual todo lo relacionado con los argumentos de función se aplica aquí también: para coche1, proporcionamos el argumento como argumento posicional, mientras que para coche2, estamos usando argumentos de palabras clave. Después de instanciar los dos objetos de coche de laCoche class, echaremos un vistazo a sus atributos y llamaremos a sus métodos. Como veremos, tras acelerar coche1, la velocidad de coche1 ha cambiado, pero no ha cambiado para coche2 ya que los dos objetos son independientes el uno del otro:

```
En [3]: # De forma predeterminada, un objeto imprime su ubicación de memoria  
coche1
```

Fuera [3]: <\_\_ main\_\_.Car at 0x7fea812e3890>

```
En [4]: # Los atributos le dan acceso a los datos de un objeto  
coche1.color
```

Fuera [4]: 'rojo'

```
En [5]: coche1.velocidad
```

Fuera [5]: 0

```
En [6]: # Llamar al método de aceleración en car1  
coche1.acelerar(20)
```

```
En [7]: # El atributo de velocidad de car1 cambió  
coche1.velocidad
```

Fuera [7]: 20

```
En [8]: # El atributo de velocidad de car2 se mantuvo igual  
coche2.velocidad
```

Fuera [8]: 0

Python también le permite cambiar atributos directamente sin tener que usar métodos:

```
En [9]: coche1.color = "verde"En
```

```
[10]: coche1.colorFuera [10]:
```

'verde'

```
En [11]: coche2.color # sin alterar
```

Fuera [11]: 'azul'

Para resumir: las clases definen los atributos y métodos de los objetos. Las clases le permiten agrupar funciones relacionadas ("métodos") y datos ("atributos") juntos para que se pueda acceder a ellos cómodamente mediante la notación de puntos:myobject.attribute o myobject.method ().

## Trabajar con objetos de fecha y hora que reconocen la zona horaria

En Capítulo 3, analizamos brevemente la zona horaria ingenua fecha y hora objetos. Si la zona horaria es importante, normalmente trabaja en el *UTC* zona horaria y solo se transforma en zonas horarias locales para fines de visualización. UTC significa *Hora universal coordinada* y es el sucesor de Greenwich Mean Time (GMT). Cuando trabaja con Excel y Python, es posible que desee convertir las marcas de tiempo ingenuas, como las entrega Excel, en una zona horaria consciente fecha y hora objetos. Para la compatibilidad con la zona horaria en Python, puede usar el paquete dateutil, que no forma parte de la biblioteca estándar pero viene preinstalado con Anaconda. Los siguientes ejemplos muestran algunas operaciones comunes cuando se trabaja con fecha y hora objetos y zonas horarias:

En [12]: `importar fecha y hora como dt  
de dateutil importar tz`

En [13]: `# Objeto de fecha y hora ingenuo de zona horaria  
marca de tiempo = dt.fecha y hora(2020, 1, 31, 14, 30)marca  
de tiempo.isoformat()`

Fuera [13]: '2020-01-31T14: 30: 00'

En [14]: `# Objeto de fecha y hora con reconocimiento de zona horaria  
timestamp_eastern = dt.fecha y hora(2020, 1, 31, 14, 30,  
tzinfo=tz.gettz("EE.UU. / Este"))  
# La impresión en isoformato facilita  
# ver el desplazamiento de UTC  
timestamp_eastern.isoformat()`

Fuera [14]: '2020-01-31T14: 30: 00-05: 00'

En [15]: `# Asignar una zona horaria a un objeto de fecha y hora ingenuo  
timestamp_eastern = marca de tiempo.reemplazar(tzinfo=tz.gettz("EE.UU. / Este"))  
timestamp_eastern.isoformat()`

Fuera [15]: '2020-01-31T14: 30: 00-05: 00'

En [dieciséis]: `# Convierta de una zona horaria a otra.  
# Dado que la zona horaria UTC es tan común,  
# hay un atajo: tz.UTC  
timestamp_utc = timestamp_eastern.astimezone(tz.UTC)  
timestamp_utc.isoformat()`

Salida [16]: '2020-01-31T19: 30: 00 +00: 00'

En [17]: `# De consciente de la zona horaria a ingenuo  
timestamp_eastern.reemplazar(tzinfo=Ninguno)`

Fuera [17]: `datetime.datetime(2020, 1, 31, 14, 30)`

En [18]: `# Hora actual sin zona horaria  
dt.fecha y hora.ahora()`

Fuera [18]: `datetime.datetime(2021, 1, 3, 11, 18, 37, 172170)`

En [19]: `# Hora actual en la zona horaria UTC  
dt.fecha y hora.ahora(tz.UTC)`

Fuera [19]: `datetime.datetime(2021, 1, 3, 10, 18, 37, 176299, tzinfo = tzutc ())`

### Zonas horarias con Python 3.9

Python 3.9 agregó soporte de zona horaria adecuado a la biblioteca estándar en forma de zona horaria módulo. Úsalo para reemplazar `eltz.gettz` llamadas de `dateutil`:

```
de zoneinfo importar ZoneInfo timestamp_eastern = dt.  
fecha y hora(2020, 1, 31, 14, 30,  
tzinfo=ZoneInfo("EE.UU. / Este"))
```

## Objetos de Python mutables frente a inmutables

En Python, los objetos que pueden cambiar sus valores se llaman *mutable* y los que no pueden se llaman *inmutable*. Cuadro C-1 muestra cómo califican los diferentes tipos de datos.

Cuadro C-1. Tipos de datos mutables e inmutables

Mutabilidad	Tipos de datos
mutable	listas, diccionarios, conjuntos
inmutable	enteros, flotantes, valores booleanos, cadenas, fecha y hora, tuplas

Conocer la diferencia es importante ya que los objetos mutables pueden comportarse de manera diferente a lo que está acostumbrado en otros lenguajes, incluido VBA. Eche un vistazo al siguiente fragmento de VBA:

```
Oscuro a Como Variante, B Como  
Variantea = Formación(1, 2, 3)B = a  
  
a(1) = 22  
Depurar.Imprimir un(0) Y "," Y a(1) Y "," Y a(2)  
Depurar.Imprimir b(0) Y "," Y B(1) Y "," Y B(2)
```

Esto imprime lo siguiente:

```
1, 22, 3  
1, 2, 3
```

Ahora hagamos el mismo ejemplo en Python con una lista:

```
En [20]: a = [1, 2, 3]
```

```
B = a
```

```
a[1] = 22
```

```
impresión(a)
```

```
impresión(B)
```

```
[1, 22, 3]
```

```
[1, 22, 3]
```

¿Qué pasó aquí? En Python, las variables son nombres que "adjunta" a un objeto. Haciendo `b = a`, adjunta ambos nombres al mismo objeto, la lista `[1, 2, 3]`. Todas las variables adjuntas a ese objeto, por lo tanto, mostrarán los cambios en la lista. Sin embargo, esto solo sucede con objetos mutables: si reemplaza la lista con un objeto inmutable como una tupla, cambiaria no cambiaria `B`. Si quieres un objeto mutable como `B` ser independiente de los cambios en `a`, tienes que copiar explícitamente la lista:

```
En [21]: a = [1, 2, 3]
```

```
B = a.Copiar()
```

```
En [22]: a
```

```
Fuera [22]: [1, 2, 3]
```

```
En [23]: B
```

```
Fuera [23]: [1, 2, 3]
```

```
En [24]: a[1] = 22 # Cambiando "a" ...En [
```

```
25]: a
```

```
Fuera [25]: [1, 22, 3]
```

```
En [26]: B # ...no afecta a "b" Fuera [26]:
```

```
[1, 2, 3]
```

Usando una lista `Copiar` método, estás creando un *copia superficial*: obtendrá una copia de la lista, pero si la lista contiene elementos mutables, estos aún se compartirán. Si desea copiar todos los elementos de forma recursiva, debe crear una *copia profunda* usando el `Copiar` módulo de la biblioteca estándar:

```
En [27]: importar Copiar
```

```
B = Copiar.copia profunda(a)
```

Veamos ahora qué sucede cuando usas objetos mutables como argumentos de función.

## Llamar a funciones con objetos mutables como argumentos

Si viene de VBA, probablemente esté acostumbrado a marcar argumentos de función como *paso por referencia* (`ByRef`) o *pasar por valor* (`ByVal`): cuando pasa una variable a una función como argumento, la función tendrá la capacidad de cambiarla (`ByRef`) o voluntad

trabajar en una copia de los valores (ByVal), dejando así intacta la variable original. ByRef es el predeterminado en VBA. Considere la siguiente función en VBA:

```
Función incremento(ByRef X Como Entero) Como Entero
```

```
    X = X + 1
```

```
    incremento = X
```

```
Función final
```

Luego, llame a la función así:

```
Sub call_increment()
```

```
    Oscuro a Como
```

```
    Enteroa = 1
```

```
    Depurar.Incremento de impresión(a)
```

```
    Depurar.Imprimir un
```

```
End Sub
```

Esto imprimirá lo siguiente:

```
2
```

```
2
```

Sin embargo, si cambia ByRef en el incremento función para ByVal, imprimirá:

```
2
```

```
1
```

¿Cómo funciona esto en Python? Cuando pasa variables, pasa nombres que apuntan a objetos. Esto significa que el comportamiento depende de si el objeto es mutable o no. Primero usemos un objeto inmutable:

```
En [28]: def incremento(X):
```

```
    X = X + 1
```

```
    regreso X
```

```
En [29]: a = 1
```

```
    impresión(incremento(a))
```

```
    impresión(a)
```

```
2
```

```
1
```

Ahora repitamos la muestra con un objeto mutable:

```
En [30]: def incremento(X):
```

```
    X[0] = X[0] + 1
```

```
    regreso X
```

```
En [31]: a = [1]
```

```
    impresión(incremento(a))
```

```
    impresión(a)
```

```
[2]
```

```
[2]
```

Si el objeto es mutable y desea dejar el objeto original sin cambios, deberá pasar una copia del objeto:

```
En [32]: a = [1]
impresión(incremento(a.Copiar()))
impresión(a)
```

```
[2]
[1]
```

Otro caso a tener en cuenta es el uso de objetos mutables como argumentos predeterminados en las definiciones de funciones. ¡Veamos por qué!

### Funciones con objetos mutables como argumentos predeterminados

Cuando escribe funciones, normalmente no debería utilizar objetos mutables como argumentos predeterminados. La razón es que el valor de los argumentos predeterminados se evalúa solo una vez como parte de la definición de la función, no cada vez que se llama a la función. Por lo tanto, el uso de objetos mutables como argumentos predeterminados puede provocar un comportamiento inesperado:

```
En [33]: # No hagas esto:
def Agrega uno(X=[]):
    X.adjuntar(1)
    regreso X
```

```
En [34]: Agrega uno()
```

```
Fuera [34]: [1]
```

```
En [35]: Agrega uno()
```

```
Fuera [35]: [1, 1]
```

Si desea utilizar una lista vacía como argumento predeterminado, haga lo siguiente:

```
En [36]: def Agrega uno(X=Ninguno):
    si X es Ninguno:
        X = []
    X.adjuntar(1)
    regreso X
```

```
En [37]: Agrega uno()
```

```
Fuera [37]: [1]
```

```
En [38]: Agrega uno()
```

```
Fuera [38]: [1]
```



---

# Índice

## Símbolos

%% tiempo de magia celular, 172%  
% timeit magia celular, 172

## A

caminos absolutos, 23, 247activando entornos Conda, 282Controles ActiveX, 17, 215Barra de actividad (código VS), 36 complementos (Excel)  
personalizado, 220, 257instalar xlwings, 210-211agregando columnas a DataFrame, 100 elementos a listas, 54 paquetes a Python Package Tracker, 223-225 función add\_package (paquete Python Tracker) ejemplo, 245-246 método agg (pandas), 113 función aggfunc (pandas), 113 alias para módulos, 68Altair, 119 Anaconda componentes de, 19 Conda (ver Conda) instalación 20-21 propósito de, 19 Aviso Anaconda comandos, lista de, 22sesión interactiva de Python finalizando, 25 a partir de, 24-25rutas de archivo largas, 39 notación, 24 descripción operativa, 21-24 propósito de, 19 Secuencias de comandos de Python, en ejecución, 39corriendo en macOS, 22 en Windows, 21 VS Code, en ejecución, 37 CLI de xlwings addin install comando, 210 propósito de, 210 comando de inicio rápido, 212 software antivirus, instalación de xlwings y, 210API (interfaces de programación de aplicaciones), 226 objeto de aplicación (xlwings), 186, 192, 207añadir método (OpenPyXL), 170AppleScript, 204 interfaces de programación de aplicaciones (API), 226 estructura de la aplicación (Python Package Tracker), 240 backend 245-248depuración 248-249 Interfaz, 241-245aplicaciones, capas de, 6-7método applymap (pandas), 105-107 función arange (NumPy), 83decoradores de argumentos (xlwings), 259, 277argumentos, objetos mutables como, 295-297operaciones aritméticas en DataFrames, 103-104en matrices NumPy, 79-80 operadores aritméticos (pandas), correspondientes métodos para, 104 rangos de matriz, 83

fórmulas basadas en matrices (xlwings), 272

matrices (NumPy)

radiodifusión, 79-80

constructores, 83

problemas de análisis de datos en, 84

tipos de datos, 78-79

obtener y configurar elementos, 82-83

descripción operativa, 77-79funciones

universales (ufuncs), 80-81

vectorización, 79-80vistas versus

copias, 83-84método asfreq (pandas),

137atributos

documentación de ayuda, 50

propósito de, 45, 291-293notación de asignación

aumentada, 63activación automática de entornos

Conda, desactivación

costoso, 283

autocompletear, 34

método de ajuste automático (xlwings), 203

automatización en Excel (ver xlwings)

## B

backends

propósito de, 241

en Python Package Tracker, 245-248

mejores prácticas para la programación

Principio SECO, 8

separación de intereses, 6-7

pruebas, 8

control de versiones, 9-11

big data, xi, 138, 169-173

Aglutinante, 33

Bokeh, 119

Clase de libro (xlwings), 188objeto

de libro (xlwings), 188colección de

libros (xlwings), 193constructor

bool, 48

tipo de datos booleanos, 47-49, 148

indexación booleana (pandas)

seleccionando datos por, 94-96

configuración de datos por, 98-

99operadores booleanos, 47, 95

declaración de ruptura, 62

puntos de interrupción (código VS),

configuración, 270radiodifusión, 79-80, 104

convertidores incorporados (xlwings), 196

opciones integradas para el objeto de rango (xlwings), 196

inteligencia empresarial (consulte Power BI)

capa empresarial, 6

Argumentos de la función ByRef (VBA), 295-296

Argumentos de la función ByVal (VBA), 295-296

## C

decorador de caché, 276

almacenamiento en caché 274-276cálculos,

capa separada para, 7funciones de

llamada, 44, 66, 295-297método de

capitalización (pandas), 105Hojas de estilo

en cascada (CSS), 177Declaración de caso

(VBA), 56estudios de caso

Informes de Excel, 143-147, 178, 202-203Estudio de

caso de Tendencias de Google

DataFrames y matrices dinámicas,

258-263

depurar UDF, 269-271explicación de Google

Trends, 257-258recuperación de datos, 263-

267trazar datos, 267-269Rastreador de

paquetes de Python

agregando paquetes, 223-225

estructura de la aplicación,

240backend 245-248bases de

datos, 229-237depuración 248

-249manejo de errores, 238-

240Interfaz, 241-245API web,

226-229comando cd, 23

formato de celda (xlwings), limpieza, 207-208

bucle celular, 158

células (cuadernos Jupyter)

modo de edición versus modo de comando, 31-32en

los scripts de Python, 289descripción operativa, 29-

31producción, 30

orden de ejecución, 32

encadenamiento de operaciones de indexación y segmentación, 52, 82

cambiando

tipos de células (cuadernos Jupyter), 30

directorios, 23

separadores de lista (Excel), 187-188

al directorio principal, 23gráficos

(Excel), 114

(ver también parcelas)

creando en OpenPyXL, 160-162

creando en XlsxWriter, 163-165

creando en xlwings, 198-199  
herencia de clases, 17  
clases 45  
  instanciación, 69, 291  
  objetos y, 291-293  
borrar formato de celda (xlwings), 207-208  
proveedores en la nube para portátiles Jupyter, 33  
bloques de código, 58-59  
celdas de código (cuadernos Jupyter), 30  
comando de código (VS Code), 37  
colecciones (xlwings)  
  gráficos (Excel), creación, 198-199  
  nombres definidos (Excel), creando, 200-202  
  imágenes (Excel), Matplotlib traza como, 199-200  
  propósito de, 186  
colores, valores hexadecimales para, 161-162  
columnas (pandas)  
  agregando a DataFrame, 100  
  para DataFrames, 90-91, 174-175  
  seleccionando, 93  
COM (modelo de objetos componentes), 204  
combinando DataFrames  
  concatenación, 108-109  
  unión, 109-111  
  fusionando 110-111  
historial de comandos, desplazándose, 23  
modo de comando (cuadernos Jupyter), 31-32  
Paleta de comandos (código VS), 37  
Símbolo del sistema (Windows), 21  
  depurando con, 249  
  deshabilitar la activación automática del entorno Conda ment, 283  
comandos en Anaconda Prompt, lista de, 22  
comentarios en Python, 47  
tipo de datos complejos, 46  
claves compuestas (bases de datos), 232  
función concat (pandas), 108-109, 134  
concatenando

DataFrames, 108-109  
liza, 53  
  instrumentos de cuerda, 49  
tuplas 57  
Conda, 25-27  
  comandos 26  
  entornos, 27  
    creando 281-283  
    deshabilitar la activación automática, 283  
    propósito de, 281  
compatibilidad multiplataforma, 17  
Matrices CSE (Excel), 260  
CSS (hojas de estilo en cascada), 177  
Archivos CSV  
  exportar datos de DataFrame como, 120-121  
  importar a DataFrames, 121-123  
cuDF, 138  
directorio actual

pip versus, 25  
expresiones condicionales, 60  
formato condicional en el paquete Python  
  Rastreador, 243  
jerarquía de configuración para xlwings, 220  
configurar VS Code, 36-37  
conexiones a bases de datos, 232-233  
memoria constante (XlsxWriter), 170  
constructores para matrices (NumPy), 83  
administradores de contexto, 150

Botón Continuar (depurador de código VS), 286  
Continuar declaración, 62  
flujo de control  
  bloques de código, 58-59  
  expresiones condicionales, 60  
  comprensiones de diccionario, 64  
  para bucles, 60-63  
  si declaraciones, 59-60  
  lista de comprensiones, 63  
  pasar declaración, 58-59  
  establecer comprensiones, 64

while bucles, 63  
convertidores (xlwings), 196  
mudado  
  tipos de datos con el módulo excel.py, 157-158  
  índices a columnas (pandas), 88  
  objetos al tipo de datos booleanos, 148  
  cadenas al tipo de datos de fecha y hora, 127  
  Hora universal coordinada (UTC), 130, 293  
  copias (DataFrames)

devuelto por métodos, 89  
vistas versus, 107  
copias (NumPy), vistas versus, 83-84  
método de copia, 88  
copias superficiales versus profundas, 295  
método corr (pandas), 135  
correlación en el análisis de series de tiempo, 133-136  
contadores de variables en bucles, 62  
Resultados de la prueba COVID-19, informes retrasados, 5  
Tiempos de CPU, 172  
llamadas entre aplicaciones (xlwings), minimizando, 206, 272-273  
compatibilidad multiplataforma, 17  
Matrices CSE (Excel), 260  
CSS (hojas de estilo en cascada), 177  
Archivos CSV  
  exportar datos de DataFrame como, 120-121  
  importar a DataFrames, 121-123  
cuDF, 138

en Windows, listar archivos en, 23

en macOS

enumerar archivos en, 23viendo el  
camino completo, 22complementos

personalizados (xlwings), 220, 257funciones  
personalizadas (ver UDF)

## D

Dask, 138

alineación de datos (pandas), 85, 89, 103

Expresiones de análisis de datos (DAX), xi

análisis de datos con capa de datos de  
pandas (ver pandas), 7

partes de datos (DataFrames), formato, 176-177

estructuras de datos

diccionarios, 55-56

lista de, 58

liza, 53-55

propósito de, 52

conjuntos 57-58

tuplas 57

en VBA, 52

tipos de datos

booleano 47-49, 148

convertir con el módulo excel.py, 157-158

en DataFrames, 87

datetime, convirtiendo cadenas a, 127

mutable versus inmutable, 294

numérico, 45-46

de matrices NumPy, 78-79

propósito de, 43

instrumentos de cuerta, 49-50

validación de datos en Python Package Tracker, 243

Visor de datos (código VS), 288visor de datos, Excel

como, 184-185ejemplo de database.py, 246bases de  
datos

conexiones, 232-233

Inyección SQL, 236-237

Consultas SQL, 234-236

estructura para el caso de Python Package Tracker  
estudio, 231-232

tipos de, 229-230

DataFrames

aplicando funciones, 105-107

operaciones aritméticas, 103-104

columnas, 90-91concatenando

108-109copias

devuelto por métodos, 89

vistas versus, 107

creando 87

estadísticas descriptivas, 111-112

datos duplicados, 102-103Hojas de

cálculo de Excel versus, 86-87métodos

de exploración, 122exportador

como archivos CSV, 120-

121propósito de, 119

formateo en Excel, 173-177

en el caso de éxito de Tendencias de Google, 258-263

agrupación de datos, 112

importador

como archivos CSV, 121-

123propósito de, 119

índices, 88-90

unión, 109-111

limitaciones, 138

derritiendo datos, 114

fusionando 110-111datos

perdidos, 100-101datos

dinámicos, 113-114

Graficado

lista de bibliotecas de trazado,

119con Matplotlib, 115-116

con Plotly, 117-119propósito  
de, 115

leer / escribir en Excel con xlwings,  
195-196

seleccionando datos

por indexación booleana, 94-96por

etiqueta, 92-93

con MultiIndexes, 96-97por

puesto, 93-94Serie versus, 85-

86, 93configuración de datos

agregando columnas, 100por

indexación booleana, 98-99por

etiqueta, 98

por puesto, 98

reemplazando valores, 99

Consultas SQL versus, 234

columnas de texto, 105

análisis de series de tiempo (ver análisis de series de tiempo)

transposición, 91

fecha número de serie (Excel), 69

tipo de datos de fecha y hora, convirtiendo cadenas a,

127módulo de fecha y hora, 69-70, 293-294

DatetimeIndex (pandas)  
creando 126-128  
filtración, 128-129zonas horarias en, 129-130  
paquete dateutil, 293  
función rango\_fecha (pandas), 126DAX (Expresiones de análisis de datos), xi  
desactivar entornos Conda, 282  
depuración  
Rastreador de paquetes de Python, 248-249en el editor de texto, 34UDF, 269-271  
  
en VS Code, 285-287tipo de datos decimal, 46  
lenguajes declarativos, 234  
decoradores (xlwings)  
orden de, 262  
propósito de, 254  
sub decorador, 276-277  
copias profundas, 295  
  
def palabra clave, sesenta y cinco  
argumentos predeterminados, objetos mutables como, 297  
nombres definidos (Excel), creando en xlwings, 200-202  
definir funciones, sesenta y cinco  
declaración del (listas), 54  
borrando  
columnas (pandas), 90  
Ambientes Conda, 282  
elementos de listas, 54  
dependencias  
despliegue de xlwings, 218-219  
en xlwings, 204  
despliegue  
definido 218  
de xlwings  
jerarquía de configuración, 220  
Dependencias de Python, 218-219  
ajustes, 221-222libros de trabajo  
independientes, 219-220ejemplo  
describe.py, 258estadísticas descriptivas, 111-112funciones deterministas, 274entorno de desarrollo (ver Anaconda;  
  
Cuadernos Jupyter; VS Code)  
dict constructor, 58  
diccionarios, 55-56comprensiones de diccionario, 64comando dir, 23

directorios  
cambiando, 23  
directorios actuales  
enumerar archivos en, 23viendo el camino completo, 22director principal, cambiando a, 23  
deshabilitar la activación automática del entorno Conda  
  
mentos, 283  
docstrings, 71  
documentación, 50  
notación de puntos, 45  
reducción de muestreo, 136  
controladores (para bases de datos), 232-233  
método dropna (pandas), 101método drop\_duplicates (pandas), 102Principio SECO, 8  
datos duplicados en DataFrames, 102-103  
método duplicado (pandas), 102matrices dinámicas (Excel), 258-263mecanografía dinámica, 44  
  
**mi**  
modo de edición (cuadernos de Jupyter), 31-32  
editar archivos (Excel)  
con OpenPyXL, 162-163  
con xlutils, 169  
elementos de matrices (NumPy), obteniendo y configurando tintineo 82-83  
Emacs, 40  
habilitar macros (Excel), 213finalizando la sesión interactiva de Python, 25puntos finales, 228  
parámetro del motor (función read\_excel o to\_excel ciones), 157  
enumerar la función, 61  
Variables de entorno, 221  
manejo de errores  
en Python Package Tracker, 238-240en VBA, dieciséis  
caracteres de escape en cadenas, 49  
EuSpRIG (European Spreadsheet Risks Interest Grupo), 5  
ejemplos  
función add\_package (paquete Python Rastreador), 245-246  
database.py, 246  
describe.py, 258  
first\_project.py, 212-213  
first\_udf.py, 253

función `get_interest_over_time(google_trends.py)`, 265-266  
`importacionesub.py`, 276  
`pep8_sample.py`, 70-72  
función de trazado (`google_trends.py`), 268-269  
`ingresos.py`, 273  
`sales_report_pandas.py`, 145-146  
`sales_report_xlwings.py`, 203  
`temperature.py`, 66

Sobresalir  
cálculos basados en matrices, 80  
gráficos de automatización (ver `xlwings`), 114  
(ver también parcelas)  
creando en OpenPyXL, 160-162  
creando en `XlsxWriter`, 163-165  
creando en `xlwings`, 198-199  
como visor de datos, 184-185  
DataFrames, lectura / escritura con `xlwings`, 195-196  
fecha número de serie, 69  
nombres definidos, creando en `xlwings`, 200-202  
archivos (ver archivos)  
flota en, 46  
formatear DataFrames en, 173-177  
como frontend  
instalar el complemento `xlwings`, 210-211  
propósito de, 209  
comando de inicio rápido, 212  
Ejecutar el botón principal, 212-213  
Función `RunPython`, 213-217  
historia de, 3  
instancias, 186, 192  
Cuadernos Jupyter versus, 28  
idioma y configuración regional, 187-188  
logaritmos en, 131

permisos de macOS, 186  
Excel moderno 11  
Administrador de nombres, 201  
en las noticias, 5-6  
modelo de objeto, 186-193  
imágenes, Matplotlib traza como, 199-200  
como lenguaje de programación, 4-5  
programación, lectura / escritura de archivos versus,  
**XIV**  
Ventajas de Python para, 12  
compatibilidad multiplataforma, 17  
características del lenguaje moderno,  
diccionarios-17  
legibilidad, 13-14

computación científica, 15  
biblioteca estándar y administrador de paquetes, 14-15  
informe de estudio de caso, 143-147, 178, 202-203  
hojas de cálculo, DataFrames versus, 86-87  
libros de trabajo independientes, 219-220  
limitaciones del análisis de series de tiempo, 125  
Confíe en el acceso a la configuración del modelo de proyecto de VBA, 252  
rango usado, 167  
control de versiones, 9-11  
versiones de, xiv-xiv  
procesos zombies, 205  
módulo `excel.py`, 157-158  
Clase `ExcelFile` (pandas), 147-152  
Clase `ExcelWriter` (pandas), 152-153  
excepciones (ver manejo de errores)  
método de expansión (`xlwings`), 195  
exportar datos de DataFrame  
como archivos CSV, 120-121  
propósito de, 119  
extraer código de macro (Xlsxwriter) de xlsm  
archivos, 164-165

## F

f-strings (cadenas literales formateadas), 49  
Tipo de datos falso booleano, 47-49  
método de llenado (pandas), 137  
objeto figura (Matplotlib), 199-200  
extensiones de archivo, visualización, 23-24  
rutas de archivo  
absoluto versus relativo, 247  
globbing 146  
Clase de ruta, 145  
en Windows como cadenas sin procesar, 120  
archivos  
en el directorio actual, listado, 23  
edición  
con OpenPyXL, 162-163  
con xlutils, 169  
leyendo  
limitaciones en pandas, 154  
con OpenPyXL, 159-160, 171  
con pandas, 147-152  
en paralelo, 171-173  
programar Excel versus, xiv  
con `pyxlsb`, 165-166  
de URL, 151  
con `xlrd`, 166-167, 170

escribiendo  
limitaciones en pandas, 154con  
OpenPyXL, 160-162, 170con  
pandas, 152-153programar  
Excel versus, xivcon XlsxWriter,  
163-165, 170con xlwt, 168-169  
métodofillna (pandas), 101  
filtración

DataFrames, 94-96DatetimeIndex  
(pandas), 128-129ejemplo de  
first\_project.py, 212-213ejemplo de  
first\_udf.py, 253flake8, 73

tipo de datos flotantes, 45-46  
tipo de datos float64 (NumPy), 78-79  
inexactitudes de punto flotante, 46  
Números de punto flotante, 45Para  
cada declaración (VBA), 60para bucles,  
60-63  
claves externas (bases de datos), 232, 236  
Controles de formulario (Excel), 215  
formateo  
DataFrames en Excel, 173-177objetos  
de fecha y hora en cadenas, 70  
cadenas en DataFrame, 105-106  
llenado hacia adelante, 137  
tipo de datos de fracción, 46  
frente termina  
definido 209  
Excel como  
instalar el complemento xlwings, 210-211  
propósito de, 209  
comando de inicio rápido, 212  
Ejecutar el botón principal, 212-213  
Función RunPython, 213-217  
propósito de, 241  
en Python Package Tracker, 241-245  
ejecutables congelados, 219uniones externas  
completas, 109funciones

aplicando a DataFrame, 105-107  
vocablo, 44, 66, 295-297  
decoradores (xlwings)  
orden de, 262  
propósito de, 254  
sub decorador, 276-277definiendo,  
sesenta y cinco  
determinista, 274

expresiones lambda, 106propósito  
de, sesenta y cinco  
recalcular (Excel), 255funciones  
universales (NumPy), 80-81funciones  
definidas por el usuario (ver UDF) VBA,  
8, 194  
volátil (Excel), 255  
módulo de funciones, 275

**G**  
GRAMO  
obtener método (diccionarios), 56  
OBTENER solicitudes, 226  
obteniendo elementos de matriz (NumPy), 82-  
83función get\_interest\_over\_time (goo-  
ejemplo de gle\_trends.py), 265-266  
Git, 9-11  
globbing 146  
Google Colab, 33  
Guía de estilo de Google para Python, 72Estudio de  
caso de Tendencias de Google  
DataFrames y matrices dinámicas, 258-263  
depurar UDF, 269-271explicación de Google  
Trends, 257-258recuperacion de datos, 263-  
267trazar datos, 267-269

interfaces gráficas de usuario (ver frontends) líneas  
de cuadrícula (Excel), ocultación, 244agrupando  
datos de DataFrame, 112GUI (ver interfaces)

**H**  
método de la cabeza (pandas), 121  
encabezados para columnas DataFrame, 174-175  
mapas de calor (Plotly), 135  
documentación de ayuda, 50  
valores hexadecimales para colores, 161-162  
ocultar líneas de cuadrícula (Excel), 244  
HoloViews, 119  
datos homogéneos, 78  
Página web de Horror Stories, 5  
Códigos de estado HTTP, 228

**I**  
Ibis, 138  
IDE (entornos de desarrollo integrados),  
35, 40-41  
si declaraciones, 59-60  
atributo iloc (pandas), 93-94, 98

objetos inmutables, 57, 294-295  
saltos de línea implícitos, 54  
declaración de importación, 66-69  
importador

Datos del marco de datos  
como archivos CSV, 121-123  
propósito de, 119  
módulos, 66-69  
ejecutar scripts versus, 247  
UDF, 252-257  
ejemplo de importaciones sub.py, 276  
método imshow (Plotly), 135 en operador, 54, 95  
indexación en Python, 51  
objetos de rango (xlwings), 190 en VBA, 51  
basado en cero versus uno, 191  
índices para DataFrames, 88-90, 174-175  
basado en el tiempo, 126  
método de información (pandas), 87, 121, 147  
método init, 291-292  
 inicialización (de objetos), 291  
uniones internas, 109  
entradas, capa separada para, 7  
instalando Anaconda, 20-21  
paquetes 26, 282  
Plotly, 117  
pytrends, 26  
pyxlsb, 26  
VS Code, 36  
xlutils, 166  
complemento xlwings, 210-211  
instancias (de Excel), 186, 192  
instanciación, 45, 69, 291  
tipo de datos int, 45-46  
enteros 45

entornos de desarrollo integrados (IDE), 35, 40-41

IntelliSense, 34

sesión interactiva de Python  
finalizando, 25  
a partir de, 24-25  
operador isin (pandas), 95  
método isna (pandas), 101  
método de artículos, 62

## J

Notación de objetos JavaScript (JSON), 226-227

condición de unión, 111

método de unión (pandas), 109-111

unión

tablas de base de datos, 230

DataFrames, 109-111

JSON (notación de objetos JavaScript), 226-227

módulo json, 226

Núcleo de Jupyter, 33

Cuadernos Jupyter

células

modo de edición versus modo de comando, 31-32

descripción operativa, 29-31  
producción, 30

en los scripts de Python, 289

orden de ejecución, 32

proveedores en la nube, 33

comentarios 47

Excel versus, 28

comandos mágicos, 116

propósito de, 19, 27-28

renombrar, 29

corriendo, 28-29, 287-

288  
guardar en VS Code, 288

Apagando, 33

VS Code versus, 34

JupyterLab, 41

## K

Kaggle, 33

combinaciones de clave / valor (ver diccionarios)

atajos de teclado

para comentarios, 47

Cuadernos Jupyter, 31-32  
argumentos de palabras clave (para funciones), 66  
Koalas 138

IDE de Komodo, 41

## L

etiquetas

seleccionar datos por (pandas), 92-93

configuración de datos por (pandas), 98 en

VBA, 17

expresiones lambda, 106  
funciones lambda (Excel), 4

Configuración de idioma (Excel), 187-188  
capas de aplicaciones, 6-7

se une a la izquierda, 109  
función len (listas), 54  
saltos de línea, 54  
pelusa 73  
lista de comprensiones, 63  
constructor de listas, 58  
separadores de lista (Excel), cambiando, 187-188  
enumerar los archivos de directorio actuales, 23liza, 53-  
55  
literales, 58, 69  
atributo loc (pandas), 92-93, 98  
devoluciones de registros, 131-132log  
ufunc (NumPy), 132logaritmos en Excel y  
Python, 131Historia de la ballena de  
Londres, 5buclees  
para bucles, 60-63  
while bucles, 63  
Lotus 1-2-3, 3  
paquetes de bajo nivel, 158  
minúsculas, transformando a / desde mayúsculas,  
50decorador lru\_cache, 275ls comando, 23  
  
lxml, 170  
  
**METRO**  
Lenguaje de fórmulas M (Power Query), xi  
Mac OS  
Anaconda Prompt, corriendo, 22  
directorio actual  
enumerar archivos en, 23  
viendo el camino completo, 22  
extraer código de macro (Xlsxwriter), 165  
extensiones de archivo, visualización, 24  
separadores de lista (Excel), cambiando, 188  
permisos en, 186, 193complementos de cinta,  
211  
Terminal, 21-22  
deshabilitar la activación automática de Conda envi-  
ambiente 283  
Código VS  
configurar, 37  
instalación 36  
dependencias de xlwings, 204  
código de macro (Xlsxwriter), extrayendo de xlsm  
archivos, 164-165objeto  
macro (xlwings), 194macros  
habilitar (Excel), 213  
  
en Python Package Tracker, 244  
corriendo (Excel), 214  
comandos mágicos (cuadernos de Jupyter), 116, 172  
Celdas de Markdown (cuadernos Jupyter), 30-31  
operadores matemáticos, 46Matplotlib, 115-116  
  
trazados como imágenes de Excel, 199-  
200matriz, 53, 80  
método medio (pandas), 112  
función derretir (pandas), 114  
derritiendo datos de DataFrame,  
114memorización, 274  
método de fusión (pandas), 110-111  
fusionar solicitudes, 10  
fusionando  
DataFrames, 110-111  
diccionarios, 56  
encadenamiento de métodos, 89  
métodos  
correspondientes operadores aritméticos para  
(pandas), 104  
Exploración de DataFrame, 122de  
DataFrames, copias devueltas, 89  
documentación de ayuda, 50  
propósito de, 45, 291-293  
métodos de cadena en DataFrames, 105  
Acceso Microsoft, 230  
migrar bases de datos, 230  
minimizar las llamadas entre aplicaciones (xlwings),  
206, 272-273  
datos faltantes en DataFrames, 100-101  
controlador de vista de modelo (MVC), 240  
Excel moderno 11  
características del lenguaje moderno en Python versus  
Sobresalir, dieciséis-17  
Modin, 138, 173  
modularidad, 6-7  
ruta de búsqueda del módulo (Python), 215-216, 221  
modulos  
módulo de fecha y hora, 69-70, 293-294  
importador, 66-69  
ejecutar scripts versus, 247  
módulo de zona horaria, 294  
MongoDB, 230  
medias móviles en el análisis de series de tiempo, 137  
MultiIndexes (pandas), seleccionando datos por, 96-97  
Multiplan, 3  
multiprocesamiento, 171-173  
objetos mutables

- como argumentos de función, 295-297  
objetos inmutables versus, 294-295  
MVC (modelo-vista-controlador), 240  
mypy 73  
MySQL, 230
- norte**  
Administrador de nombres (Excel), 201  
rangos con nombre  
creando en xlwings, 200-202  
Python Package Tracker, 242  
espacios de nombres de scripts, 68  
nombrar
- columnas (pandas), 90  
índices (pandas), 88  
guiones, 68  
variables, 44
- Valores de NaN  
manejo al leer archivos de Excel, 149  
propósito de, 100-101  
reemplazando con ceros, 103  
listas anidadas, 53, 77  
noticias, 48  
Excel en, 5-6  
Ninguna constante incorporada, 48  
normalización (bases de datos), 230-231  
Bases de datos NoSQL, 230
- celdas del cuaderno (ver celdas)  
Notepad ++, 40  
tipos de datos numéricos, 45-46  
NumFOCUS, 15  
NumPy, 4  
matrices  
radiodifusión, 79-80  
constructores, 83  
tipos de datos, 78-79  
obtener y configurar elementos, 82-83  
descripción operativa, 77-79  
funciones universales (ufuncs), 80-81  
vectorización, 79-80  
vistas versus copias, 83-84  
problemas de análisis de datos en, 84  
propósito de, 77
- O**  
modelo de objetos (Excel), 186-193  
mapeador relacional de objetos (ORM), 232  
objetos  
atributos y métodos, 291-293  
documentación de ayuda, 50
- propósito de, 45  
como instancias de clase, 69, 291  
clases y, 291-293  
colecciones, 186  
convertir a tipo de datos booleanos, 148  
documentación de ayuda, 50  
inicialización, 291  
mutable como argumentos de función, 295-297  
mutable versus inmutable, 294-295  
propósito de, 44  
variables, 44
- método ohlc (pandas), 137  
oletools, 178  
indexación basada en uno, 191  
matrices NumPy unidimensionales, 78  
relaciones de uno a muchos (bases de datos), 232  
modo abierto (código VS), 40  
Software de código abierto (OSS), 15  
abrir instancias de Excel, 186  
OpenPyXL
- editar archivos, 162-163  
formatear DataFrames  
partes de datos, 176  
índices y encabezados, 174  
parallelización, 172  
leer archivos, 159-160, 171  
cuándo usar, 156  
escribir archivos, 160-162, 170  
XlsxWriter versus, 156  
precedencia del operador, 95  
optimizando  
paquetes de lector / escritor, 169-173  
UDF  
almacenamiento en caché 274-276  
minimizar las llamadas entre aplicaciones  
(xlwings), 272-273  
valores brutos, 273  
rendimiento de xlwings, 206-207  
argumentos opcionales (para funciones), sesenta y cinco  
opciones para el objeto de rango (xlwings), 196-198  
Oráculo, 230
- ORM (mapeador relacional de objetos), 232  
OSS (software de código abierto), 15  
uniones externas, 109  
producción  
en cuadernos de Jupyter, 30  
capa separada para, 7

# PAG

administradores de paquetes, 14-15, 25-27

paquetes

- agregando a Python Package Tracker, 223-225
- edificio, 221
- instalación 26, 282paquetes
- de lector / escritor
  - módulo excel.py, 157-158
  - lista de, 178
  - OpenPyXL, 159-163optimización para archivos grandes, 169-173pxlsb, 165-166cuándo usar, 156-157xird, 166-167

XlsxWriter, 163-165

- xlutils, 169
- xlwt, 168-169

traspuesta, 157

versiones de, 26-27

pandas 4

DataFrames

- aplicando funciones, 105-107
- operaciones aritméticas, 103-104
- columnas, 90-91concatenando 108-109copias, 89

creando 87

estadísticas descriptivas, 111-112

- datos duplicados, 102-103Hojas de cálculo de Excel versus, 86-87métodos de exploración, 122exportador, 119

formateo en Excel, 173-177

en el caso de éxito de Tendencias de Google, 258-263

agrupación de datos, 112

importador, 119

índices, 88-90

unión, 109-111

limitaciones, 138

derritiendo datos, 114

fusionando 110-111datos

perdidos, 100-101datos

dinámicos, 113-114

Graficado, 115-119

leer / escribir en Excel con xlwings, 195-196

seleccionar datos, 92-97Serie

versus, 85-86, 93configuración

de datos, 97-100

Consultas SQL versus, 234

columnas de texto, 105

transponer 91

vistas versus copias, 107

DatetimeIndex

creando 126-128filtración,

128-129zonas horarias en,

129-130Archivos de Excel

limitaciones, 154

leyendo, 147-152

escribiendo, 152-

153NumPy (ver NumPy)

tipos de parcelas, 118

paquetes de lector / escritor

módulo excel.py, 157-158

lista de, 178

OpenPyXL, 159-163optimización para

archivos grandes, 169-173pxlsb, 165-

166cuándo usar, 156-157xird, 166-167

XlsxWriter, 163-165

xlutils, 169

xlwt, 168-169

análisis de series temporales

correlación, 133-136cambio

porcentual, 131-133rebase

133-136remuestreo 136-137

ventanas rodantes, 137

cambiando, 131-133paralelización, 171

-173directorio principal, cambiando a, 23

analizar cadenas en objetos de fecha y hora, 70

pasar declaración, 58-59Clase de ruta, 145

módulo pathlib, 145

método pct\_change (pandas), 132

Guía de estilo PEP 8

ejemplos 70-72

pelusa 73

ejemplo de pep8\_sample.py, 70-72cambio

porcentual en el análisis de series de tiempo,

131-133

optimización del rendimiento

de paquetes de lector / escritor, 169-173

de UDF

almacenamiento en caché 274-276

minimizar las llamadas entre aplicaciones (xlwings), 272-273  
valores brutos, 273  
en xlwings, 206-207 permisos en macOS, 186, 193 imágenes (Excel), Matplotlib traza como, 199-200 Almohada, 200

pepita, 14-15, 25-27  
datos dinámicos de DataFrame, 113-114 función pivot\_table (pandas), 113-114 ejemplo de función de trazado (google\_trends.py), 268-269  
método de la trama (pandas), 115, 267-269 Plotly, 117-119  
mapas de calor, 135  
instalación 26  
parcelas  
como imágenes de Excel, 199-200 lista de bibliotecas de trazado, 119 en Matplotlib, 115-116 en Plotly, 117-119 tipos de (pandas), 118 Graficado

Datos del marco de datos  
lista de bibliotecas de trazado, 119 con Matplotlib, 115-116 con Plotly, 117-119 propósito de, 115 con UDF, 267-269  
método pop (listas), 54  
posición (pandas)  
seleccionando datos por, 93-94  
configuración de datos por, 98  
argumentos posicionales (para funciones), 66  
Solicitudes POST, 226  
PostgreSQL, 230, 248  
Power BI, 12  
Power Pivot, xi, 11  
Power Query, xi, 11  
El programador pragmático (Hunt y Thomas), 8  
Capa de presentación, 6  
claves primarias (bases de datos), 231  
función de impresión, 38, 269  
programación de Excel, lectura / escritura de archivos ver-sus, xiv  
lenguajes de programación  
mejores prácticas  
Principio SECO, 8

separación de intereses, 6-7  
pruebas, 8  
control de versiones, 9-11  
Excel como, 4-5  
propiedades del objeto de la aplicación (xlwings), 207  
números pseudoaleatorios, generando, 83 PTVS (Python Tools para Visual Studio), 35 solicitudes de extracción, 10 comando pwd, 22 PyArrow, 138 carpeta de pycache, 68 PyCharm, 40 PyDev, 41 pyexcel, 178 PyExcelerate, 178 PyInstaller, 219 pylightxl, 178 PyPI (índice de paquetes de Python), 14, 224, 228 PyPy, 14 PySpark, 138 Pitón  
ventajas para Excel, 12  
compatibilidad multiplataforma, 17  
características del lenguaje moderno, dieciséis-17 legibilidad, 13-14 computación científica, 15  
biblioteca estándar y administrador de paquetes, 14-15  
Operadores booleanos de distribución  
Anaconda (ver Anaconda), 47  
clases 45  
comentarios 47  
flujo de control  
bloques de código, 58-59  
expresiones condicionales, 60  
comprensiones de diccionario, 64  
para bucles, 60-63 si declaraciones, 59-60 lista de comprensiones, 63  
pasar declaración, 58-59 establecer comprensiones, 64

while bucles, 63  
estructuras de datos  
diccionarios, 55-56  
lista de, 58  
liza, 53-55  
propósito de, 52  
conjuntos 57-58  
tuplas 57

tipos de datos  
booleano 47-49  
mutable versus inmutable, 294  
numérico, 45-46propósito de, 43

instrumentos de cuerda,  
49-50funciones  
vocación, 44, 66, 295-297  
definiendo, sesenta y cinco  
propósito de, sesenta y cinco

historia de, 3-4  
indexación, 51  
sesión interactiva  
finalizando, 25  
a partir de, 24-25  
saltos de línea, 54  
logaritmos en, 131  
permisos de macOS, 186  
operadores matemáticos, 46ruta de  
búsqueda del módulo, 215-216, 221  
módulos  
módulo de fecha y hora, 69-70, 293-294  
importador, 66-69módulo de zona horaria,  
294

objetos  
atributos y métodos, 45, 291-293como  
instancias de clase, 69clases y, 291-293  
documentación de ayuda, 50

inicialización, 291  
mutable como argumentos de función, 295-297  
mutable versus inmutable, 294-295propósito  
de, 44

paquetes, edificio, 221  
Guía de estilo PEP 8  
ejemplos 70-72  
pelusa 73  
propósito de, xi-xii  
guardar archivos, 256  
guiones (ver guiones)  
rebanar 52  
variables, 44  
versiones de, xiv  
dependencias de xlwings, 218-219

comando de Python, 24  
Biblioteca de análisis de datos de Python (ver  
pandas) Ventana interactiva de Python (Código VS),  
289Intérpretes de Python (xlwings), 221Estudio de  
caso de Python Package Tracker

estructura de la aplicación, 240  
backend 245-248  
depuración 248-249  
Interfaz, 241-245

bases de datos  
conexiones, 232-233  
Inyección SQL, 236-237  
Consultas SQL, 234-236  
estructura de, 231-232  
tipos de, 229-230manejo

de errores, 238-240paquetes,  
agregando, 223-225API web,  
226-229Python REPL, 24-25

Herramientas de Python para Visual Studio (PTVS), 35  
Estilo pitónico, 59  
Configuración de PYTHONPATH (xlwings), 215-216,  
221, 256

pytrends, 26, 263-266

pyxlsb  
instalación 26  
leer archivos, 151, 165-166  
cuándo usar, 156

**Q**  
consultas (SQL), 234-236comando de  
inicio rápido (xlwings), 212  
importar UDF, 252-257Función  
RunPython y, 213-216salir de  
comando, 25

**R**  
números aleatorios, generando, 83  
función de rango, 61-61objeto de  
rango (xlwings), 189-191  
convertidores, 196  
opciones, 196-198  
cuerdas crudas, 120  
valores brutos, 206, 273función  
de lectura (excel.py), 158  
legibilidad de Python, 13-14  
paquetes de lector / escritor  
módulo excel.py, 157-158  
lista de, 178  
OpenPyXL, 159-163optimización para  
archivos grandes, 169-173pyxlsb, 165-  
166cuándo usar, 156-157xlrd, 166-167

- XlsxWriter, 163-165  
xlutils, 169  
xlwt, 168-169  
leyendo  
DataFrames en Excel con xlwings, 195-196  
archivos (Excel)  
limitaciones en pandas, 154con  
OpenPyXL, 159-160, 171con  
pandas, 147-152en paralelo,  
171-173programar Excel  
versus, xivcon pyxlsb, 165-166  
de URL, 151  
  
con xlrd, 166-167, 170función read\_csv  
(pandas), 121-123, 127función read\_excel  
(pandas), 86, 144, 147-152,  
157  
rebase en el análisis de series de tiempo, 133-  
136recalcular funciones (Excel), 255registros  
(bases de datos), 231globbing recursivo, 146  
Redis, 230  
  
configuración regional (Excel), 187-188  
método de reinindexación (pandas), 89bases  
de datos relacionales, 229-230caminos  
relativos, 23, 247quitando  
  
datos duplicados (pandas), 102filas de  
datos faltantes (pandas), 101  
renombrar  
columnas (pandas), 90  
Cuadernos Jupyter, 29  
reordenar columnas (pandas), 91  
método de reemplazo (pandas), 99  
reemplazar valores (pandas), establecer datos por, 99  
informe de estudio de caso (Excel), 143-147, 178,  
202-203  
API de transferencia de estado representacional (REST), 227  
Paquete de solicitudes, 228  
argumentos requeridos (para funciones), sesenta y cinco  
método de remuestreo (pandas), 136remuestreo en el  
análisis de series de tiempo, 136-137reiniciando  
  
índices (pandas), 88  
Multiíndices (pandas), 97método  
reset\_index (pandas), 88función de  
remodelación (NumPy), 83método de  
resolución (clase de ruta), 145  
API REST (transferencia de estado representacional), 227  
  
Botón de reinicio (depurador de código VS), 287  
devuelve decoradores (xlwings), 262, 277  
declaración de retorno (para funciones), sesenta y  
cincoejemplo de revenue.py, 273método rglob  
(clase Path), 146se une a la derecha, 109  
  
método de balanceo (pandas), 137  
ventanas móviles en el análisis de series de tiempo, 137  
Botón Ejecutar archivo (Código VS), 39Ejecutar el botón  
principal (xlwings), 212-213ejecutar el orden de las celdas  
del cuaderno Jupyter, 32corriendo
- ### Aviso Anaconda
- en macOS, 22  
en Windows, 21  
código del editor de texto, 34  
Cuadernos Jupyter, 28-29, 287-288  
macros (Excel), 214  
guiones  
importar módulos versus, 247en VS  
Code, 37-40Código VBA en xlwings,  
193-194VS Code en Anaconda Prompt, 37  
Función RunPython (xlwings), 213-217,  
221
- ## S
- sales\_report\_openpyxl.py, 178  
sales\_report\_pandas.py ejemplo, 145-146  
sales\_report\_xlsxwriter.py, 178ejemplo de  
sales\_report\_xlwings.py, 203entornos de  
espacio aislado, 187, 193ahorro  
  
Cuadernos Jupyter en VS Code, 288  
Archivos de Python, 256  
escalares 79  
computación científica en Python, 15  
guiones  
Celdas del cuaderno Jupyter en, 289  
espacios de nombres de, 68  
nombrar 68  
corriendo  
importar módulos versus, 247  
en VS Code, 37-40  
desplazarse por el historial de comandos, 23  
Seaborn, 119  
seleccionar  
columnas (pandas), 93  
Datos del marco de datos  
por indexación booleana, 94-96

por etiqueta, 92  
con MultiIndexes, 96-97  
por puesto, 93-94auto  
argumento (clases), 292  
separación de intereses, 6-7, 240  
Serie  
operaciones aritméticas, 104  
DataFrames versus, 85-86, 93  
estadísticas descriptivas, 111-112  
establecer comprensiones, 64  
constructor de conjuntos, 57-58  
lenguajes basados en conjuntos, 234  
conjuntos 57-58  
configuración  
elementos de matriz (NumPy), 82-83  
Datos del marco de datos  
agregando columnas, 100  
indexación booleana, 98-99  
etiqueta, 98  
por puesto, 98  
reemplazando valores, 99  
índices (pandas), 88  
configuración para la implementación de xlwings, 221-222  
método set\_index (pandas), 88  
copia superficiales, 295  
objeto de hoja (xlwings), 187, 190  
hojas (ver archivos)  
método de cambio (pandas), 131  
cambio en el análisis de series de tiempo, 131-133  
Mostrar casilla de verificación de la consola (complemento xlwings), 249  
apagar los cuadernos de Jupyter, 33  
espacio en blanco significativo, 59  
devoluciones simples, 132

un solo clic en VS Code, 40  
rebanar  
matrices (NumPy), 82-83  
con etiquetas (pandas), 92  
en Python, 52  
objetos de rango (xlwings), 190  
clasificación  
índices (pandas), 89-90  
liza, 54  
método sort\_index (pandas), 89-90  
fuente de control, 9-11  
código de espagueti, 17

hojas de cálculo (Excel), DataFrames versus, 86-87  
Spyder, 40  
Inyección SQL, 236-237  
Consultas SQL, 234-236

Servidor SQL, 230  
SQLAlchemy, 232-233  
hacer cumplir las claves externas, 236  
Inyección SQL, 237  
SQLite, 229-230  
claves foráneas en, 236  
reemplazando con PostgreSQL, 248  
Extensión de código VS, 232  
libros de trabajo independientes (xlwings), 219-220  
biblioteca estándar (Python), 14-15  
iniciar sesión interactiva de Python, 24-25  
recursos apátridas, 227  
Barra de estado (código VS), 37

Paso en el botón (depurador de código VS), 286  
Botón Step Out (depurador de código VS), 287  
Botón Paso a paso (depurador de código VS), 286  
Botón de parada (depurador de código VS), 287  
almacenar funciones de VBA, 194  
método strftime, 70

instrumentos de cuerda, 49-50  
convertir al tipo de datos de fecha y hora, 127  
formateo en DataFrames, 105-106  
métodos en DataFrames, 105  
cuerdas crudas, 120  
método de tira (pandas), 105  
método strftime, 70  
guía de estilo para Python (consulte la guía de estilo PEP 8) propiedad de estilo (DataFrames), 177  
marco de estilo, 178  
sub decorador (xlwings), 276-277  
Texto sublime, 40  
suma función universal (NumPy), 81  
conmutación de paquetes de lector / escritor, 157  
resaltado de sintaxis, 30, 34

## T

mesa se une, 230  
tablas en Python Package Tracker, 242  
método de la cola (pandas), 121  
ejemplo de temperature.py, 66  
Terminal (macOS), 21-22  
deshabilitar la activación automática del entorno Conda  
ment, 283  
operadores ternarios, 60  
pruebas, 8  
columnas de texto en DataFrames, 105  
editores de texto  
lista de, 40  
VS Code (ver VS Code)

función de texto (SQLAlchemy), 237% tiempo  
de magia celular, 172análisis de series  
temporales  
correlación, 133-136  
DatetimeIndex (pandas)  
creando 126-128filtración,  
128-129zonas horarias en,  
129-130Limitaciones de  
Excel, 125  
cambio porcentual, 131-133  
rebase 133-136remuestreo  
136-137ventanas rodantes,  
137  
cambiando, 131-133serie de  
tiempo, propósito de, 125zonas  
horarias  
en DatetimeIndex (pandas), 129-130en  
Python, 293-294% timeit magia celular,  
172módulo de zona horaria, 294

títulos, agregando a DataFrames, 174función  
to\_datetime (pandas), 127método to\_excel  
(pandas), 152-153, 157transponer DataFrames,  
91Tipo de datos booleano verdadero, 47-49  
probar / excepto declaraciones, 238-240  
constructor de tuplas, 58

tuplas 57  
matrices NumPy bidimensionales, 78  
anotaciones de tipo, 73  
sugerencias de tipo, 73  
método tz\_convert (pandas), 130  
método tz\_localize (pandas), 130

**U**  
UDF (funciones definidas por el usuario)  
depuración 269-271Estudio de caso  
de Tendencias de Google  
DataFrames y matrices dinámicas,  
258-263  
explicación de Google Trends, 257-258  
recuperación de datos, 263-267trazar datos,  
267-269importador, 252-257optimización del  
rendimiento de  
almacenamiento en caché 274-276  
minimizar las llamadas entre aplicaciones  
(xlwings), 272-273  
valores brutos, 273

requisitos para, 252  
sub decorador, 276-277  
ufuncs (funciones universales en NumPy), 80-81UI  
(ver interfaces)  
Función ÚNICA (Excel), 263pruebas  
unitarias, 8  
funciones universales (NumPy), 80-81  
desembalaje de diccionarios, 56método de  
actualización (xlwings), 199actualizando  
xlwings, 210  
mayúsculas, transformando a / desde minúsculas,  
50muestreo, 136  
URL, lectura de archivos de, 151  
rango usado, 167  
interfaces de usuario (ver interfaces)  
funciones definidas por el usuario (ver UDF)  
UTC (hora universal coordinada), 130, 293

## V

Vaex, 138  
validando datos en Python Package Tracker, 243valores  
(pandas), configuración de datos reemplazando, 99  
Explorador de variables (cuadernos Jupyter), 287  
variables, 44  
VBA (Visual Basic para aplicaciones), xi  
complementos, 8  
Argumentos de las funciones ByRef y ByVal,  
295-296  
Declaración de caso, 56  
bloques de código, 59  
contadores de variables en bucles, 62  
estructuras de datos, 52  
Para cada declaración, 60  
funciones, 8, 194  
indexación, 51  
características del lenguaje moderno,  
dieciséis-17legibilidad, 13  
corriendo en xlwings, 193-194Función RunPython, 213-217,  
221Confie en el acceso a la configuración del modelo de  
proyecto de VBA,  
252  
variables, 44  
Con declaración, 150  
funciones de envoltura, 255  
Objeto de rango xlwings versus, 189-191  
VCS (sistema de control de versiones), 9  
vectorización, 79-80, 85, 103control de  
versiones, 9-11, 34versiones

de paquetes, 26-27  
de Windows, determinando, 20  
función de vista (xlwings), 184-185  
visita  
    directorio actual en macOS, 22  
    extensiones de archivo, 23-24  
    vistas, 23-24  
copias versus, 83-84, 107  
Empuje, 40

entornos virtuales, 27  
VisiCalc, 3  
Visual Basic para aplicaciones (ver VBA)  
Visual Studio, 35  
funciones volátiles (Excel), 255  
VS Code (código de Visual Studio)  
    ventajas 34-35  
    comentarios 47  
    componentes de, 19  
configurar, 36-37  
depurando con, 249, 270, 285-287  
instalación 36

Cuadernos Jupyter  
    corriendo, 287-288  
    ahorro, 288  
pelusa 73  
propósito de, 19

Secuencias de comandos de Python, en ejecución, 37-40  
Botón Ejecutar archivo, 39  
ejecutándose en Anaconda Prompt, 37  
haciendo un solo clic en, 40  
Extensión SQLite, 232  
Visual Studio versus, 35

## W

tiempo de pared, 172  
API web, 226-229  
while bucles, 63  
espacio en blanco, 59  
Ventanas  
    Controles ActiveX, 215  
    Anaconda Prompt, corriendo, 21  
    Símbolo del sistema, 21  
        deshabilitar la activación automática de Conda entorno, 283  
    directorio actual, lista de archivos en, 23  
    extraer código de macro (Xlsxwriter), 164  
    extensiones de archivo, 24  
    visualización, 24  
rutas de archivo como cadenas sin formato, 120  
Controles de formulario (Excel), 215  
ejecutables congelados, 219

separadores de lista (Excel), cambiando, 187  
Mostrar casilla de verificación de la consola (complemento xlwings), 249  
Utilice el servidor UDF, 221  
funciones definidas por el usuario (ver UDF) VS Code

configurar, 37  
instalación 36  
dependencias de xlwings, 204  
procesos zombies, 205  
IDE de Wing Python, 41  
WinPython, 218  
con declaración, 150  
funciones de envoltura VBA, 255  
función de escritura (excel.py), 158  
paquetes de escritor (ver paquetes de lector / escritor)  
escritura  
    DataFrames a Excel con xlwings, 195-196  
archivos (Excel)  
    limitaciones en pandas, 154  
    con OpenPyXL, 160-162, 170  
    pandas, 152-153  
    programar Excel versus, xiv  
    con XlsxWriter, 163-165, 170  
    con xlwt, 168-169

## X

xlrd  
    paralelización, 172  
    leer archivos, 166-167, 170  
    cuándo usar, 156  
archivos xls, lectura / escritura / edición, 166  
archivos xlsb, lectura, 151, 165-166  
archivos xlsm  
    habilitar macros, 213  
    extraer código de macro (Xlsxwriter), 164-165  
    XlsxWriter  
        formatear DataFrames  
            partes de datos, 176  
            índices y encabezados, 175  
    OpenPyXL versus, 156  
    cuándo usar, 156  
    escribir  
    archivos, 163-165, 170  
    xltrail, 10  
xlutils  
    editar archivos, 169  
    instalación 26, 166  
    cuándo usar, 156  
xlwings

- gráficos (Excel), creación, 198-199  
convertidores, 196  
DataFrames, lectura / escritura en Excel, 195-196  
nombres definidos (Excel), creando, 200-202  
dependencias, 204  
despliegue  
    jerarquía de configuración, 220  
    Dependencias de Python, 218-219  
    ajustes, 221-222  
    libros de trabajo  
    independientes, 219-220  
Excel como visor de datos, 184-185  
Excel como interfaz  
    instalar el complemento xlwings, 210-211  
    propósito de, 209  
    comando de inicio rápido, 212  
    Ejecutar el botón principal, 212-213  
    Función RunPython, 213-217  
Dependencia de Excel, 157  
Modelo de objetos de Excel y, 186-193  
permisos de macOS, 186  
faltan soluciones alternativas de funcionalidad, 207-208  
optimización del rendimiento, 206-207  
imágenes (Excel), Matplotlib traza como, 199-200
- propósito de, 183-184  
Rastreador de paquetes de Python (consulte Paquete de Python-rastreador de edad)  
opciones de objeto de rango, 196-198  
Mostrar casilla de verificación de la consola, 249  
actualización 210  
funciones definidas por el usuario (ver UDF)  
Código VBA, en ejecución, 193-194  
CLI de xlwings  
addin install comando, 210  
propósito de, 210  
comando de inicio rápido, 212  
xlwings PRO, 222  
xlwt  
    cuándo usar, 156  
escribir archivos, 168-169

## Z

- El Zen de Python, 13  
indexación de base cero, 191  
ceros, reemplazando los valores de NaN con, 103  
procesos zombies, 205

## Sobre el Autor

---

**Felix Zumstein** es creador y mantenedor de xlwings, un popular paquete de código abierto que permite la automatización de Excel con Python en Windows y macOS. También organiza las reuniones de xlwings en Londres y Nueva York para promover una amplia gama de soluciones innovadoras para Excel.

Como director ejecutivo de xltrail, un sistema de control de versiones para archivos de Excel, ha hablado con cientos de usuarios que utilizan Excel para tareas críticas de la empresa y, por lo tanto, ha obtenido una visión profunda de los casos de uso típicos y los problemas de Excel en varias industrias.

## Colofón

---

El animal en la portada de *Python para Excel* es una serpiente de coral falsa(*Anilius scytale*). Esta serpiente de colores brillantes, también conocida como la serpiente de pipa estadounidense, se encuentra en la región de Guayanas en América del Sur, la selva amazónica y Trinidad y Tobago.

La serpiente coral falsa crece hasta unos 70 cm de longitud y tiene bandas de color rojo brillante y negro. Su apariencia con bandas es similar a la de la serpiente de coral, de quien deriva uno de sus nombres comunes; sin embargo, a la serpiente de coral falsa le faltan las distintivas bandas amarillas de la serpiente de coral "verdadera". Su cuerpo tiene aproximadamente el mismo diámetro en la mayor parte de su longitud con una cola muy corta, lo que le da una apariencia de tubo. Sus ojos pequeños están cubiertos por grandes escamas en la cabeza.

Se ha observado que esta serpiente excavadora es ovovivípara. Se alimenta de escarabajos, anfibios, lagartijas, peces y otras serpientes pequeñas. La serpiente de coral falso también retiene espolones, o pequeños trozos de hueso que sobresalen cerca de su respiradero, que son vestigios de las caderas y ocasionalmente de los huesos de la parte superior de las piernas. Esta característica, junto con los huesos gruesos y la forma distintiva de su cráneo, hace que la serpiente de coral falsa se parezca mucho a la condición de lagarto ancestral de las serpientes.

El estado de conservación de la serpiente coral falsa es "Datos insuficientes", lo que significa que aún no hay suficiente información para juzgar la amenaza de su extinción. Muchos de los animales de las portadas de O'Reilly están en peligro; todos ellos son importantes para el mundo.

La ilustración de la portada es de Karen Montgomery, basada en un grabado en blanco y negro de *Historia Natural de la Cyclopedia inglesa*. Las fuentes de la portada son Gilroy Semibold y Guardian Sans. La fuente del texto es Adobe Minion Pro; la fuente del título es Adobe Myriad Condensed; y la fuente del código es Ubuntu Mono de Dalton Maag.



O'REILLY®

# Hay mucho mas de donde vino esto.

Experimente libros, videos, cursos de capacitación en línea en vivo y más de O'Reilly y nuestros más de 200 socios, todo en un solo lugar.

Obtenga más información en [oreilly.com/online-learning](http://oreilly.com/online-learning)