BUSCANDO PATRÓN
CON EXPRESIONES REGULARES

WORDCAMP
PARA DESARROLLADORES
SEVILLA 2019

¡HOLA!

Soy Pablo López

Estoy aquí para enseñaros a lidiar con expresiones regulares.

Puedes encontrarme en @desarrollowp

1.
Expresiones regulares

Breve introducción...

Las expresiones regulares son patrones utilizados para encontrar una determinada combinación de caracteres dentro de una cadena de texto.

Expresiones regulares => regex

- No se consideran lenguaje de programación
- Son utilizadas para buscar patrones, realizar operaciones de sustitución, validaciones, etc...
- Se utilizan en HTML5, JavaScript, PHP... con pequeñas diferencias entre ellos
- Se leen siempre de izquierda a derecha

Algunos desarrolladores cuando se enfrentan a un problema piensan que la solución es usar expresiones regulares. En este momento, ya tienen dos problemas

-- Jamie Zawinski

La siguiente secuencia sigue con...

WORDCAMP
PARA DESARROLLADORES
SEVILLA 2019

16 32 64

Pero también puede seguir con...

10 14 16

WORDCAMP
PARA DESARROLLADORES
SEVILLA 2019

Ó también con...

7004

Polinomio de Lagrange

La serie 2, 4, 8 obviamente podría continuar con 16, pero también por 10 ó 7004. Es siempre posible encontrar una regla. Todo depende de lo complicada que sea la regla.

$$f(x) = 1165\ X^3 - 6989\ X^2 + 12814\ X - 6988$$

La satisfacción cuando clavas una expresión regular es directamente proporcional al miedo a no saber si tuviste en cuenta todas las casuísticas.

-- Pablo López

2.
Caracteres y Metacaracteres

Notación y convenciones...

Metacaracteres

- ^ Inicio de regex
- $ Fin de regex
- * Coincide con lo anterior cero o más veces
- + Coincide con lo anterior una o más veces
- ? Coincide con lo anterior cero o una vez
- - Definir un rango (A-Z)

Metacaracteres

- { } Repetición
- ( ) Agrupamiento
- [ ] Un carácter o rango de caracteres
- . Cualquier carácter, excepto salto de línea
- \ Escape
- | Elementos alternos
- ! Negación

Metacaracter comodín => .

/.a.a.a/

El punto es el comodín, representa cualquier carácter excepto el salto de línea

Matches

mañana

banana

papaya

fabada

Caracteres especiales

- \t Tabulador
- \v Tabulador vertical
- \r Retornos de carro
- \n Nueva línea
- \e Escape
- \0 Null

Conjunto de caracteres

Definiremos un conjunto de caracteres por medio de corchetes [ ]. Evalúa caracteres, NO palabras. No importa el orden.

- /[aeiou]/
- /[áéíóú]/
- /c[ao]sa/ casa, cosa
- /c[oa]sa/ casa, cosa

Rango de caracteres

Mediante corchetes y guiones medios.

- /[a-z]/ (alfabeto inglés, sin acentos ni “ñ”)
- /[A-Z]/
- /[0-9]/
- /[a-zA-z0-9]/
- /[á-ü]/

Deben escribirse en orden unicode

Conjunto negativo de caracteres

El carácter ^ dentro de los corchetes es una negación de uno de los conjuntos de caracteres.

- /[^aeiou]/
- /[^"#$%&/()=*]/

Metacaracteres en conjunto de caracteres

Los metacaracteres dentro de los conjuntos de caracteres ya están "escapados", excepto los siguientes:

-] - ^ /
- /var[[(][0-9][\])]/ var(9) ó var[9]

Abreviaciones de conjuntos de caracteres

\d	Dígitos	[0-9]
\w	Caracteres	[a-zA-Z0-9_]
\s	Whitespaces	[\t\r\n]
\D	No dígito	[^0-9]
\W	No caracteres	[^a-zA-Z0-9_]
\S	No whitespaces	[^ \t\r\n]

Flags

- /regex/
- /regex/g **g**lobal No se detiene ante la primera coincidencia
- /regex/i **i**nsensitive Busca en mayúsculas y minúsculas
- /regex/m **m**ultiline ^ y $ aplican a cada línea, no a toda la cadena
- /regex/s **s**ingle line . coincidirá con saltos de línea

- /regex/gms

3.
Expresiones de repetición

Metacaracteres, cuantificadores, greedy, lazy...

Metacaracteres de repetición

- * Cero o más veces el elemento precedente
- + Una o más veces el elemento precedente
- ? Cero o una vez el elemento precedente

- /as*/ a, as, asss
- /as+/ a, as, asss
- /as?/ a, as, asss

Cuantificadores de repetición

- {num} Num veces el elemento precedente
- {min,max} Elemento precedente entre min - max
- {min,} Elemento precedente min veces

- /\d{4}/ 4 dígitos exactamente
- /\d{4,8}/ Entre 4 y 8 dígitos, ambos inclusive
- /\d{4,}/ Mínimo 4 dígitos

Expresiones Greedy (codiciosas)

Las expresiones regulares son **greedy** (codiciosas) por defecto. Siempre van a intentar seleccionar la cadena más grande

/ (.*)<\/p>/

(lorem ipsum dolor sit amet)

Expresiones Lazy (perezosas)

Serán **lazy** (perezosas) añadiendo una **?** a nuestra expresión regular

Greedy	**Lazy**
(.*)	(.*?)
(.+)	(.+?)
(?)	(??)
{min.max}	{min,max}?

Greedy vs Lazy

/ (.*)<\/p>/

(lorem ipsum dolor sit amet)

/ (.*?)<\/p>/

(lorem ipsum)(dolor sit amet)

4.
Agrupar y alternar

Capturando bloques de información...

Metacaracteres para agrupar

Por medio de los paréntesis () podemos agrupar expresiones regulares.

- Aplicar operadores de repetición a grupos
- Hacer las expresiones más legibles
- Capturar un grupo para matchear o reemplazar
- No es posible agrupar dentro de un conjunto de caracteres [()]

Metacaracteres para alternar

Por medio del metacaracter | (pipe) podemos alternar expresiones regulares. Equivale a OR.

- Matchea expresiones de izquierda a derecha
- Tienen prioridad las expresiones de la izquierda
- Se suele alternar expresiones dentro de ()

/<(p|div)>/

Expresiones Eager (ansiosas)

Las expresiones regulares son **eager** (ansiosas).

- /(moto|motocicleta)/ → moto, motocicleta
- /moto(cicleta)?/ → moto, motocicleta

5.
Trabajando con grupos

Grupos, referencias, no captura, identificadores...

Grupos y referencias

Las expresiones agrupadas entre paréntesis **()** por defecto son “capturadas”. Podemos referenciarlos del **\1** al **\9**.

- /<(div|p)>.*<\/(\1)>/

✔ Hola mundo

✔ Hola mundo

✘ Hola mundo

Grupos de no captura

No se captura un grupo anteponiendo **?:**

- /(http|https):\/\/([^\/\r\n]+)(\/[^\r\n]*)?/

MATCH INFORMATION

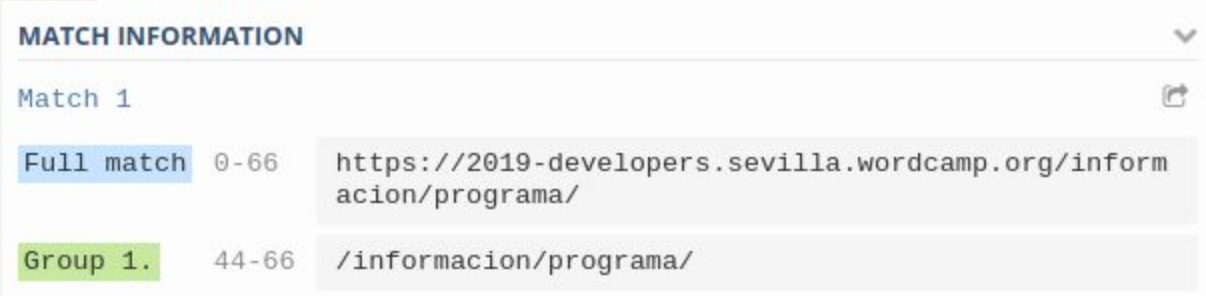
Match 1

Full match	0-66	https://2019-developers.sevilla.wordcamp.org/informacion/programa/
Group 1.	0-5	https
Group 2.	8-44	2019-developers.sevilla.wordcamp.org
Group 3.	44-66	/informacion/programa/

Grupos de no captura

No se captura un grupo anteponiendo **?:**

- /(?:http|https):\/\/(?:[^\/\r\n]+)(\/[^\r\n]*)?/



Nombrar grupos

Obtendremos una clave en lugar de un número en el array de resultados mediante (**?<name>**...), (**?'name'**...) o (**?P<name>**...).

- /(?<scheme>.*):\/\/(?<domain>[^\/\r\n]+)(?<path>\/[^\r\n]*)?/

Full match	0-75	https://2019-developers.sevilla.wordcamp.org/informacion/programa-schedule/
Group `scheme`	0-5	https
Group `domain`	8-44	2019-developers.sevilla.wordcamp.org
Group `path`	44-75	/informacion/programa-schedule/

6.
Aserciones

Si se cumple o no una condición pasada o futura...

Positive Lookahead (?=...) (*Si termina en...*)

WORDCAMP PARA DESARROLLADORES SEVILLA 2019

Futuro positivo. Sólo *matchea* si encuentra una condición futura. La condición no es capturada.

- /guarda(?=barros|bosques)/

✔ guardabarros
✔ guardabosques
✘ guardaespaldas

Negative Lookahead (?!...) (*Si no termina en...*)

Futuro negativo. Sólo *matchea* si no encuentra una condición futura. La condición no es capturada.

- /guarda(?!barros|bosques)/

  ✘ guardabarros

  ✘ guardabosques

  ✔ guardaespaldas

Positive Lookbehind (?<=...) (*Si empieza por...*)

Pasado positivo. Sólo *matchea* si encuentra una condición pasada. La condición no es capturada.

- /(?<=solu|informa)ción/

✔ Tengo la información
✔ con la solución
✘ de la ecuación

Negative Lookbehind (?<!...) (*Si no empieza por...*)

Pasado negativo. Sólo *matchea* si no encuentra una condición pasada. La condición no es capturada.

- /(?<!solu|informa)ción/

  ✘ Tengo la información
  ✘ con la solución
  ✔ de la ecuación

7.
Condicionales

Capturar en caso de que se cumpla una condición...

Condicionales if-then-else

Si **if** se evalua como *true*, se intentará matchear con **then**, en caso contrario con **else**.

- /(This condition)(?(1) is true| is false)/

This condition is true

This other one is false

8.
Expresiones regulares legibles

Para no dejar una mala herencia...

Expresiones regulares legibles

En ocasiones nos encontramos con un chorizo de expresión regular en un método sin comentar.

- /(?:(?=.{17}$)97[89][-](?:[0-9]+[-]){2}[0-9]+[-][0-9]|97[89][0-9]{10}|(?=.{13}$)(?:[0-9]+[-]){2}[0-9]+[-][0-9Xx]|[0-9]{9}[0-9Xx])/

Más info

Expresiones regulares legibles

- /^(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?!.*\s).*$/

```
"/^" .              //Inicio de regex
"(?=.*\d)" .        //Si existe al menos un número
"(?=.*[a-z])" .     //Si existe al menos una minúscula
"(?=.*[A-Z])" .     //Si existe al menos una mayúscula
"(?!.*\s)" .        //Que no existan espacios
".*" .              //Todo, cero o más veces
"$/"                //Fin de regex
```

9.
¿Y en WordPress?

Disponemos de algunas funciones auxiliares...

get_shortcode_regex()

Devuelve la expresión regular utilizada para buscar shortcodes dentro de un contenido.

Combina todas las etiquetas de shortcodes registrados en una sola expresión regular.

get_shortcode_regex()

WORDCAMP PARA DESARROLLADORES SEVILLA 2019

```
return
    '\\['                              // Opening bracket
    . '(\\[?)'                         // 1: Optional second opening bracket for escaping shortcodes: [[tag]]
    . "($tagregexp)"                   // 2: Shortcode name
    . '(?![\\w-])'                     // Not followed by word character or hyphen
    . '('                              // 3: Unroll the loop: Inside the opening shortcode tag
    .     '[^\\]\\/]*'                 // Not a closing bracket or forward slash
    .     '(?:'
    .         '\\/(?!\\])'             // A forward slash not followed by a closing bracket
    .         '[^\\]\\/]*'             // Not a closing bracket or forward slash
    .     ')*?'
    . ')'
    . '(?:'
    .     '(\\/)'                      // 4: Self closing tag ...
    .     '\\]'                        // ... and closing bracket
    . '|'
    .     '\\]'                        // Closing bracket
    .     '(?:'
    .         '('                      // 5: Unroll the loop: Optionally, anything between the opening and closing shortcode tags
    .             '[^\\[]*+'           // Not an opening bracket
    .             '(?:'
    .                 '\\[(?!\\/\\2\\])' // An opening bracket not followed by the closing shortcode tag
    .                 '[^\\[]*+'       // Not an opening bracket
    .             ')*+'
    .         ')'
    .         '\\[\\/\\2\\]'           // Closing shortcode tag
    .     ')?'
    . ')'
    . '(\\]?)';                        // 6: Optional second closing brocket for escaping shortcodes: [[tag]]
```

wp_spaces_regexp()

Devuelve la expresión regular para caracteres de espacios en blanco comunes: espacios, nuevas líneas, tabs,   y el nbsp UTF-8.

Evita posibles errores en la codificación de un espacio

wp_spaces_regexp()


```
$spaces = wp_spaces_regexp();
$pattern =
        '/<p>'                      // Opening paragraph
        . '(?:' . $spaces . ')*+'   // Optional leading whitespace
        . '(.*)'                    // The regex
        . '(?:' . $spaces . ')*+'   // Optional trailing whitespace
        . '<\\/p>/';                // Closing paragraph
```

10.
¿Al lío?

¡Abrid vuestros portátiles!

https://regex101.com/

r/Vyrqce

Añade el atributo rel=”nofollow” a los enlaces

https://regex101.com/

r/zmJ99V

Extrae las URLs de los enlaces que aparecen en el siguiente texto

https://regex101.com/

r/krS02a

Extrae los shortcodes para su eliminación

¡GRACIAS!

¿Alguna pregunta?

Puedes encontrarme en @desarrollowp

Presentation template by SlidesCarnival