Usando expresiones regulares en Python

miércoles, 04 de enero de 2006

Desde el pan con nocilla, no se ha inventado mejor que las expresiones regulares. Una expresión regular es, básicamente, un patrón que describe a un cierto texto. Si has estudiado informática (no ordenadores, ni procesadores de texto) sabrás algo de las matemáticas (sí, la informática tiene algo que ver con las matemáticas) que hay detrás de todo esto. Sin embargo, aquí no voy hablar de eso, sino de cómo utilizar expresiones regulares en Python a través del módulo re.

Lo básico

Para empezar veamos un ejemplo sencillo. Puedes probar todo esto utilizando el intérprete interactivo de comandos:

```
>>> import re
>>> if (re.search("b", "abc")):
...     print "b está en abc"
b está en abc
```

El ejemplo es claro, se busca el patrón b en la cadena abc, utilizando la función search del módulo re. Veamos algo parecido utilizando una expresión un poco más complicada:

```
>>> if (re.search("\Aa[0-9].*(end|fin)$", "a7fin")):
...     print "se ha encontrado el patrón"
...
se ha encontrado el patrón
>>> if (re.search("\Aa[0-9].*(end|fin)$", "a2 lo que quiera fin")):
...     print "se ha encontrado el patrón"
...
se ha encontrado el patrón
```

En este caso, todas las cadenas que comienzen (**\A**) con a seguidas de un número (**[0-9]**), y de cualquier secuencia de caracteres (**.***) y terminadas (**$**) en end o fin (**(end|fin)**) encajarán con el patrón \Aa[0-9].*(end|fin)$. Sencillo ¿no?

En el ejemplo anterior hemos visto que pasamos como parámetro la expresión regular y la cadena en la que queremos buscar. Esto está bien si queremos aplicar la expresión regular una sola vez. Sin embargo, si vamos a utilizar la expresión regular varias veces, que es lo habitual, por motivos de eficiencia conviene compilar la expresión creando un objeto expresión regular. Veamos más ejemplos:

```
>>> pattern = re.compile ('H.*-HRV_.*-0000(19|2[01234])_.*$')
>>> name_files = [
      'H-000-MSG1__-MSG1________-HRV______-000019___-200601040930-C_',
...   'H-000-MSG1__-MSG1________-HRV______-000020___-200601040930-C_',
...   'H-000-MSG1__-MSG1________-HRV______-000021___-200601040930-C_',
...   'H-000-MSG1__-MSG1________-HRV______-000022___-200601040930-C_',
...   'H-000-MSG1__-MSG1________-HRV______-000023___-200601040930-C_',
...   'H-000-MSG1__-MSG1________-HRV______-000024___-200601040930-C_',
...   'H-000-MSG1__-MSG1________-HRV______-000025___-200601040930-C_',
...   'H-000-MSG1__-MSG1________-HRV______-000001___-200601040930-C_',
...   'H-000-MSG1__-MSG1________-HRV______-000005___-200601040930-C_',
...   'H-000-MSG1__-MSG1________-HRV______-000010___-200601040930-C_',
...   'H-000-MSG1__-MSG1________-HRV______-000015___-200601040930-C_',
...   'H-000-MSG1__-MSG1________-HRV______-000018___-200601040930-C_'
      ]
>>> for name in name_files:
...     if (pattern.search(name)):
...         print name + " OK"
...     else:
...         print name + " NO OK"
...
H-000-MSG1__-MSG1________-HRV______-000019___-200601040930-C_ OK
H-000-MSG1__-MSG1________-HRV______-000020___-200601040930-C_ OK
H-000-MSG1__-MSG1________-HRV______-000021___-200601040930-C_ OK
H-000-MSG1__-MSG1________-HRV______-000022___-200601040930-C_ OK
H-000-MSG1__-MSG1________-HRV______-000023___-200601040930-C_ OK
H-000-MSG1__-MSG1________-HRV______-000024___-200601040930-C_ OK
```

```
H-000-MSG1__-MSG1________-HRV______-000025___-200601040930-C_ NO OK
H-000-MSG1__-MSG1________-HRV______-000001___-200601040930-C_ NO OK
H-000-MSG1__-MSG1________-HRV______-000005___-200601040930-C_ NO OK
H-000-MSG1__-MSG1________-HRV______-000010___-200601040930-C_ NO OK
H-000-MSG1__-MSG1________-HRV______-000015___-200601040930-C_ NO OK
H-000-MSG1__-MSG1________-HRV______-000018___-200601040930-C_ NO OK
```

Este ejemplo está extraido de Meteogest, que hace uso de expresiones regulares para decidir qué operaciones realizar con archivos basándose en su nombre. En este caso sólo interesan los ficheros que en el penúltimo campo, su valor sea del 19 al 24. El patrón utilizado es sencillo, y obviamente, para este ejemplo podría simplificarse, pero he querido conservarlo tal y como está en el fichero de configuración de ejemplo de Meteogest.

Nótese que las expresiones regulares distinguen entre mayúsculas y minúsculas por lo que:

```
>>> pattern = re.compile ('H')
>>> result = pattern.search('aloha')
>>> result == None
True
```

En cambio, si:

```
>>> pattern = re.compile ('H', re.IGNORECASE)
>>> result = pattern.search('aloha')
>>> result == None
False
```

Puedes consultar otros flags a la hora de construir un patrón en la documentación del módulo re.

También podemos aplicar split con expresiones regulares, lo que es realmente útil:

```
>>> pattern = re.compile ('a[bc]')
>>> text = 'e8acjhf90abcklhd78ac867acaba8'
>>> pattern.split(text)
['e8', 'jhf90', 'cklhd78', '867', '', 'a8']
```

search Vs match

Si se observan los métodos de los objetos expresión regular, se verá que hay dos funciones, search y match. La direferencia entre ambas es que search busca en **toda la cadena** mientras que match **sólo lo hace al principio**.

Utilizaremos un patrón útil para decidir si en una cadena aparece el carácter ".". Como es un caracter que tiene un significado en expresiones regulares, es necesario utilizar el caracter de escape \:

```
>>> pattern = re.compile ('\.')
>>> result = pattern.search ('.gnome')
>>> result == None
False
>>> result = pattern.search ('gno.me')
>>> result == None
False
>>> result = pattern.match ('.gnome')
>>> result == None
False
>>> result = pattern.match ('gno.me')
>>> result == None
True
```

Por tanto se puede decir que search(**"\Apattern"**, text) es lo mismo match(**pattern**, text).

Los objetos Match

Hasta ahora hemos evaluado expresiones y decidido si verifican un patrón en función de si el valor de retorno es None o no. En Python None se evalúa como False por lo que es adecuado para estructuras condicionales. Sin embargo, cuando el valor de retorno no es None, se devuelve un objeto de tipo Match (que se evalúa como cualquier objeto como True). Veamos qué métodos contiene:

```
>>> pattern = re.compile ('a[bc]')
>>> result = pattern.search ('e8acjhf90abcklhd78ac867acaba8')
>>> dir (result)
['__copy__', '__deepcopy__', 'end', 'expand',
'group', 'groupdict', 'groups', 'span', 'start']
>>> result.string
'e8acjhf90abcklhd78ac867acaba8'
>>> result.string[result.start():result.end()]
'ac'
```

Observando la cadena del ejemplo, vemos que el patrón se repite más de una vez. Para procesar cada uno de ellos:

```
>>> text = 'e8acjhf90abcklhd78ac867acaba8'
>>> iterator = pattern.finditer (text)
>>> for result in iterator:
...  print text[result.start():result.end()]
ac
ab
ac
ac
ab
```

Finalmente, el método group. Podemos dar un nombre a uno o un conjunto de elementos del patrón y luego referirnos a ellos. Como siempre, un ejemplo es mucho más útil que cualquier explicación:

```
>>> pattern = re.compile ('(?P<name>.+)\.(?P<extension>.+)')
>>> result = pattern.match ('hello.txt')
>>> result.group('name')
'hello'
>>> result.group('extension')
'txt'
```

Conclusión

Y eso es todo. Conviene conocer muy bien las herramientas con las que contamos a la hora de programar soluciones a nuestros problemas. La expresiones regulares son una herramienta muy útil, cuya potencia sólo se descubre cuando se necesita hacer uso extensivo de ellas. Casi cualquier operación de manipulación de cadenas se puede hacer de manera eficiente con expresiones regulares. Buscar patrones en textos es muy sencillo, así como validar que determinadas cadenas cumplan ciertas condiciones. Por ejemplo, pueden buscar en Internet expresiones regulares que verifiquen si una dirección de correo electrónico es o no correcta (puede ser más complicado de lo que parece si se es estricto) e integrar esta verificación en sus aplicaciones. E infinitas cosas más.

Comentario[s]

KarlsBerg
Escrito por Invitado el 2006-01-15 13:43:53

Muy bueno, te pido permiso para añadir este link a la biblioteca de documentacion python en español que tengo en dotpy.net.

Gracias

Licencia de la página
Escrito por Javi el 2006-01-15 13:54:50

Si lees la licencia, verás que no necesitas pedir permiso para reproducir cualquier artículo. Lo único es que no utilices el artículo con fines comerciales y no hagas obras derivativas (aunque en el caso de este artículo esto tiene poco sentido).

Lo único que has de hacer es incluir un enlace al artículo original como reconociemto.

Y sí, puedes linkarlo (eso ni se pregunta) ;)

muy interesantei
Escrito por Invitado el 2006-02-12 20:38:15

queria pedirte algun consejo de como poder empesar con esto de la programacion por mi cuenta. ya que no tengo los recursos como para estudiar en alguna escuela . mi nombre es sergio tengo 14 años y te pido por favor informacion mi correo es sergio22_2468hotmail.com

grasias

tkinter
Escrito por horacio el 2006-04-13 01:45:50

esta bueno el tuto...aunque le falta lo mas interesnte de esto de programar que es lo grafico supongo que decir algo sobre el modulo tkinter o sobre wxwindows no estaria nada mal ya que no hay mucho de eso en español.

Escribe tu comentario

Por favor cíñete al tema del artículo, sé educado y no envíes spam. Gracias por participar :)

Nombre:

Título:

BBCode:

Comentario:

Escribe lo que ves: *
Pulsa en recargar si tienes problemas para distinguir la imagen

Enviar