Expresiones regulares. Ejercicios resueltos

Ejercicio 1. Extracción de imágenes

Diseña una expresión regular que capture y extraiga el nombre de un archivo de imagen (esto es, *.img, *.jpg, *.png) de una url. Por ejemplo, en las siguientes:

```
http://www.domain.com/static/js/imagen1.jpg
http://www.domain.com/index.php?a=imagen2.img
http://www.domain.com/index.php?a=/imagen3.png
http://www.domain.com/?v=http://domain.com/static/imagen4.mpg
http://www.domain.com/?v=http://domain.com/static/?v=http://www.domain/src/este.js
&var=lol
```

Detalles:

- El nombre del archivo estará precedido por el carácter "/" o "=".
- Luego, caracteres distintos del carácter "/" o "=", seguidos de un punto y una extensión válida. Ésta será la cadena de caracteres que nos interesa capturar

In [1]:

```
import re

casos = """http://www.domain.com/static/js/imagen1.jpg
http://www.domain.com/index.php?a/imagen2.img
www.domain.com/index.php?a=/imagen3.png                          # Dos puntos
www.domain.com/index.php?a=/imagen4.png
=www.domain.com/index.php?a=/imagen5.png
=/www.domain.com/index.php?a=/imagen6.png
http://www.domain.com/?v=http://domain.com/static/imagen7.mpg
http://www.domain.com/?v=http://domain.com/static/imagen8.jpeg/
http://www.domain.com/?v=http://domain.com/static/?v=http://www.domain/src/este.js&var=lol
"""

for caso in casos.split("\n"):
  result = re.findall(r'[/=]([^/=]*\.(?:jpg|mpg|img|png|jpeg))', caso)
  if result:
    print(result[0])   #sólo el primer grupo, pues la extensión ya está incluida
  else:
    print("---")
```

```
imagen1.jpg
imagen2.img
imagen3.png
imagen4.png
imagen5.png
imagen6.png
imagen7.mpg
imagen8.jpeg
---
---
```

Explicación

Explicación:

- La cadena "[/=]" representa el caracter inicial que puede ser "/" o "=".
- La cadena "[^/=]*" representa cualquier cadena de caracteres que no contenga "/" ni "=".
- Un par de paréntesis "normal", representa un grupo que se desea capturar.
- Un par de paréntesis abiero así: "(?:", representa un grupo que NO se desea capturar.
  En el programa, se extrae el primer resultado de la lista de varios posibles, en realidad, uno solo. Si se prueba capturando los dos grupos descritos se verá por qué es necesario el seleccionar uno.

Ejercicio 2. Coloreado de un fragmento de texto, en un archivo HMTL

He aquí un fragmento de un archivo html:

```
<pre>
    <p>'Este fragmento de html es negro, <font color="00ff00">ahora verde</font> y
de nuevo negro...'</p>
</pre>
```

Viendo cómo queda en la pantalla con mi navegador, resulta lo siguiente:

'Este fragmento de html es negro, ahora verde y de nuevo negro...'

Te pido una expresión regular que identifique, en una cadena de caracteres, un fragmento de texto coloreado y el código del color que tiene:

```
>>> cadena = 'Este fragmento de html es negro, <font color="00ff00">ahora verde</f
ont> y de nuevo negro...'
>>> obtener_texto_y_color(cadena)
('00ff00', 'ahora verde')
```

Obviamente, lo más interesante de la función `obtener-texto-y-color` es el patrón usado, en el que se define la expresión regular adecuada. Pero no es lo único.

In [2]:

```
import re

cadena1 = 'Este fragmento de html es negro, <font color="00ff00">ahora verde</font> y de nu

# primera versión del patrón:

patron1 = re.compile('<font color="(......)">(.+)</font>')

patron1.findall(cadena1)
```

Out[2]:

```
[('00ff00', 'ahora verde')]
```

In [3]:

```
"""
Esta versión no funciona si hay más espacios, o si las etiquetas están en mayúsculas,
o si hay otras etiquetas...
"""

cadena2 = 'Fragmento negro, <FONT a=5   color  =  "00ff00" otra_label=456>ahora verde</foNt

# segunda versión del patrón:

patron2 = re.compile('<font.*color *= *"(......)".*>(.+)</font>', re.IGNORECASE)

patron2.findall(cadena2)
```

Out[3]:

```
[('00ff00', 'ahora verde')]
```

In [4]:

```
"""
Ahora consideramos que la cadena tiene varios fragmentos coloreados, y queremos extraerlos
"""

cadena3 = 'Normal, <FONT color = "00ff00">verde</foNt> negro <FONT color = "00aa00">otro co

# Tercera versión del patrón:

patron3 = re.compile('<font.*color *= *"(......)".*>(.+)</font>', re.IGNORECASE)
patron3.findall(cadena3)

# Y no nos gusta la respuesta: Deberíamos excluir los fragmentos con FONT dentro de FONT.
```

Out[4]:

```
[('00aa00', 'otro color')]
```

Necesitamos excluir los fragmentos con FONT dentro de FONT. Para ello, usamos el siquiente patrón:

```
'()'
```

Explicación:

```
(?:       # principio de un grupo que no se captura
  (?!     # principio de un grupo negativo anticipado
    font  # secuencia literal font
  )       # fin de un grupo negativo
  .       # un carácter cualquiera
)         # fin del grupo que no capturado
```

In [5]:

```
"""
Juntando todo:
"""

cadena4 = 'Normal, <FONT color = "00ff00">verde</FONT> negro <font color = "00aa00">otro co

# Tercera versión del patrón:

patron4 = re.compile('(<font(?:(?!font).)*</font>)', re.IGNORECASE)
patron4.findall(cadena4)

# Por supuesto, a cada una de estas cadenas le podemos aplicar de nuevo el patrón segundo:

def extraer_texto_y_color(cadena):
    patron4 = re.compile('(<font(?:(?!font).)*</font>)', re.IGNORECASE)
    fragmentos = patron4.findall(cadena)
    patron2 = re.compile('<font.*color *= *"(......)".*>(.+)</font>', re.IGNORECASE)
    return [patron2.findall(i)[0] for i in fragmentos]

extraer_texto_y_color(cadena4)
```

Out[5]:

```
[('00ff00', 'verde'), ('00aa00', 'otro color')]
```