Índice

[**¿Qué son las expresiones regulares?** 1](#_Toc75020845)

[**Aplicaciones de las expresiones regulares** 1](#_Toc75020846)

[**Máquina de estado finito** 2](#_Toc75020847)

[**Introducción al lenguaje de expresiones regulares** 3](#_Toc75020848)

[**El carácter (.)** 4](#_Toc75020849)

[**Las clases predefinidas y construidas** 4](#_Toc75020850)

[**Dígitos: \d** 4](#_Toc75020851)

[**Palabras: \w** 5](#_Toc75020852)

[**Espacios: \s** 5](#_Toc75020853)

[**Los delimitadores** 5](#_Toc75020854)

[**Los contadores** 6](#_Toc75020855)

[**(?) como delimitador** 6](#_Toc75020856)

[**Not (^), su uso y sus peligros** 7](#_Toc75020857)

[**Principio (^) y final de línea ($)** 8](#_Toc75020858)

[**Logs** 8](#_Toc75020859)

[**URLs** 9](#_Toc75020860)

[**Mails** 10](#_Toc75020861)

[**Localizaciones** 10](#_Toc75020862)

[**Búsqueda y reemplazo** 12](#_Toc75020863)

[**Uso de REGEX para descomponer querys GET** 13](#_Toc75020864)

[**Banderas** 14](#_Toc75020865)

[**‘grep’ y ‘find’ desde consola** 16](#_Toc75020866)

# **¿Qué son las expresiones regulares?**

Son patrones en los que definimos que cadenas de caracteres entran o no entran en el patrón diseñado.

Además de ser útiles para quedarnos con parte de la información que necesitamos y descartamos la que no.

# **Aplicaciones de las expresiones regulares**

Las expresiones regulares pueden llegar a ser muy raras por la forma en la que se ven, pero son muy útiles. Aprender a usarlas nos ayuda en pocas palabras, a buscar.

Se diferencia del **CTRL** + **F** porque éste busca textos precisos y te arroja el **match**. Con expresiones regulares es más complejo, por ejemplo, se pueden buscar patrones (buscar todas las palabras que estén entre dos espacios, palabras que empiecen con mayúscula, encontrar la primera palabra de cada línea, etc.)

Se usa mucho en logs de servidores que son archivos enormes para analizarlos

Las expresiones regulares son usadas tanto en **frontend** como en **backend**.

## **Máquina de estado finito**

Realiza cómputos en forma automática sobre una **entrada** para **producir** una **salida**. Este modelo está **conformado** por un **alfabeto**, un conjunto de **estados finito**, una **función de transición**, un **estado inicial** y un conjunto de **estados finales**.

# **Introducción al lenguaje de expresiones regulares**

Las expresiones regulares intentan solucionar problemas del día a día. Se intenta buscar la forma en que ciertos datos son presentados, por ejemplo, un número de teléfono dependiendo de país y zona, tiene determinada cantidad de números, a veces separados con guiones o puntos, pero la estructura siempre es la mismo.

Otro ejemplo de uso de expresiones regulares sería intentar cambiar en un **.csv** los precios de modo europeo a modo americano

* Un dígito se puede expresar con **/d**
* Carácter de palabra: **/w**

# **El carácter (.)**

El punto significa **encuéntrame todo lo que sea un carácter**.

* **Cadena de Caracteres**: Es un carácter ASCII generalmente, seguido de otro carácter y de otro. No todos son visibles, **el espacio**, por ejemplo. Cada carácter es un carácter.

# **Las clases predefinidas y construidas**

Las búsquedas en las expresiones regulares funcionan en múltiplos de la cantidad de caracteres que explícitamente indicamos.

## **Dígitos: \d**

* Encuentra todos los dígitos de 0 a 9.
* **\d** es equivalente a poner **[0-9]**.
* Si en vez de **\d**, usamos por ejemplo **[0-2]** nos encontrará solamente los dígitos de **0** a **2**.
* Podemos usar “**\D**” para encontrar justo lo contrario, todo lo que **no** son **dígitos**.

## **Palabras: \w**

* Encuentra todo lo que puede ser parte de una palabra, tanto letras (minúsculas o mayúsculas) como números.
* **\w** es equivalente a poner **[a-zA-Z0-9\_]**.
* Si en vez de **\w**, usamos por ejemplo **[a-zA-Z]** nos encontrará solamente las letras.
* Podemos usar “**\W**” para encontrar justo lo contrario, todos los caracteres que **no** **son parte de palabras**.

## **Espacios: \s**

* Encuentra todos los espacios (los saltos de línea también son espacios).
* Podemos usar “**\S**” para encontrar justo lo contrario, todo lo que **no son espacios**.

Escapados de caracteres: **\**

* La barra invertida o backslash permite escapar un carácter especial, por ej, el punto (.)

# **Los delimitadores**

* **\***: greedy - 0 o más.
* **+**: 1 o más.
* **?**: 0 o 1. Dependiendo de como se use, este carácter puede también ser **que este** o **que no**.

# **Los contadores**

Aplicando a un carácter que lo preceda se puede colocar entre llaves de esta forma, para indicarle que busque la cantidad deseada de caracteres.

{**Cota inferiror**, **Cota superior**}

* **{3}**: Exactamente 3.
* **{3,}**: 3 o más.
* **{3,5}**: 3,4 o 5

# **(?) como delimitador**

El **?** indica al patrón que encuentre las coincidencias de manera rápida (o greedy), es decir, devolviendo el resultado más pequeño que haga match hasta donde se encuentra el delimitador, y esto lo haga tantas veces como sea posible dentro de la cadena.

* **\*?:** Coincide con el elemento anterior cero o más veces, pero el menor número de veces que sea posible.
* **+?**:Coincide con el elemento anterior una o más veces, pero el menor número de veces que sea posible.
* **??**: Coincide con el elemento anterior cero o una vez, pero el menor número de veces que sea posible.

La función de (**?**) como delimitador conociste justamente en delimitar a la menor cantidad posible de los matches.

# **Not (^), su uso y sus peligros**

Este carácter nos permite **negar** una **clase** o construir “**anticlases**”, vamos a llamarlo así, que es, **toda la serie de caracteres que no queremos que entren en nuestro resultado de búsqueda**.

Para esto definimos una clase, por ejemplo: [0-9], y la negamos [^0-9] para buscar todos los caracteres que coincidan con cualquier carácter que no sea 0,1,2,3,4,5,6,7,8 ó 9

# **Principio (^) y final de línea ($)**

Estos dos caracteres indican en qué posición de la línea debe hacerse la búsqueda:

* El **^** se utiliza para indicar el principio de línea.
* El **$** se utiliza para indicar final de línea.

Ejemplo

^ ------------- $

# **Logs**

Las expresiones regulares son muy útiles para encontrar líneas específicas que nos dicen algo muy puntual dentro de los archivos de logs que pueden llegar a tener millones de líneas.

# **URLs**

Una de las cosas que más vamos a usar en la vida, seamos frontend o backend, serán directamente dominios o direcciones de internet, ya sea direcciones completas de archivo (una url) o puntualmente dominios para ver si es correcto un mail o no.

**TLD**

**T**op **L**evel **D**omains, dominio de nivel superior. Es la terminación de una página, puede ser **.com**, **.org**, **.mx**, etc.

Búsqueda:



(esta última parte selecciona a los TLD).

Búsqueda la url completa



No tiene en cuenta a los espacios en blanco.

Búsqueda “para simplificarte la vida”

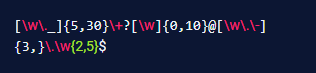


Es completamente inútil si existen URLs no válidas, por ejemplo, http-123423312, esta página es basura, ruido, pero con esta búsqueda quedará seleccionada, en un mundo ideal dónde se guardan las URLs de la misma forma y sin errores sería idónea, **pero** **dónde existe el error humano puede volverse inútil**.

Lista actualizada de [TLD](https://en.wikipedia.org/wiki/List_of_Internet_top-level_domains).

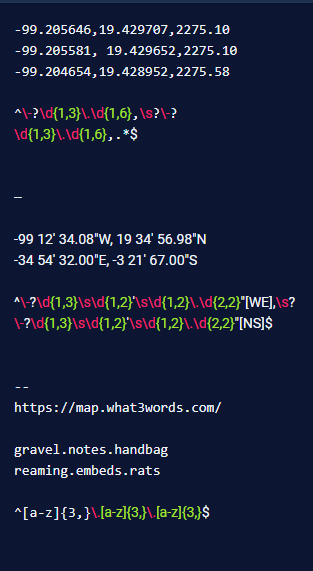
# **Mails**

Quedamos en que ya podemos definir URLs, y dentro de las URLs están los dominios. No es infalible, pero es muy útil para detectar la gran mayoría de errores que cometen los usuarios al escribir sus emails.



# **Localizaciones**

Tips comunes de qué hacer y sobre todo qué no hacer con expresiones regulares, usando como ejemplo datos de posicionamiento en el mapa: **latitud** y **longitud**.



# **Búsqueda y reemplazo**

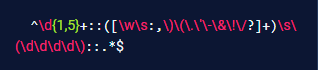
Al igual que una navaja suiza, las expresiones regulares son una herramienta increíblemente útil, pero tienes que darle la importancia y las responsabilidades adecuadas a cada una, ya que no son la panacea (mítico medicamento que cura todas las enfermedades), no solucionan todos los problemas.

El uso más conveniente de las expresiones regulares es buscar coincidencias o matches de cadenas en un texto, y si es necesario, reemplazarlas con un texto diferente.

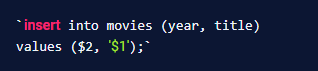
**Grupos**

Se pueden generar grupos al utilizar paréntesis “**()**”, cada uno de esos paréntesis a la hora de reemplazar podemos acceder a el valor dentro del grupo con la variable **$1**, **$2**, **$n**, por ejemplo:

Find



Replace



Cabe destacar que **$n** es una forma de crear **variables** en la **terminal** y es una forma generalizada en la gran mayoría.

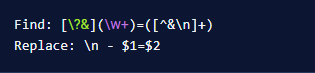
# **Uso de REGEX para descomponer querys GET**

Al hacer consultas a sitios web mediante el método **GET** se envían todas las variables al servidor a través de la misma URL.

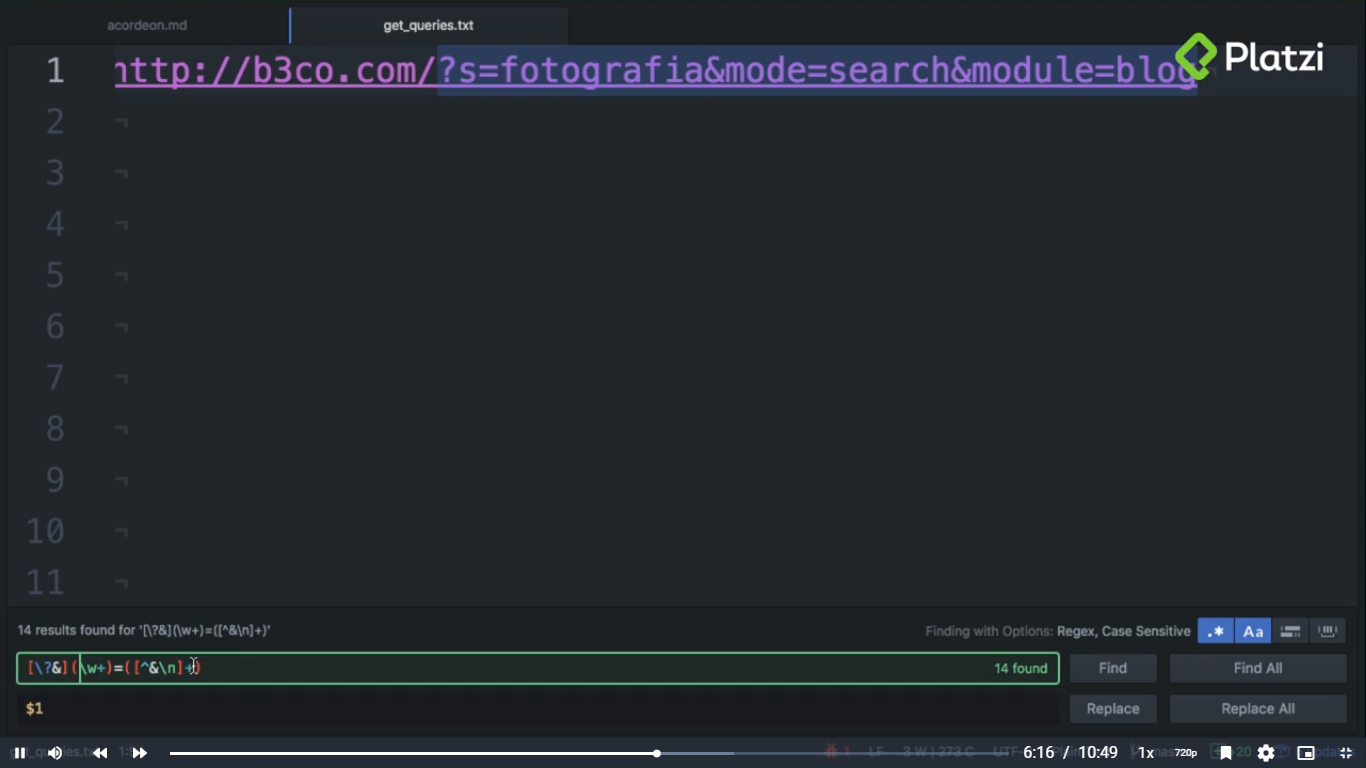
La parte de esta url que viene luego del signo de interrogación **?** se le llama query del request que es: **variable1**=**valor1**&**variable2**=**valor2**&... y así tantas veces como se necesite.

Veremos cómo extraer estas variables usando expresiones regulares.

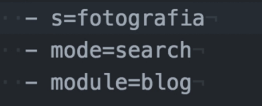
Encontrar el nombre de la variable y su valor, hacemos:



Pasando de esto



A esto



**Nota**: En Visual Studio Code al querer hacer una nueva línea en el **replace** debemos colocar el salto de línea: **\n**

# **Banderas**

Las expresiones regulares pueden tener banderas que afectan la búsqueda, éstas deberán de estar hasta el final de la línea.

Listado de Banderas en js

* **i**: Con este indicador, la búsqueda no distingue entre mayúsculas y minúsculas: no hay diferencia entre A y a
* **g**: Con esta marca, la búsqueda busca todas las coincidencias, sin esta bandera, solo se devolvería la primera coincidencia.
* **m**: Modo multilínea.
* **s**: Habilita el modo “dotall”, que permite un punto. para que coincida con el carácter de nueva línea **\n**
* **u**: Permite el soporte completo de Unicode. La bandera permite el procesamiento correcto de pares sustitutos.
* **y**: Modo “adhesivo” búsqueda en la posición exacta del texto.

# **‘grep’ y ‘find’ desde consola**

