11 Regex

Hasta ahora hemos estado leyendo archivos, buscando patrones y extrayendo varios trozos de líneas que nos parecen interesantes. Hemos ido utilizando métodos de cadena como `split` y `find`, utilizando listas y troceado de cadenas para extraer partes de las líneas.

Esta tarea de búsqueda y extracción es tan común que Python tiene una biblioteca muy poderosa llamada **expresiones regulares** que maneja muchas de estas tareas de manera muy elegante. La razón por la que no hemos introducido expresiones regulares anteriormente en el libro es que, si bien son muy potentes, son un poco complicadas y su sintaxis requiere un tiempo para acostumbrarse.

Las expresiones regulares son casi su propio lenguaje de programación para buscar y analizar cadenas. De hecho, se han escrito libros completos sobre el tema de las expresiones regulares. En este capítulo, solo cubriremos los conceptos básicos de las expresiones regulares. Aquí tienes más detalles sobre expresiones regulares. También en la documentación de Python:

https://docs.python.org/3.7/library/re.html

La biblioteca de expresiones regulares, `re`, debe importarse en tu programa antes de poder usarla. El uso más simple de la biblioteca de expresiones regulares es la función `search()`. El siguiente programa demuestra un uso trivial de la función de búsqueda.

```
# Search for lines that contain 'From'
import re
hand = open('mbox-short.txt')
for line in hand:
    line = line.rstrip()
    if re.search('From:', line):
        print(line)

# Code: http://www.py4e.com/code3/re01.py
```

Abrimos el archivo, recorremos cada línea y usamos la función `search()` para imprimir solo las líneas que contienen la cadena "From:". Este programa no usa el poder real de las expresiones regulares, ya que podríamos haber utilizado `line.find()` con la misma facilidad para lograr el

Copyright 2026

mismo resultado.

El poder de las expresiones regulares llega cuando agregamos caracteres especiales a la cadena de búsqueda que nos permiten controlar con mayor precisión qué líneas coinciden con la cadena. Agregar estos caracteres especiales a nuestra expresión regular nos permite hacer una comparación y extracción sofisticadas mientras escribimos muy poco código.

Por ejemplo, el carácter `^` se usa en expresiones regulares para coincidir con "el principio" de una línea. Podríamos cambiar nuestro programa para que solo coincida con las líneas donde "De:" está al principio de la línea de la siguiente manera:

https://trinket.io/embed/python3/77c2350d8b

Ahora solo haremos coincidir las líneas que **comiencen con** la cadena "De:". Este es un ejemplo muy simple que podríamos haber hecho de manera equivalente con el método `startswith()` de la biblioteca de cadenas. Pero sirve para introducir la idea de que las expresiones regulares contienen caracteres de acción especiales que nos dan más control en cuanto a qué coincidirá con la expresión regular.

Coincidencia de caracteres en expresiones regulares

Hay una serie de caracteres especiales que nos permiten construir expresiones regulares aún más potentes. El carácter especial más utilizado es el punto, que coincide con cualquier carácter.

En el siguiente ejemplo, la expresión regular "F..m:" coincidiría con cualquiera de las cadenas "From:", "Fxxm:", "F12m:" o "F!@M:".

https://trinket.io/embed/python3/d558453717

Esto es particularmente poderoso cuando se combina con la capacidad de indicar que un caracter puede repetirse cualquier cantidad de veces utilizando los caracteres "*" o "+" en su expresión regular. Estos caracteres especiales significan que, en lugar de coincidir con un solo caracter en la cadena de búsqueda, coinciden con cero o más caracteres (en el caso del asterisco), o uno o más

caracteres (en el caso del signo más) .

Podemos restringir aún más las líneas 'interesantes' utilizando el caracter repetido de **comodín** (wildcard) en el siguiente ejemplo:

https://trinket.io/embed/python3/e82ead95e5

La cadena de búsqueda `^From:.+@` coincidirá con las líneas que comienzan con "From:", seguido de uno o más caracteres (". +"), Seguido de un signo de at. Así que esto coincidirá con la siguiente línea:

```
From: uct.ac.za
```

Puede pensar que el comodín ".+" se expande para que coincida con todos los caracteres entre el carácter de dos puntos y la arroba.

Es bueno saber que los caracteres de más y asterisco son "agresivos". Por ejemplo, la siguiente cadena coincidiría con la última arroba en la cadena cuando el ".+" Empuja hacia afuera, como se muestra a continuación:

```
From: iupui.edu
```

Es posible decirle a un asterisco o signo más que no sea tan "codicioso" al agregar otro personaje.

Consulte la documentación detallada para obtener información sobre cómo desactivar el comportamiento codicioso.

Extracción de datos usando expresiones regulares

Si queremos extraer datos de una cadena en Python, podemos usar el método `findall()` para extraer todas las subcadenas que coinciden con una expresión regular. Usemos el ejemplo de querer extraer todo lo que parece una dirección de correo electrónico desde cualquier línea, independientemente del formato. Por ejemplo, queremos extraer las direcciones de correo electrónico de cada una de las siguientes líneas:

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
Return-Path: <postmaster@collab.sakaiproject.org>
          for <source@collab.sakaiproject.org>;
Received: (from apache@localhost)
Author: stephen.marquard@uct.ac.za
```

No queremos escribir código para cada uno de los tipos de líneas, dividir y dividir de manera diferente para cada línea. Este programa siguiente utiliza `findall()` para encontrar las líneas con direcciones de correo electrónico en ellas y extraer una o más direcciones de cada una de esas líneas.

https://trinket.io/embed/python3/e8248e2794

El método `findall()` busca la cadena en el segundo argumento y devuelve una lista de todas las cadenas que parecen direcciones de correo electrónico. Estamos utilizando una secuencia de dos caracteres que coincide con uno o más caracteres que no sean un espacio en blanco (`\S`).

La salida del programa sería:

```
['csev@umich.edu', 'cwen@iupui.edu']
```

Al traducir la expresión regular, estamos buscando subcadenas que tengan al menos un carácter que no sea un espacio en blanco, seguido de un signo `@`, seguido de al menos un carácter más que no sea un espacio en blanco. La "`\S+`" coincide con la mayor cantidad posible de caracteres que no sean espacios en blanco.

La expresión regular coincidiría dos veces (csev@umich.edu y cwen@iupui.edu), pero no coincidiría con la cadena "@2PM" porque no hay caracteres que no estén en blanco **antes** del signo de inicio. Podemos usar esta expresión regular en un programa para leer todas las líneas de un archivo e imprimir cualquier cosa que parezca una dirección de correo electrónico de la siguiente manera:

https://trinket.io/embed/python3/674fe79c92

Leemos cada línea y luego extraemos todas las subcadenas que coinciden con nuestra expresión regular. Como `findall()` devuelve una lista, simplemente verificamos si el número de elementos en nuestra lista devuelta es más que cero para imprimir solo las líneas donde encontramos al menos una subcadena que parece una dirección de correo electrónico.

Si ejecutamos el programa en `mbox.txt` obtenemos el siguiente resultado:

```
['wagnermr@iupui.edu']
['cwen@iupui.edu']
['<postmaster@collab.sakaiproject.org>']
['<200801032122.m03LMFo4005148@nakamura.uits.iupui.edu>']
['<source@collab.sakaiproject.org>;']
['<source@collab.sakaiproject.org>;']
['<source@collab.sakaiproject.org>;']
['apache@localhost)']
['source@collab.sakaiproject.org;']
```

Algunas de nuestras direcciones de correo electrónico tienen caracteres incorrectos como "`<`" o ";" al principio o al final. Declaremos que solo nos interesa la parte de la cadena que comienza y termina con una letra o un número.

Para ello, utilizamos otra característica de las expresiones regulares. Los corchetes se utilizan para indicar un conjunto de múltiples caracteres aceptables que estamos dispuestos a considerar que coincidan. En cierto sentido, el "`\S`" está pidiendo que coincida con el conjunto de "caracteres que no son espacios en blanco". Ahora vamos a ser un poco más explícitos en términos de los caracteres con los que coincidiremos.

Aquí está nuestra nueva expresión regular:

```
[a-zA-Z0-9]\S*@\S*[a-zA-Z]
```

Esto se está complicando un poco y puedes empezar a ver porqué las expresiones regulares son un lenguaje en sí mismas. Al traducir esta expresión regular, vemos que estamos buscando subcadenas que comiencen con una sola minúscula, mayúscula o número "[a-zA-Z0-9]", seguido de cero o más caracteres que no estén en blanco ("`\S*`"), seguido de un signo arroba, seguido de cero o más caracteres que no estén en blanco, seguido de una letra mayúscula o minúscula. Ten en cuenta que cambiamos de "+" a `*` para indicar cero o más caracteres que no están en blanco ya que "[a-zA-Z0-9]" ya es un carácter que no está en blanco. Recuerda que el `*` o "+" se aplica al carácter único inmediatamente a la izquierda del signo más o asterisco.

Si usamos esta expresión en nuestro programa, nuestros datos son mucho más limpios:

https://trinket.io/embed/python3/1b176f4d0e

Copyright 2026

```
...
['wagnermr@iupui.edu']
['cwen@iupui.edu']
['postmaster@collab.sakaiproject.org']
['200801032122.m03LMFo4005148@nakamura.uits.iupui.edu']
['source@collab.sakaiproject.org']
['source@collab.sakaiproject.org']
['source@collab.sakaiproject.org']
['apache@localhost']
```

Observa que en las líneas "source@collab.sakaiproject.org", nuestra expresión regular eliminó dos letras al final de la cadena (">;"). Esto se debe a que cuando agregamos "[a-zA-Z]" al final de nuestra expresión regular, estamos exigiendo que cualquier cadena que el analizador de expresiones regulares encuentre debe terminar con una letra. Entonces, cuando ve ">" después de "sakaiproject.org> simplemente se detiene en la última letra "coincidente" que encontró (es decir, la "g" fue la última buena coincidencia).

También tenga en cuenta que la salida del programa es una lista de Python que tiene una cadena como elemento único en la lista.

Combinando búsqueda y extracción

Si queremos encontrar números en las líneas que comienzan con la cadena "X-", como por ejemplo:

```
X-DSPAM-Confidence: 0.8475
X-DSPAM-Probability: 0.0000
```

No solo queremos números de punto flotante de cualquier línea. Solo queremos extraer números de líneas que tengan la sintaxis anterior.

Podemos construir la siguiente expresión regular para seleccionar las líneas:

```
^X-.*: [0-9.]+
```

Tras traducir esto, estamos diciendo que queremos líneas que comiencen con "X-", seguidas de cero o más caracteres (".*"), Seguidas de dos puntos (":") y luego un espacio. Después del espacio, estamos buscando uno o más caracteres que sean un dígito (0-9) o un punto "[0-9.]+". Ten en cuenta que dentro de los corchetes, el período coincide con un período real (es decir, no es un comodín entre los corchetes).

Esta es una expresión muy estricta que coincidirá en gran medida solo con las líneas que nos interesan de la siguiente manera:

https://trinket.io/embed/python3/18870dcf26

Cuando ejecutamos el programa, vemos que los datos se filtran bien para mostrar solo las líneas que estamos buscando.

```
X-DSPAM-Confidence: 0.8475
X-DSPAM-Probability: 0.0000
X-DSPAM-Confidence: 0.6178
X-DSPAM-Probability: 0.0000
```

Pero ahora tenemos que resolver el problema de extraer los números. Si bien sería lo suficientemente simple como para usar `split`, podemos usar otra característica de las expresiones regulares para buscar y analizar la línea al mismo tiempo.

Los paréntesis son otro carácter especial en las expresiones regulares. Cuando agrega paréntesis a una expresión regular, se ignoran cuando coinciden con la cadena. Pero cuando está utilizando `findall()`, los paréntesis indican que aunque desea que la expresión completa coincida, solo le interesa extraer una parte de la subcadena que coincida con la expresión regular.

Entonces hacemos el siguiente cambio a nuestro programa:

https://trinket.io/embed/python3/38a31c1ce9

En lugar de llamar a `search()`, agregamos paréntesis a la parte de la expresión regular que representa el número de punto flotante para indicar que solo queremos que `findall()` nos devuelva la parte del número de punto flotante de la cadena correspondiente .

La salida de este programa es la siguiente:

```
['0.8475']
['0.0000']
['0.6178']
['0.0000']
['0.6961']
['0.0000']
..
```

Los números aún están en una lista y deben convertirse de cadenas a puntos flotantes, pero hemos utilizado el poder de las expresiones regulares para buscar y extraer la información que encontramos interesante.

Como otro ejemplo de esta técnica, si observas el archivo, hay varias líneas del formulario:

```
Details: http://source.sakaiproject.org/viewsvn/?view=rev&amp;rev=39772
```

Si quisiéramos extraer todos los números de revisión (el número entero al final de estas líneas) utilizando la misma técnica anterior, podríamos escribir el siguiente programa:

https://trinket.io/embed/python3/93a830d48e

Al traducir nuestra expresión regular, buscamos líneas que comiencen con "Details:", seguidas de cualquier número de caracteres (".*"), seguidas de "rev=" y luego con uno o más dígitos. Queremos encontrar líneas que coincidan con la expresión completa, pero solo queremos extraer el número entero al final de la línea, por lo que rodeamos "[0-9]+" con paréntesis.

Cuando ejecutamos el programa, obtenemos el siguiente resultado:

```
['39772']
['39771']
['39770']
['39769']
...
```

Recuerde que el "[0-9] +" es "codicioso" y trata de hacer una cadena de dígitos tan grande como sea posible antes de extraer esos dígitos. Este comportamiento "codicioso" es la razón por la que obtenemos los cinco dígitos para cada número. La biblioteca de expresiones regulares se expande

Copyright 2026

en ambas direcciones hasta que encuentra un no dígito, o el principio o el final de una línea.

Ahora podemos usar expresiones regulares para rehacer un ejercicio anterior en el libro en el que estábamos interesados en la hora del día de cada mensaje de correo. Buscamos líneas con la forma siguiente:

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
```

Y queríamos extraer la hora del día para cada línea. Anteriormente hicimos esto con dos llamadas a `split`. Primero, la línea se dividió en palabras y luego sacamos la quinta palabra y la dividimos nuevamente en el caracter de dos puntos para sacar los dos caracteres que nos interesaban.

Si bien esto funcionó, en realidad dio como resultado código bastante frágil que asumía que las líneas estaban bien formateadas. Si tuvieras que agregar suficientes comprobaciones de errores (o un bloque grande de try/except) para asegurarte de que tu programa nunca fallara cuando se le presentaran líneas de formato incorrecto, el código se inflaría a 10-15 líneas de código bastante difícil de leer.

Podemos hacer esto de una manera mucho más simple con la siguiente expresión regular:

```
^From .* [0-9][0-9]:
```

La traducción de esta expresión regular es que estamos buscando líneas que comiencen con "From " (nota el espacio), seguidas de cualquier número de caracteres (".*"), seguidas de un espacio, seguidas de dos dígitos `[0 -9][0-9]`, seguido de un caracter de dos puntos. Esta es la definición de los tipos de líneas que estamos buscando.

Para extraer solo la hora usando `findall()`, agregamos paréntesis alrededor de los dos dígitos de la siguiente manera:

```
^From .* ([0-9][0-9]):
```

Esto resulta en el siguiente programa:

https://trinket.io/embed/python3/6f58c712ee

Cuando el programa se ejecuta, produce el siguiente resultado:

```
['09']
['18']
['16']
['15']
...
```

carácter de escape

Ya que usamos caracteres especiales en expresiones regulares para hacer coincidir el comienzo o el final de una línea o especificar comodines, necesitamos una forma de indicar que estos caracteres son "normales" y queremos que coincidan con el caracter real, como un signo de dólar o un acento circunflejo.

Podemos indicar que queremos simplemente hacer coincidir un caracter precediendo a ese caracter con una barra invertida. Por ejemplo, podemos encontrar cantidades de dinero con la siguiente expresión regular.

```
import re
x = 'We just received $10.00 for cookies.'
y = re.findall('\$[0-9.]+',x)
```

Como prefijamos el signo de dólar con una barra invertida, en realidad coincide con el signo de dólar en la cadena de entrada en lugar de coincidir con el "fin de línea", y el resto de la expresión regular coincide con uno o más dígitos o el carácter de período. **Nota:** Dentro de los corchetes, los caracteres no son "especiales". Entonces cuando decimos "[0-9.]", Realmente significa dígitos o un punto. Fuera de los corchetes, un punto es el carácter de "comodín" y coincide con cualquier caracter. Dentro de los corchetes, el período es un período.

Resumen

Si bien esto solo rozó la superficie de las expresiones regulares, hemos aprendido un poco sobre el lenguaje de las expresiones regulares. Son cadenas de búsqueda con caracteres especiales que comunican sus deseos al sistema de expresión regular en cuanto a qué define "coincidencia" y qué se extrae de las cadenas coincidentes. Éstos son algunos de esos caracteres especiales y secuencias de caracteres:

`^` Coincide con el principio de la línea.

`$` Coincide con el final de la línea.

`.` Coincide con cualquier caracter (un comodín).

`\s` Coincide con un caracter de espacio en blanco.

`\S` Coincide con un caracter que no es un espacio en blanco (opuesto a `\s`).

`*` Se aplica al caracter inmediatamente anterior e indica que coincida con cero o más de los caracteres anteriores.

`*?` Se aplica al caracter inmediatamente anterior e indica que coincida con cero o más de los caracteres anteriores en "modo no codicioso".

`+` Se aplica al caracter inmediatamente anterior e indica que coincida con uno o más de los caracteres anteriores.

`+?` Se aplica al caracter inmediatamente anterior e indica que coincida con uno o más de los caracteres anteriores en "modo no codicioso".

`[aeiou]` Coincide con un solo caracter siempre que ese caracter esté en el conjunto especificado. En este ejemplo, coincidiría con "a", "e", "i", "o", o "u", pero no con otros caracteres.

`[a-z0-9]` Puedes especificar rangos de caracteres usando el signo menos. Este ejemplo es un solo caracter que debe ser una letra minúscula o un dígito.

`[^A-Za-z]` Cuando dentro de corchetes el primer caracter es un acento circunflejo, invierte la lógica. Este ejemplo coincide con un solo caracter que es cualquier cosa **distinta de** una letra mayúscula o minúscula.

`()` Cuando se agregan paréntesis a una expresión regular, se ignoran con el fin de hacer coincidir permitiendo extraer un subconjunto particular de la cadena coincidente en lugar de toda la cadena cuando se usa `findall()`.

`\b` Coincide con la cadena vacía, pero solo al principio o al final de una palabra.

`\B` Coincide con la cadena vacía, pero no al principio o al final de una palabra.

`\d` Coincide con cualquier dígito decimal; Equivalente al conjunto [0-9].

`\D` Coincide con cualquier caracter que no sea un dígito; equivalente al conjunto [`^` 0-9].

Copyright 2026

Sección extra para usuarios de Unix / Linux

El soporte para buscar archivos usando expresiones regulares se incorporó al sistema operativo Unix desde la década de 1960 y está disponible en casi todos los lenguajes de programación de una forma u otra.

De hecho, hay un programa de línea de comandos integrado en Unix llamado **grep** (Analizador de Expresión Regular Generalizada) que hace prácticamente lo mismo que los ejemplos de `search()` en este capítulo. Entonces, si tienes un sistema Macintosh o Linux, puedes probar los siguientes comandos en tu terminal.

```
$ grep '^From:' mbox-short.txt
From: stephen.marquard@uct.ac.za
From: louis@media.berkeley.edu
From: zqian@umich.edu
From: rjlowe@iupui.edu
```

Esto le dice a `grep` que le muestre las líneas que comienzan con la cadena "From:" en el archivo `mbox-short.txt`. Si experimentas con el comando `grep` un poco y lees la documentación de `grep`, encontrarás algunas diferencias sutiles entre el soporte de expresiones regulares en Python y el soporte de expresiones regulares en `grep`. Como ejemplo, `grep` no admite el caracter que no está en blanco `\S`, por lo que deberá usar la notación de conjuntos un poco más compleja `[^ ]`, que simplemente significa que coincida con un caracter que sea cualquier cosa Aparte de un espacio.

Depurando

Python tiene una documentación incorporada simple y rudimentaria que puede ser muy útil si necesitas una actualización rápida para activar tu memoria sobre el nombre exacto de un método en particular. Esta documentación se puede ver en el intérprete de Python en modo interactivo.

Puedes abrir un sistema de ayuda interactivo usando `help()`.

```
>>> help()
help> modules
```

Si sabes qué módulo deseas usar, puedes usar el comando `dir()` para encontrar los métodos en el módulo de la siguiente manera:

```
>>> import re
>>> dir(re)
['A', 'ASCII', 'DEBUG', 'DOTALL', 'I', 'IGNORECASE', 'L', 'LOCALE', 'M', 'MULTILINE',
'RegexFlag', 'S', 'Scanner', 'T', 'TEMPLATE', 'U', 'UNICODE', 'VERBOSE', 'X', '_MAXCACHE',
'__all__', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__',
'__package__', '__spec__', '__version__', '_alphanum_bytes', '_alphanum_str', '_cache',
'_compile', '_compile_repl', '_expand', '_locale', '_pattern_type', '_pickle', '_subx',
'compile', 'copyreg', 'enum', 'error', 'escape', 'findall', 'finditer', 'fullmatch',
'functools', 'match', 'purge', 'search', 'split', 'sre_compile', 'sre_parse', 'sub', 'subn',
'template']
>>>
```

También puedes obtener una pequeña cantidad de documentación sobre un método en particular usando el comando help.

```
>>> import re
>>> help(re.search)
```

La documentación incorporada no es muy extensa, pero puede ser útil cuando tienes prisa o no tienes acceso a un navegador web o motor de búsqueda.

Ejercicios

Ejercicio 1: Escribe un programa simple para simular el funcionamiento del comando `grep` en Unix. Pídele al usuario que ingrese una expresión regular y cuente el número de líneas que coincidieron con la expresión regular:

```
$ python grep.py
Enter a regular expression: ^Author
mbox.txt had 1798 lines that matched ^Author

$ python grep.py
Enter a regular expression: ^X-
```

```
mbox.txt had 14368 lines that matched ^X-

$ python grep.py
Enter a regular expression: java$
mbox.txt had 4218 lines that matched java$
```

Ejercicio 2: Escribe un programa para buscar líneas del formulario

```
`New Revision: 39772`
```

y extrae el número de cada una de las líneas utilizando una expresión regular y el método `findall()`. Calcula el promedio de los números e imprime el promedio.

```
Enter file:mbox.txt
38549.7949721

Enter file:mbox-short.txt
39756.9259259
```

Revision #1
Created 2025-04-05 11:12:41 CEST by Javier Quintana
Updated 2025-04-05 11:14:33 CEST by Javier Quintana