1 Expresiones Regulares

Pequeño manual de creación de expresiones regulares utilizando JavaScript.

> **(Para realizar las siguientes prácticas crear un archivo .js vacío enlazado a un .html y emplear la consola del navegador).**

Para crear el patrón de RegExp tenemos en JavaScript dos modos:

```
let re1 = /abc/;    //Utilizando el carácter backslash
let re2 = new RegExp("abc");    //Objeto RegExp()
```

Fijarse que cuando se emplea el objeto *RegExp* se emplea un *string* normal para escribir el patrón.

- Para crear las expresiones regulares emplearemos muchos "caracteres especiales" (. , + * etc). Si necesitamos "buscar" alguno de estos caracteres en el texto será necesario "escaparlo", para ello emplearemos el carácter especial *slash* \.

```
let dieciochoPlus = /dieciocho\+/;
```

1.1 Método *test*

El método **test** de los objetos *RegExp* es el modo más simple de comprobar la existencia de un patrón en un texto.

```
console.log(/abc/.test("abcde"));
// true
// --------- //
//Otro modo de hacerlo definiendo las variables:
// "re" -> Es la expresión regular
// "texto1" y "texto2" -> Textos donde comprobamos la existencia de ese patrón
let re = new RegExp("abc");
let texto1 = "abcde";
console.log(re.test(texto1));
// true
let texto2 = "abxde";
console.log(re.test(texto2));
// false
```

Vemos que en este caso, el patrón consiste en caracteres alfanuméricos, no tiene ningún tipo de caracteres especiales.

1.2 Conjuntos de caracteres

Si colocamos los caracteres entre corchetes, se buscará la existencia de cualquiera de ellos en el texto. En los siguientes ejemplos, todos hacen lo mismo.

```
console.log(/[0123456789]/.test("en 1992"));
// -- //
console.log(/[0-9]/.test("en 1992"));
// -- //
console.log(/\d/.test("en 1992"));
// Todos : true
```

Como vemos en este último ejemplo, varios grupos de caracteres comunes tienen sus propios atajos:

- **\d** : Un dígito.
- **\w** : Un carácter alfanumérico,
- **\s** :