

# Resumen del capítulo: Recuperación de datos de recursos en línea

## ¿Qué es la minería web?

A veces, es posible que no recibas suficientes datos para realizar un análisis exhaustivo. En tales casos, es necesario hacer más investigaciones. Esto te permite tener en cuenta más factores, identificar más patrones y llegar a conclusiones inesperadas. Los analistas pueden enriquecer sus datos complementándolos con datos de Internet. Primero rastrean los recursos que pueden ser relevantes, luego recuperan todos los datos necesarios. Este proceso se llama minería web, o análisis sintáctico.

## Cosas que un analista necesita saber sobre Internet: navegadores, HTML y HTTP

Internet es una red de computadoras que intercambian datos. Tiene reglas para intercambiar y representar información en Internet (considerando la forma en que las computadoras la muestran). Para que esto suceda, se inventaron varias cosas:

1. Un lenguaje para crear documentos: **HTML**
2. Aplicaciones de software para ver dichos documentos: **navegadores**
3. Reglas generales para transferir documentos: **HTTP**

## Protocolos de transferencia

Para que el proceso de intercambio de datos funcione, es necesario que existan ciertas reglas que rijan cómo una computadora envía datos a otra. El intercambio de datos en Internet se basa en el principio de "solicitud-respuesta": un navegador genera una solicitud, luego el servidor la analiza y envía una respuesta. Las reglas para formular solicitudes y respuestas están determinadas por lo que se conoce como protocolo de transferencia, en este caso, HTTP.

La mayoría de los sitios web actuales utilizan un protocolo de transferencia de datos con seguridad mejorada llamado **HTTPS**. Este protocolo garantiza que toda la comunicación entre tu navegador y un sitio web esté encriptada.

Cuando accedes a un sitio web, tu navegador envía una solicitud HTTP al servidor. El servidor, a su vez, formula una respuesta: el código HTML de la página correspondiente. Una solicitud formada por un navegador puede contener lo siguiente:

- Un método HTTP: determina la operación que se debe realizar. Existen varios métodos y los más populares son GET y POST. El primero solicita datos del servidor, mientras que el segundo los envía.
- Ruta: el segmento de la dirección que sigue al nombre del sitio (en [example.com/hello](http://example.com/hello), la ruta es /hello).
- La versión del protocolo HTTP utilizado para enviar la solicitud (por ejemplo, HTTP/1.1).
- Encabezados de solicitud, que se utilizan para enviar información adicional al servidor.
- Solicitar cuerpo. Por ejemplo, el cuerpo de una solicitud POST son los datos que se envían. No todas las solicitudes tienen cuerpo.

La respuesta puede contener:

- La versión HTTP.
- El código y el mensaje de la respuesta (por ejemplo, "200 OK" si todo va bien, o "404 Not Found" si no se encuentra la ruta solicitada).
- Encabezados que contienen información adicional para el navegador.
- Cuerpo de la respuesta (por ejemplo, cuando abres el sitio web, verás el código HTML de esta página en el cuerpo de la respuesta).

## Introducción a HTML

Para exportar una lista de productos del sitio web de una tienda virtual, primero deberás obtener el código de la página y su contenido.

Con HTML, cada objeto de la página debe marcarse para que se muestre correctamente. Este marcado implica colocar bloques de información dentro de

comandos llamados "etiquetas". Estas etiquetas le dicen a los navegadores cómo mostrar la información que rodean.

Un elemento HTML se compone de etiquetas y el contenido que está dentro de ellas. Una etiqueta HTML consta de un nombre rodeado de corchetes angulares. Un elemento comienza con una etiqueta de apertura con el nombre de la etiqueta y termina con una etiqueta de cierre con una barra y el nombre de la etiqueta. Se hace referencia al elemento por el nombre de la etiqueta.

Esta es la estructura típica de una página HTML:

## 1. `<html> ... </html>`

La etiqueta `<html>` presenta cada documento HTML e indica su comienzo, mientras que `</html>` marca su final. El `<head>` y el `<body>` del documento HTML se encuentran entre estas etiquetas.

## 2. `<head> ... </head>`

Estas etiquetas marcan el encabezado del documento. Las etiquetas que introducen el título del documento (`<title>`) y la metainformación (adicional) (`<meta>`) se colocan entre estas etiquetas.

## 3. `<body> ... </body>`

La etiqueta `<body>` marca el comienzo del cuerpo de una página HTML.

Todo el contenido de la página (títulos, párrafos de texto, tablas, imágenes) se coloca dentro del cuerpo.

Para que el marcado sea más claro, los desarrolladores dejan comentarios dentro de etiquetas especiales `<!-- -->` en el código de la página.

Esto es muy útil para los analistas, así que dejaremos comentarios en código en nuestros ejemplos.

Los analistas a menudo analizan las tablas. Por lo general, se colocan en los elementos de la tabla, entre las etiquetas `<table>` y `</table>`. La etiqueta de apertura `<table>` marca el comienzo de una tabla, y la etiqueta de cierre `</table>` marca su final. Dentro de este elemento, el contenido de la tabla se divide en filas mediante etiquetas `<tr>` (fila de tabla), y las filas, a su vez, se dividen en celdas mediante etiquetas `<td>` (datos de tabla). La primera fila generalmente contiene encabezados de columna en

lugar de celdas normales. Se colocan entre las etiquetas `<th>` (encabezado de la tabla).

El texto a menudo se coloca dentro de un elemento `p` (párrafo). El comienzo del párrafo se marca con la etiqueta `<p>` y el final con la etiqueta `</p>`.

La etiqueta de bloque `<div>` (división), que puede envolver varios elementos, es bastante común. `div` es útil, ya que puede incorporar cualquier cantidad de elementos, incluso de diferentes tipos (por ejemplo, un encabezado con una imagen más un par de párrafos de texto), y asignarles características o comportamientos comunes.

También puedes colocar **atributos** entre corchetes para brindar más información sobre cómo debe comportarse el elemento. Diferentes tipos de información requieren diferentes atributos.

El nombre del atributo le dice al navegador a qué función se refiere el atributo, mientras que el valor especifica lo que debería suceder con la función. La mayoría de las veces, deberás trabajar con los atributos `id` y `class`. El atributo `id` proporciona un identificador único para un elemento. El valor del atributo `class` es un nombre que pueden compartir varios elementos, al igual que varios miembros de la familia comparten un apellido.

## Herramientas para desarrolladores

Todos los navegadores modernos tienen una barra de herramientas para desarrolladores web, una navaja suiza para cualquier desarrollador. Aquí puedes ver el código de una página completa o de un elemento en particular, ver el estilo de cada elemento en la página e incluso cambiar la forma en que se muestran en tu PC. Puedes acceder a él presionando `Control+Shift+i`.

## Tu primera solicitud GET

Para obtener datos del servidor, usaremos el método `get()`, y para enviar solicitudes HTTP, necesitamos la librería de solicitudes. Importamos la librería:

```
import requests
```

El método `get()` actúa como un navegador. Le pasaremos el enlace como argumento. El método enviará al servidor una solicitud GET, luego procesará la respuesta que

reciba y devolverá una respuesta, un objeto que contiene la respuesta del servidor a la solicitud.

```
req = requests.get(URL) # guardando el objeto de respuesta como variable requerida
```

Un objeto de respuesta contiene la respuesta del servidor: el código de estado, el contenido de la solicitud y el código de la propia página HTML. Los atributos de los objetos de respuesta hacen posible obtener solo los datos relevantes del servidor. Por ejemplo, un objeto de respuesta con el atributo de texto solo devolverá el contenido de texto de la solicitud:




```
print(req.text) # el nombre del atributo se coloca después del objeto de respuesta y se divide por un punto
```

El atributo *status\_code* te dice si el servidor respondió o si ocurrió un error.

```
print(req.status_code)
```

Desafortunadamente, no todas las solicitudes regresan con datos. A veces, las solicitudes devuelven errores; cada uno tiene un código especial dependiendo de su tipo. Estos son los errores más comunes:

### Códigos de error

 Error code	 Name	 Implication
<u>200</u>	OK	Todo está bien
<u>302</u>	Found	La página ha sido movida
<u>400</u>	Bad Request	Error en la sintaxis de la solicitud
<u>404</u>	Not Found	No se puede encontrar la página
<u>500</u>	Internal Server Error	Error por parte del servidor
<u>502</u>	Bad Gateway	Error en el intercambio de datos entre servidores
<u>503</u>	Server Unavailable	El servidor no puede procesar solicitudes temporalmente

# Expresiones regulares

Para buscar cadenas en textos grandes, necesitarás una herramienta poderosa: las expresiones regulares. Una **expresión regular** es una regla para buscar subcadenas (fragmentos de texto dentro de cadenas). Es posible crear reglas complejas para que una expresión regular devuelva varias subcadenas.





Para comenzar a trabajar con expresiones regulares en Python, necesitamos importar el módulo `re` (es decir, expresiones regulares). Siguen dos etapas.





En la primera etapa, creamos el patrón de la expresión regular. Este es un algoritmo que describe lo que debe buscarse dentro del texto (por ejemplo, todas las letras mayúsculas).

Luego, este patrón se pasa a los métodos específicos de `re`. Estos métodos buscan, reemplazan y eliminan símbolos. En otras palabras, el patrón identifica qué buscar y cómo, mientras que el método define qué hacer con las coincidencias que se encuentran.

La siguiente tabla enumera los patrones de expresiones regulares más simples. Puedes crear expresiones regulares más complejas combinándolos.

## Sintaxis de expresiones regulares

 Regular expression	 Description	 Example	 Explanation
<code>[]</code>	Caracter único contenido entre paréntesis	<code>[a-]</code>	a o -
<code>[^...]</code>	Negación	<code>[^a]</code>	cualquier caracter excepto «a»
<code>:</code>	Rango	<code>[0-9]</code>	rango: cualquier dígito del 0 al 9
<code>.</code>	Cualquier caracter único excepto una nueva línea	<code>a.</code>	as, a1, a_
<code>\d (see [0-9]).</code>	Cualquier dígito	<code>a\d   a[0-9]</code>	a1, a2, a3
<code>\w</code>	Cualquier letra, dígito o _	<code>a\w</code>	a_, a1, ab
<code>[A-z]</code>	Cualquier letra latina	<code>a[A-z]</code>	ab

 Regular expression	 Description	 Example	 Explanation
[А-я]	Cualquier letra cirílica	a[А-я]	ая
?	0 o 1 entrada	a?	a o nada
±	1 o más entradas	a±	a o aa, o aaa
*	0 o más entradas	a*	Nada o - a, o aa
^	Comienzo de cadena	^a	a1234, abcd
\$	Fin de cadena	a\$	1a, ba

Estas son algunas de las tareas más comunes para los analistas:

- encontrar una subcadena dentro de una cadena
- dividir cadenas en subcadenas
- reemplazar partes de una cadena con otras cadenas

Para completar estas tareas, necesitarás los siguientes métodos `re`:

1. **`search(pattern, string)`** busca un `pattern` en una `string`. Aunque `search()` recorre toda la cadena para encontrar el patrón, solo devuelve la primera subcadena que encuentra:

```
import re
print(re.search(pattern, string))
```

El método `search()` devuelve un objeto de tipo de coincidencia. El parámetro `span` define un rango de índices que coinciden con el patrón. El parámetro de coincidencia indica el valor de la propia subcadena.

Si no necesitamos información sobre el span, podemos devolver solo la subcadena usando el método `group()`:

```
import re
print(re.search(pattern, string).group())
```

2. **`split(pattern, string)`** rompe una `string` en los puntos donde aparece el patrón.

```
import re
print(re.split(pattern, string))
```

La cadena se divide donde se encuentra el patrón. El número de divisiones se puede controlar mediante el parámetro **maxsplit** del método `split()`.

```
import re
print(re.split(pattern, string, maxsplit = num_split))
```

3. **sub(pattern, repl, string)** busca el `pattern` subcadena dentro de una `string` y lo reemplaza con la subcadena repl (es decir, reemplazar).

```
import re
print(re.sub(pattern, repl, string))
```

4. **findall(pattern, string)** devuelve una lista de todas las subcadenas de una `string` que coinciden con el `pattern`. Compáralo con el método `search()` que solo devuelve la primera subcadena coincidente.

```
import re
print(re.findall(pattern, string))
```

El método `findall()` es particularmente útil porque permite determinar el número de subcadenas recurrentes en una cadena con la función `len()`:

```
import re
print(len(re.findall(pattern, string)))
```

## Análisis sintáctico de HTML

La extracción manual de valores de datos puros de una cadena que contiene el código de una página puede ser difícil. Para resolver esto necesitamos la librería BeautifulSoup. Los métodos de la librería BeautifulSoup convierten un archivo HTML en



una estructura de árbol. Luego, el contenido necesario se puede encontrar mediante etiquetas y atributos.

```
from bs4 import BeautifulSoup
soup = BeautifulSoup(req.text, 'lxml')
```

El primer argumento presenta datos que formarán una estructura de árbol. El segundo argumento es un analizador sintáctico. Este define la forma en que una página web entra en el árbol. Existen numerosos analizadores, y todos crean diferentes estructuras a partir de un mismo documento HTML. Hemos elegido el analizador `lxml` por su rendimiento de alta velocidad. Pero, por supuesto, existen otros analizadores, como `html.parser`, `xml` y `html5lib`.

Una vez que el código se convierte en una estructura de árbol, los datos se pueden buscar utilizando varios métodos. El primer método de búsqueda se llama **`find()`**. Se ejecuta a través de un documento HTML y encuentra el primer elemento cuyo nombre se pasó como argumento y lo devuelve junto con las etiquetas y su contenido.

```
tag_content = soup.find(tag)
```

Para mostrar el contenido sin etiquetas, necesitarás el método **`text`**. Este devolverá el resultado en forma de cadena:

```
tag_content.text
```

Todavía hay otro método de búsqueda, que es **`find_all()`**. A diferencia del método anterior, **`find_all()`** encuentra todas las instancias de un elemento dado en un documento HTML y devuelve una lista:

```
tag_content = soup.find_all(tag)
```

Extraigamos solo el contenido de los párrafos con la ayuda del método de `text`:

```
for tag_content in soup.find_all(tag):  
    print(tag_content.text)
```

Los métodos `find()` y `find_all()` tienen un filtro extra para buscar elementos de página: el parámetro **attrs** (atributos). Se utiliza para buscar por clases e identificadores. Sus nombres se especifican en el panel de herramientas para desarrolladores web.

Debes pasar a `attrs` un diccionario con nombres y valores de atributos:

```
soup.find(tag, attrs={"attr_name": "attr_value"})
```

## API

A veces es necesario solicitar información de fuentes externas cuya estructura es mucho más complicada que la de una página HTML ordinaria. Para evitar tener que estudiar su estructura para obtener datos más rápido, los analistas envían solicitudes GET a aplicaciones de terceros a través de una interfaz de transferencia de datos especial llamada **API** (interfaz de programación de aplicaciones).

La librería `requests` permite pasar parámetros a una URL. Cuando buscas cierto contenido en un sitio web de varias páginas, debes pasar el diccionario `PARAM` a la palabra clave **params** (parámetros). Por ejemplo:

```
URL = 'https://yandex.com/'  
PARAM={"page": "4"}  
req = requests.get(url = URL, params = PARAM)
```

Esta solicitud debe devolver la cuarta página (según el catálogo) del sitio web <https://yandex.com/>.

## JSON

Al responder a tu solicitud, el servidor devuelve datos estructurados en uno de varios formatos especiales, de los cuales el más común es JSON (Notación de objetos de

JavaScript). Parece un revoltijo de dígitos, letras, dos puntos y llaves.

Así es como se ven los datos en este formato:

```
[
  {
    "name": "General Slocum",
    "date": "Junio 15, 1904"
  },
  {
    "name": "Camorta",
    "date": "Mayo 6, 1902"
  },
  {
    "name": "Norge",
    "date": "Junio 28, 1904"
  }
]
```

Si JSON incluye varios elementos, estos se escriben entre corchetes `[ ... ]`, al igual que en las listas. Un objeto JSON individual parece un diccionario: está delimitado por corchetes y tiene pares `clave : valor`.

JSON permite recopilar datos dentro de un objeto (una lista de pares `clave : valor`) y luego hacer una cadena para pasar en una solicitud. El receptor vuelve a convertir esta cadena en un objeto.

Python tiene un módulo incorporado para trabajar con datos en formato JSON:

```
import json
```

Su método `json.loads()` convierte cadenas que están en formato JSON:

```
x = '{"Nombre": "General Slocum", "fecha": "Junio 15, 1904"}'
y = json.loads(x)

print('Nombre : {0}, fecha : {1}'.format(y['Nombre'], y['fecha']))
```

```
# Respuesta
Nombre : General Slocum, fecha : Junio 15, 1904
```