12 Programas en red

Si bien muchos de los ejemplos de este libro se han centrado en leer archivos y buscar datos en esos archivos, hay muchas fuentes diferentes de información cuando se considera internet.

En este capítulo simularemos ser un navegador web y recuperar páginas web utilizando el Protocolo de transporte de hipertexto (HTTP). Luego leeremos los datos de la página web y los analizaremos.

Protocolo de transporte de hipertexto - HTTP

El protocolo de red que alimenta la web es en realidad bastante simple y hay un soporte incorporado en Python llamado `sockets` que hace que sea muy fácil hacer conexiones de red y recuperar datos a través de esos sockets en un programa Python.

Un **socket** es muy parecido a un archivo, excepto que un solo socket proporciona una conexión bidireccional entre dos programas. Puedes leer y escribir en el mismo socket. Si escribes algo en un socket, se envía a la aplicación en el otro extremo del socket. Si lees desde el socket, se te proporcionan los datos que la otra aplicación ha enviado.

Pero si intentas leer un socket cuando el programa en el otro extremo del socket no ha enviado ningún dato, simplemente siéntate y espera. Si los programas en ambos extremos del socket simplemente esperan algunos datos sin enviar nada, esperarán por mucho tiempo.

Una parte importante de los programas que se comunican a través de Internet es tener algún tipo de protocolo. Un protocolo es un conjunto de reglas precisas que determinan quién debe ir primero, qué deben hacer, y luego cuáles son las respuestas a ese mensaje, y quién envía el siguiente, y así sucesivamente. En cierto sentido, las dos aplicaciones en cada extremo del socket están bailando y asegurándose de no pisarse los dedos de los pies.

Hay muchos documentos que describen estos protocolos de red. El protocolo de transporte de hipertexto se describe en el siguiente documento:

Copyright 2026

http://www.w3.org/Protocols/rfc2616/rfc2616.txt

Este es un documento largo y complejo de 176 páginas con muchos detalles. Si lo encuentras interesante, siéntete libre de leerlo todo. Pero si echas un vistazo a la página 36 de RFC2616 encontrarás la sintaxis de la solicitud GET. Para solicitar un documento de un servidor web, hacemos una conexión al servidor `www.pr4e.org` en el puerto 80 y luego enviamos una línea del formulario

```
GET http://data.pr4e.org/romeo.txt HTTP/1.0
```

donde el segundo parámetro es la página web que solicitamos, y luego también enviamos una línea en blanco. El servidor web responderá con cierta información de encabezado sobre el documento y una línea en blanco seguida del contenido del documento.

El navegador web más simple del mundo

Quizás la forma más fácil de mostrar cómo funciona el protocolo HTTP es escribir un programa Python muy simple que haga una conexión a un servidor web y siga las reglas del protocolo HTTP para solicitar un documento y mostrar lo que el servidor envía.

https://trinket.io/embed/python3/734cda1053

Primero, el programa realiza una conexión al puerto 80 en el servidor [www.py4e.com] (http://www.py4e.com). Dado que nuestro programa desempeña el papel de "navegador web", el protocolo HTTP dice que debemos enviar el comando GET seguido de una línea en blanco.

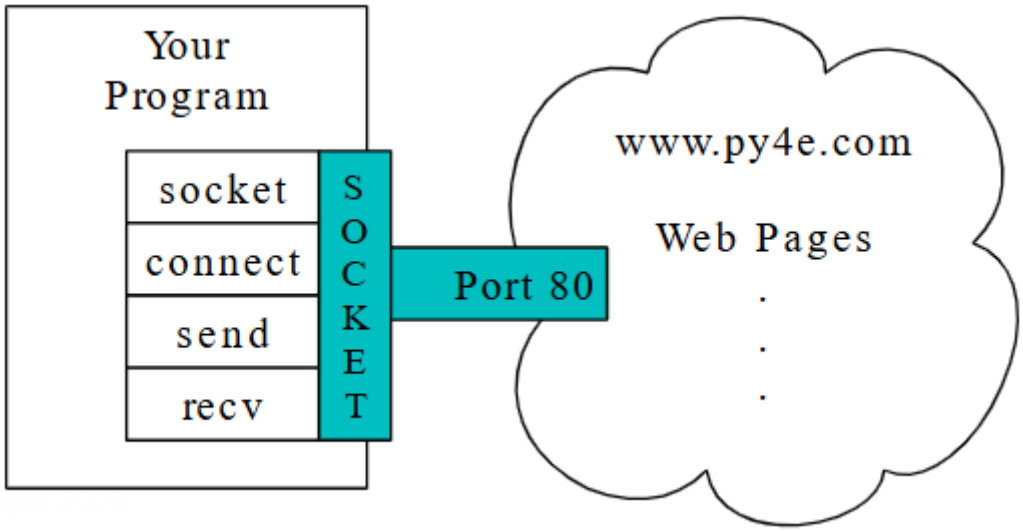


Imagen - A Socket Connection

Una vez que enviamos esa línea en blanco, escribimos un bucle que recibe datos en fragmentos de 512 caracteres del socket e imprime los datos hasta que no haya más datos para leer (es decir, recv() devuelve una cadena vacía).

El programa produce la siguiente salida:

```
HTTP/1.1 200 OK
Date: Sun, 14 Mar 2010 23:52:41 GMT
Server: Apache
Last-Modified: Tue, 29 Dec 2009 01:31:22 GMT
ETag: "143c1b33-a7-4b395bea"
Accept-Ranges: bytes
Content-Length: 167
Connection: close
Content-Type: text/plain

But soft what light through yonder window breaks
It is the east and Juliet is the sun
Arise fair sun and kill the envious moon
Who is already sick and pale with grief
```

La salida comienza con los encabezados que el servidor web envía para describir el documento. Por ejemplo, el encabezado `Content-Type` indica que el documento es un documento de texto sin formato (`text/plain`).

Una vez que el servidor nos envía los encabezados, agrega una línea en blanco para indicar el final de los encabezados y luego envía los datos reales del archivo `romeo.txt`.

Este ejemplo muestra cómo realizar una conexión de red de bajo nivel con sockets. Los sockets se pueden utilizar para comunicarse con un servidor web o con un servidor de correo o con muchos otros tipos de servidores. Todo lo que se necesita es encontrar el documento que describe el protocolo y escribir el código para enviar y recibir los datos de acuerdo con el protocolo.

Sin embargo, dado que el protocolo que usamos con mayor frecuencia es el protocolo web HTTP, Python tiene una biblioteca especial diseñada específicamente para admitir el protocolo HTTP para la recuperación de documentos y datos a través de la web.

Copyright 2026

Recuperar una imagen sobre HTTP

En el ejemplo anterior, recuperamos un archivo de texto sin formato que tenía nuevas líneas en el archivo y simplemente copiamos los datos a la pantalla mientras se ejecutaba el programa. Podemos usar un programa similar para recuperar una imagen usando HTTP. En lugar de copiar los datos en la pantalla a medida que se ejecuta el programa, acumulamos los datos en una cadena, recortamos los encabezados y luego guardamos los datos de la imagen en un archivo de la siguiente manera:

https://trinket.io/embed/python3/ba6a2ee376

Cuando el programa se ejecuta produce el siguiente resultado:

```
$ python urljpeg.py
2920 2920
1460 4380
1460 5840
1460 7300
...
1460 62780
1460 64240
2920 67160
1460 68620
1681 70301
Header length 240
HTTP/1.1 200 OK
Date: Sat, 02 Nov 2013 02:15:07 GMT
Server: Apache
Last-Modified: Sat, 02 Nov 2013 02:01:26 GMT
ETag: "19c141-111a9-4ea280f8354b8"
Accept-Ranges: bytes
Content-Length: 70057
Connection: close
Content-Type: image/jpeg
```

Copyright 2026

Puedes ver que para esta URL, el encabezado `Content-Type` indica que el cuerpo del documento es una imagen (`image/jpeg`). Una vez que el programa se completa, puedes ver los datos de la imagen abriendo el archivo `stuff.jpg` en un visor de imágenes.

A medida que se ejecuta el programa, puedes ver que no obtenemos 5120 caracteres cada vez que llamamos al método `recv()`. Recibimos tantos caracteres como nos ha sido transferido a través de la red por el servidor web en el momento que llamamos `recv()`. En este ejemplo, obtenemos 1460 o 2920 caracteres cada vez que solicitamos hasta 5120 caracteres de datos.

Sus resultados pueden ser diferentes dependiendo de la velocidad de su red. También ten en cuenta que en la última llamada a `recv()` obtenemos 1681 bytes, que es el final de la secuencia, y en la siguiente llamada a `recv()` obtenemos una cadena de longitud cero que nos dice que el servidor ha llamado a `close()` en su extremo del socket y no hay más datos por venir.

Podemos ralentizar nuestras llamadas sucesivas `recv()` sin comentar la llamada a `time.sleep()`. De esta manera, esperamos un cuarto de segundo después de cada llamada para que el servidor pueda "adelantarse" y enviarnos más datos antes de que llamemos a `recv()` nuevamente. Con el retraso, en su lugar el programa se ejecuta de la siguiente manera:

```
$ python urljpeg.py
1460 1460
5120 6580
5120 11700
...
5120 62900
5120 68020
2281 70301
Header length 240
HTTP/1.1 200 OK
Date: Sat, 02 Nov 2013 02:22:04 GMT
Server: Apache
Last-Modified: Sat, 02 Nov 2013 02:01:26 GMT
ETag: "19c141-111a9-4ea280f8354b8"
Accept-Ranges: bytes
Content-Length: 70057
Connection: close
Content-Type: image/jpeg
```

Ahora, aparte de las primeras y últimas llamadas a `recv()`, ahora obtenemos 5120 caracteres cada vez que pedimos nuevos datos.

Hay un búfer entre el servidor que realiza las solicitudes `send()` y nuestra aplicación que realiza las solicitudes `recv()`. Cuando ejecutamos el programa con tiempos de espera, en algún momento el servidor podría llenar el búfer en el socket y ser forzado a pausar hasta que nuestro programa comience a vaciar el búfer. La pausa de la aplicación de envío o la aplicación de recepción se denomina "control de flujo".

Recuperar páginas web con `urllib`

Si bien podemos enviar y recibir datos manualmente a través de HTTP utilizando la biblioteca de sockets, hay una forma mucho más sencilla de realizar esta tarea común en Python utilizando la biblioteca `urllib`.

Usando `urllib`, puedes tratar una página web como un archivo. Simplemente indique qué página web deseas recuperar y `urllib` se encarga de todos los detalles del protocolo HTTP y del encabezado.

El código equivalente para leer el archivo `romeo.txt` de la web usando`urllib` es el siguiente:

https://trinket.io/embed/python3/8ead88f3fa

Una vez que la página web se ha abierto con `urllib.urlopen`, podemos tratarla como un archivo y leerla utilizando un bucle `for`.

Cuando el programa se ejecuta, solo vemos la salida del contenido del archivo. Los encabezados aún se envían, pero el código `urllib` consume los encabezados y solo nos devuelve los datos.

```
But soft what light through yonder window breaks
It is the east and Juliet is the sun
Arise fair sun and kill the envious moon
Who is already sick and pale with grief
```

Como ejemplo, podemos escribir un programa para recuperar los datos para `romeo.txt` y calcular la frecuencia de cada palabra en el archivo de la siguiente manera:

https://trinket.io/embed/python3/6669c56892

Nuevamente, una vez que hemos abierto la página web, podemos leerla como un archivo local.

Analizando HTML y raspando la web

Uno de los usos comunes de la capacidad `urllib` en Python es **raspar** la web. El raspado web (web scraping) tiene lugar cuando escribimos un programa que pretende ser un navegador web y recupera páginas, luego examina los datos en esas páginas en busca de patrones.

Como ejemplo, un motor de búsqueda como Google buscará en el origen de una página web y extraerá los enlaces a otras páginas y recuperará esas páginas, extrayendo enlaces, etc. Usando esta técnica, Google Spider se abre paso a través de casi todas las páginas de la web.

Google también utiliza la frecuencia de los enlaces de las páginas que encuentra a una página en particular como una medida de cuán "importante" es una página y qué tan alta debe aparecer la página en los resultados de búsqueda.

Análisis HTML usando expresiones regulares

Una forma sencilla de analizar HTML es usar expresiones regulares para buscar y extraer subcadenas que coincidan con un patrón en particular.

Aquí hay una simple página web:

```
<h1>The First Page</h1>
<p>
If you like, you can switch to the
<a href="http://www.dr-chuck.com/page2.htm">
```

```
Second Page</a>.
</p>
```

Podemos construir una expresión regular bien formada para que coincida y extraiga los valores de enlace del texto anterior de la siguiente manera:

```
href="http://.+?"
```

Nuestra expresión regular busca cadenas que comiencen con `href="http://`, seguidas de uno o más caracteres (".+?"), seguidas de otra comilla doble. El signo de interrogación ".+? " indica que la coincidencia debe realizarse de manera "no codiciosa" en lugar de "codiciosa". Una coincidencia no codiciosa trata de encontrar la cadena de coincidencia **más pequeña** posible y una coincidencia codiciosa trata de encontrar la **más grande** posible cadena coincidente.

Agregamos paréntesis a nuestra expresión regular para indicar qué parte de nuestra cadena coincidente nos gustaría extraer y producir el siguiente programa:

https://trinket.io/embed/python3/436b82b9ea

El método de expresión regular `findall` nos dará una lista de todas las cadenas que coinciden con nuestra expresión regular, devolviendo solo el texto del enlace entre las comillas dobles.

Cuando ejecutamos el programa, obtenemos el siguiente resultado:

```
python urlregex.py
Enter - http://www.dr-chuck.com/page1.htm
http://www.dr-chuck.com/page2.htm
```

```
python urlregex.py
Enter - http://www.py4e.com/book.htm
http://www.greenteapress.com/thinkpython/thinkpython.html
http://allendowney.com/
http://www.py4e.com/code
http://www.lib.umich.edu/espresso-book-machine
http://www.py4e.com/py4inf-slides.zip
```

Las expresiones regulares funcionan muy bien cuando tu HTML está bien formateado y es predecible. Pero como hay muchas páginas HTML "rotas" por ahí, una solución que solo use expresiones regulares podría perder algunos enlaces válidos o terminar con datos erróneos.

Esto se puede resolver utilizando una biblioteca de análisis HTML robusta.

Analizando HTML usando BeautifulSoup

Hay una serie de bibliotecas de Python que pueden ayudarte a analizar HTML y extraer datos de las páginas. Cada una de las bibliotecas tiene sus fortalezas y debilidades y puedes elegir una según tus necesidades.

Como ejemplo, simplemente analizaremos algunas entradas de HTML y extraeremos enlaces utilizando la biblioteca **BeautifulSoup**. Puedes descargar e instalar el código de BeautifulSoup abriendo una terminal y escribiendo `pip install bs4`

A pesar de que el HTML se parece a XML [1] y algunas páginas se construyen cuidadosamente para ser XML, la mayoría de HTML generalmente se rompe de manera que un analizador XML rechaza la página completa de HTML como incorrectamente formado. BeautifulSoup tolera HTML muy defectuoso y aún le permite extraer fácilmente los datos que necesita.

Usaremos `urllib` para leer la página y luego usaremos `BeautifulSoup` para extraer los atributos `href` de las etiquetas de anclaje (`a`).

https://trinket.io/embed/python3/d64a07806f

El programa solicita una dirección web, luego abre la página web, lee los datos y pasa los datos al analizador BeautifulSoup, y luego recupera todas las etiquetas de anclaje e imprime el atributo `href` para cada etiqueta.

Cuando el programa se ejecuta se ve como sigue:

```
python urllinks.py
Enter - http://www.dr-chuck.com/page1.htm
```

```
http://www.dr-chuck.com/page2.htm
```

```
python urllinks.py
Enter - http://www.py4e.com/book.htm
http://www.greenteapress.com/thinkpython/thinkpython.html
http://allendowney.com/
http://www.si502.com/
http://www.lib.umich.edu/espresso-book-machine
http://www.py4e.com/code
http://www.py4e.com/
```

Puedes usar BeautifulSoup para extraer varias partes de cada etiqueta de la siguiente manera:

https://trinket.io/embed/python3/0d6e91e4d7

```
python urllink2.py
Enter - http://www.dr-chuck.com/page1.htm
TAG: <a href="http://www.dr-chuck.com/page2.htm">
Second Page</a>
URL: http://www.dr-chuck.com/page2.htm
Content: ['\nSecond Page']
Attrs: [('href', 'http://www.dr-chuck.com/page2.htm')]
```

Estos ejemplos solo comienzan a mostrar el poder de BeautifulSoup cuando se trata de analizar HTML.

Leyendo archivos binarios usando urllib

A veces deseas recuperar un archivo que no sea de texto (o binario), como un archivo de imagen o video. Los datos en estos archivos generalmente no son útiles para imprimir, pero puedes hacer fácilmente una copia de una URL a un archivo local en tu disco duro usando `urllib`.

El patrón es abrir la URL y usar `read` para descargar todo el contenido del documento en una variable de cadena (`img`) y luego escribir esa información en un archivo local de la siguiente manera:

https://trinket.io/embed/python3/5594c1233b

Este programa lee todos los datos de una sola vez a través de la red y los almacena en la variable `img` en la memoria principal de tu ordenador, luego abre el archivo `cover.jpg` y escribe los datos en tu disco. Esto funcionará si el tamaño del archivo es menor que el tamaño de la memoria de tu ordenador.

Sin embargo, si se trata de un gran archivo de audio o video, este programa puede fallar o, al menos, ejecutarse muy lentamente cuando el ordenador se queda sin memoria. Para evitar quedarse sin memoria, recuperamos los datos en bloques (o búferes) y luego escribimos cada bloque en tu disco antes de recuperar el siguiente bloque. De esta manera, el programa puede leer archivos de cualquier tamaño sin utilizar toda la memoria que tienes en tu ordenador.

https://trinket.io/embed/python3/a78adf3c8a

En este ejemplo, leemos solo 100,000 caracteres a la vez y luego escribimos esos caracteres en el archivo `cover.jpg` antes de recuperar los siguientes 100,000 caracteres de datos de la web.

Este programa se ejecuta de la siguiente manera:

```
python curl2.py
568248 characters copied.
```

Si tienes un ordenador Unix o Macintosh, es probable que tengas un comando integrado en tu sistema operativo que realice esta operación de la siguiente manera:

```
curl -O http://www.py4e.com/cover.jpg
```

El comando `curl` es la abreviatura de "copy URL", por lo que estos dos ejemplos se denominan `curl1.py` y `curl2.py` en www.py4e.com/code3 ya que implementan una funcionalidad similar a la del comando `curl`. También hay un programa de ejemplo `curl3.py` que realiza esta tarea con un poco más de eficacia, en caso de que realmente desees utilizar este patrón en un programa que

estés escribiendo.

Ejercicios

Ejercicio 1: Cambia el programa de socket `socket1.py` para solicitar al usuario la URL para que pueda leer cualquier página web. Puede usar `split('/')` para dividir la URL en sus componentes, de modo que puedas extraer el nombre de host para la llamada socket `connect`. Agrega la comprobación de errores utilizando `try` y `except` para manejar la condición en la que el usuario ingresa una URL con formato incorrecto o inexistente.

Ejercicio 2: Cambia tu programa de socket para que cuente la cantidad de caracteres que ha recibido y deje de mostrar cualquier texto después de haber mostrado 3000 caracteres. El programa debe recuperar todo el documento, contar el número total de caracteres y mostrar el recuento del número de caracteres al final del documento.

Ejercicio 3: Usa `urllib` para replicar el ejercicio anterior de (1) recuperar el documento de una URL, (2) mostrar hasta 3000 caracteres, y (3) contar el número total de caracteres en el documento. No te preocups por los encabezados de este ejercicio, simplemente muestra los primeros 3000 caracteres del contenido del documento.

Ejercicio 4: Cambia el programa `urllinks.py` para extraer y contar las etiquetas de párrafo (p) del documento HTML recuperado y mostrar el recuento de los párrafos como salida de tu programa. No mostrar el texto del párrafo, solo contarlos. Prueba tu programa en varias páginas web pequeñas, así como en algunas páginas web más grandes.

Ejercicio 5: (Avanzado) Cambia el programa de socket para que solo muestre datos después de que se hayan recibido los encabezados y una línea en blanco. Recuerda que `recv` está recibiendo caracteres (nuevas líneas y demás), no líneas.

> [1]. El formato XML se describe en el siguiente capítulo. ↩

Revision #1
Created 2025-04-05 11:14:40 CEST by Javier Quintana
Updated 2025-04-05 11:16:36 CEST by Javier Quintana