Fechas

En Javascript tenemos la clase Date, que encapsula información sobre fechas y métodos para operar, permitiéndonos almacenar la fecha y hora local (timezone).

```
let date = new Date(); // Crea objeto Date almacena la fecha actual
console.log(typeof date); // Imprime object
console.log(date instanceof Date); // Imprime true
console.log(date); // Imprime (en el momento de ejecución) Fri Jun 24 2016 12:27:32 GMT+0200 (CEST)
```

Podemos enviarle al constructor el número de milisegundos desde el 1/171970 a las 00:00:00 GMT (Llamado **Epoch** o **UNIX time**). Si pasamos más de un número, (sólo el primero y el segundo son obligatorios),el orden debería ser: 1to → año, 2º → mes (0..11), 3º → día, 4º → hora, 5º → minuto, 6º → segundo. Otra opción es pasar un string que contenga la fecha en un formato válido.

```
let date  = new Date(1363754739620); // Nueva fecha 20/03/2013 05:45:39 (milisegundos desde Epoch)
let date2 = new Date(2015, 5, 17, 12, 30, 50); // 17/06/2015 12:30:50 (Mes empieza en 0 -> Ene, ... 11 -> Dic)
let date3 = new Date("2015-03-25"); // Formato de fecha largo sin la hora YYYY-MM-DD (00:00:00)
let date4 = new Date("2015-03-25T12:00:00"); // Formato fecha largo con la fecha
let date5 = new Date("03/25/2015"); // Formato corto MM/DD/YYYY
let date6 = new Date("25 Mar 2015");  // Formato corto con el mes en texto (March también sería válido).
let date7 = new Date("Wed Mar 25 2015 09:56:24 GMT+0100 (CET)"); // Formato completo con el timezone
```

Si, en lugar de un objeto Date, queremos directamente obtener los milisegundos que han pasado desde el 1/1/1970 (Epoch), lo que tenemos que hacer es usar los métodos **Date.parse**(string) y **Date.UTC**(año, mes, día, hora, minuto, segundos). También podemos usar **Date.now()**, para la fecha y hora actual en milisegundos.

```
let nowMs = Date.now(); // Momento actual en ms
let dateMs = Date.parse("25 Mar 2015"); // 25 Marzo 2015 en ms
let dateMs2 = Date.UTC(2015, 2, 25); // 25 Marzo 2015 en ms
```

La clase Date tiene **setters** y **getters** para las propiedades: **fullYear**, **month** (0-11), **date** (día), **hours**, **minutes**, **seconds**, y **milliseconds**. Si pasamos un valor negativo, por ejemplo, mes = -1, se establece el último mes (dic.) del año anterior.

```
// Crea un objeto fecha de hace 2 horas
let twoHoursAgo = new Date(Date.now() - (1000*60*60*2)); // (Ahora - 2 horas) en ms
// Ahora hacemos lo mismo,pero usando el método setHours
let now = new Date();
now.setHours(now.getHours() - 2);
```

Cuando queremos imprimir la fecha, tenemos métodos que nos la devuelven en diferentes formatos:

```
let now = new Date();

console.log(now.toString());
console.log(now.toISOString()); // Imprime 2016-06-26T18:00:31.246Z
console.log(now.toUTCString()); // Imprime Sun, 26 Jun 2016 18:02:48 GMT
console.log(now.toDateString()); // Imprime Sun Jun 26 2016
console.log(now.toLocaleDateString()); // Imprime 26/6/2016
console.log(now.toTimeString()); // Imprime 20:00:31 GMT+0200 (CEST)
console.log(now.toLocaleTimeString()); // Imprime 20:00:31
```

Autor: Arturo Bernal Mayordomo

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 estos modificadores de dos formas:

```
let reg  = new RegExp("^[0-9]{2}","gi");
let 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*.

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

En este caso, "reg" se encuentra en "Hello, I'm using **reg**ular 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, /**b**case**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.

```
let str = "I am amazed in America";
let 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

let 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.

```
let str = "I am amazed in America";
let 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:

```
let str = "I am amazed in America";
let reg = /am/gi;
let 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()**.

```
let 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.

```
let 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"
```