

Expresiones regulares

Las expresiones regulares nos permiten buscar patrones en un string, por tanto buscar el trozo que coincida o cumpla dicha expresión. En JavaScript, podemos crear una expresión regular instanciando un objeto de la clase `RegExp` o escribiendo directamente la expresión entre dos barras `'/'`.

En este apartado veremos conceptos básicos sobre las expresiones regulares, podemos aprender más sobre ellas en el siguiente [enlace](#). También, hay páginas web como [esta](#) que nos permiten probar expresiones regulares con cualquier texto.

Una expresión también puede tener uno o más modificadores como `'g'` → Búsqueda global (Busca todas las coincidencias y no sólo la primera), `'i'` → Case-insensitive (No distingue entre mayúsculas y minúsculas), o `'m'` → Búsqueda en más de una línea (Sólo tiene sentido en cadenas con saltos de línea). En Javascript puedes crear un objeto de expresión regular con estos modificadores de dos formas:

```
var reg = new RegExp("[0-9]{2}", "gi");
var reg2 = /^[0-9]{2}/gi;
console.log(reg2 instanceof RegExp); // Imprime true
```

Estas dos formas son equivalentes, por tanto podéis elegir cual usar.

Expresiones regulares básicas

La forma más básica de una expresión regular es incluir sólo caracteres alfanuméricos (o cualquier otro que no sea un carácter especial). Esta expresión será buscada en el string, y devolverá `true` si encuentra ese substring.

```
var str = "Hello, I'm using regular expressions";
var reg = /reg/;
console.log(reg.test(str)); // Imprime true
```

En este caso, “reg” se encuentra en “Hello, I'm using regular expressions”, por lo tanto, el método `test` devuelve `true`.

Corchetes (opción entre caracteres)

Entre corchetes podemos incluir varios caracteres (o un rango), y se comprobará si el carácter en esa posición de la cadena coincide con alguno de esos .

`[abc]` → Algún carácter que sea `'a'`, `'b'`, ó `'c'`

`[a-z]` → Algún carácter entre `'a'` y `'z'` (en minúsculas)

`[0-9]` → Algún carácter entre 0 y 9 (carácter numérico)

`[^ab0-9]` → Algún caracter excepto (`^`) `'a'`, `'b'` ó número.

Pipe → `|` (opción entre subexpresiones)

`(exp1|exp2)` → La coincidencia en la cadena será con la primera expresión o con la segunda.

Podemos añadir tantos pipes como queramos.

Meta-caracteres

- . (punto) → Un único carácter (cualquier carácter excepto salto de línea)
- \w → Una palabra o carácter alfanumérico. \W → Lo opuesto a \w
- \d → un número o dígito. \D → Lo opuesto a \d
- \s → Carácter de espacio. \S → Lo opuesto a \s
- \b → Delimitador de palabra. Coincide el principio o final de palabra (no un carácter). Por ejemplo, \bcase\b/, coincide con “case” pero no “uppercase” o “casein”.
- \n → Nueva línea
- \t → Carácter de tabulación

Cuantificadores

- + → El carácter que le precede (o la expresión dentro de un paréntesis) se repite 1 o más veces.
 - * → Cero o más ocurrencias. Lo contrario a +, no tiene porqué aparecer
 - ? → Cero o una ocurrencia. Se podría interpretar como que lo anterior es opcional.
 - {N} → Debe aparecer N veces seguidas.
 - {N,M} → De N a M veces seguidas. {N,} → Al menos N veces seguidas.
 - ^ → Principio de cadena.
 - \$ → Final de la cadena.
- Podemos encontrar más sobre las expresiones regulares de JavaScript [aquí](#).

Ejemplos

- `/^[0-9]{8}[a-z]$/i` → Comprueba si un string tiene un DNI. Esta expresión coincidirá con una cadena que comience (^) con 8 números, seguidos con una letra. No puede haber nada después (\$ → final de cadena). El modificador ‘i’ hace que no se distinga entre mayúsculas y minúsculas.
- `/^\\d{2}\\\\\\d{2}\\\\(\\d{2}|\\d{4})$/` → Comprueba si una cadena contiene una fecha en el formato **DD/MM/YY** ó **DD/MM/YYYY**. El metacaracter \\d es equivalente a usar [0-9], y el año no puede tener tres números, o son 2 o 4.
- `/\\b[aeiou]\\w*\\b/i` → Comprueba si hay alguna palabra en el string que comienza por una vocal seguida de cero o más caracteres alfanuméricos (números, letras o ‘_’).

Métodos para las expresiones regulares en JavaScript

A partir de un objeto RegExp, podemos usar dos métodos para comprobar si una cadena cumple una expresión regular. Los dos métodos son **test** y **exec**.

El método **test()** recibe una cadena e intenta de encontrar una la coincidencia con la expresión regular. Si el modificador global ‘g’ se especifica, cada vez que se llame al método se ejecutará desde la posición en la que se encontró la última coincidencia, por tanto podemos saber cuántas veces se encuentran coincidencias con esa expresión. Debemos recordar que si usamos el modificador ‘i’ no diferencia entre mayúsculas y minúsculas.

```
var str = "I am amazed in America";
var reg = /am/g;
console.log(reg.test(str)); // Imprime true
console.log(reg.test(str)); // Imprime true
console.log(reg.test(str)); // Imprime false, hay solo dos coincidencias
var reg2 = /am/gi; // "Am" será lo que busque ahora
console.log(reg.test(str)); // Imprime true
console.log(reg.test(str)); // Imprime true
```

```
console.log(reg.test(str)); // Imprime true. Ahora tenemos 3 coincidencias con este nuevo patrón
```

Si queremos más detalles sobre las coincidencias, podríamos usar el método **exec()**. Este método devuelve un objeto con los detalles cuando encuentra alguna coincidencia. Estos detalles incluyen el índice en el que empieza la coincidencia y también el string entero.

```
var str = "I am amazed in America";
var reg = /am/gi;
console.log(reg.exec(str)); // Imprime ["am", index: 2, input: "I am amazed in America"]
console.log(reg.exec(str)); // Imprime ["am", index: 5, input: "I am amazed in America"]
console.log(reg.exec(str)); // Imprime ["Am", index: 15, input: "I am amazed in America"]
console.log(reg.exec(str)); // Imprime null. No hay más coincidencias
```

Una mejor forma de usarlo sería:

```
var str = "I am amazed in America";
var reg = /am/gi;
var match;
while (match = reg.exec(str)) {
  console.log("Patrón encontrado!: " + match[0] + ", en la posición: " + match.index);
}
/* Esto imprimirá:
* Patrón encontrado!: am, en la posición: 2
* Patrón encontrado!: am, en la posición: 5
* Patrón encontrado!: Am, en la posición: 15 */
```

De forma similar, hay métodos de la clase **String** (sobre la cadena esta vez) que podemos usar, estos admiten expresiones regulares como parámetros (vamos, a la inversa). Estos métodos son **match** (Funciona de forma similar a **exec**) y **replace**.

El método **match** devuelve un array con todas las coincidencias encontradas en la cadena si ponemos el modificador global → **g**, si no ponemos el modificador global, se comporta igual que **exec()**.

```
var str = "I am amazed in America";
console.log(str.match(/am/gi)); // Imprime ["am", "am", "Am"]
```

El método **replace** devuelve una nueva cadena con las coincidencias de la expresión regular reemplazadas por la cadena que le pasamos como segundo parámetro (si el modificador global no se especifica, sólo se modifica la primera coincidencia). Podemos enviar una función anónima que procese cada coincidencia encontrada y nos devuelva la cadena con los reemplazos correspondientes.

```
var str = "I am amazed in America";
console.log(str.replace(/am/gi, "xx")); // Imprime "I xx
xxazed in xxerica"
```

```
console.log(str.replace(/am/gi, function(match) {  
  return "-" + match.toUpperCase() + "-";  
})); // Imprime "I -AM- -AM-azed in -AM-erica"
```