FECHAS JAVASCRIPT

<https://lenguajejs.com/p/javascript/fundamentos/fechas>

En muchas ocasiones necesitaremos guardar o trabajar con fechas en nuestros programas. Una fecha tiene datos mixtos: día, mes y año, pero también puede ser más precisa y tener hora, minutos y/o segundos. Además, la hora puede estar en varios formatos.

Toda esta información no se podría guardar en una sola variable numérica, y tampoco es apropiado guardar en formato de texto, porque luego no podríamos hacer cálculos de fechas con ella. Para ello, tenemos un objeto llamado **Date** que nos vendrá perfecto para estos casos.

**¿Qué es el tipo de dato Date?**

Javascript nos provee de un tipo de dato llamado **Date**, con el que podemos trabajar facilmente con fechas de forma nativa y práctica. Sin embargo, trabajar con fechas no es fácil y la primera vez que tenemos que hacerlo es muy fácil equivocarse si no tenemos claros algunos conceptos.

Lo primero es ver los constructores para saber como podemos construir una variable de tipo **Date**:

| Constructor | Descripción |
| --- | --- |
| DATE **new Date()** | Obtiene la fecha del momento actual. |
| DATE **new Date(str)** | Convierte el texto con formato **YYYY/MM/DD HH:MM:SS** a fecha. |
| DATE **new Date(num)** | Convierte el número **num**, en formato **Tiempo UNIX**, a fecha UTC. |
| DATE **new Date(y, m, d, h, min, s, ms)** | Crea una fecha UTC a partir de componentes numéricos**\***. |

Podemos utilizar estas cuatro formas para crear fechas en Javascript. Observa que en algunos casos se menciona **fecha UTC**. De momento, vamos a obviar esta parte y más adelante profundizaremos en ella. Veamos algunos ejemplos para crear fechas con estos 4 constructores:

// Obtenemos la fecha actual y la guardamos en la variable f

var f = new Date();

// Obtenemos la fecha 30 de Enero de 2018, a las 23h 30m 14seg

var f = new Date('2018/01/30 23:30:14');

// Obtenemos la fecha del juicio final a partir de un timestamp o Tiempo UNIX

new Date(872817240000);

// Creamos una fecha pasando cada uno de sus componentes numéricos\*

new Date(y, m, d, h, min, s, ms);

**OJO**: Mucho cuidado con los **«componentes numéricos»** mencionados en el último ejemplo. Si utilizamos el formato **new Date(y, m, d, h, min, s, ms)**, hay que saber que, como mínimo, los parámetros **y** (*año*) y **m** (*mes*) son **obligatorios**, el resto son parámetros opcionales.

Además, si utilizamos este esquema, hay que tener en cuenta que hay ciertas características especiales:

* El parámetro **m** (*mes*) se proporciona con valores no reales que se recalcularán. Es decir, si indicamos un **1** nos referimos a **febrero** y no a **enero**. Si queremos referirnos a **enero**tenemos que indicar un **0**.
* Si indicamos **valores negativos** contabilizamos hacia atrás. Por ejemplo, con **2018, -1**estaríamos indicando **diciembre de 2017**. De la misma forma, **2018, 12** haría referencia a **enero de 2019**. Lo mismo ocurre con otros parámetros; **2018, 0, 32** haría referencia al 1 de febrero de 2018.

**Tiempo UNIX**

El **Tiempo UNIX** (*o UNIX timestamp*) es un formato numérico utilizado para calcular una fecha en UNIX. Es una forma poco práctica y legible para humanos, pero muy eficiente en términos informáticos. Se trata de un número que representa la cantidad de **segundos** transcurridos desde la fecha **1/1/1970, a las 00:00:00**.

Así pues, siendo números, resulta muy fácil trabajar y operar con ellos. Una fecha **A** y una fecha posterior **B**, si hacemos **B - A** nos devuelve el número de segundos transcurridos entre ambas fechas, con lo que se podría sacar la diferencia de tiempo.

No obstante, el **Tiempo UNIX** sirve para trabajar con fechas a bajo nivel. Si lo deseas, al final del tema encontrarás una tabla con librerías más cómodas y prácticas para trabajar con fechas en Javascript.

Existen dos métodos que se pueden utilizar para crear fechas, al margen de los constructores anteriores, sólo que estos devuelven directamente el **Tiempo UNIX** de la fecha especificada:

| Método | Descripción |
| --- | --- |
| NUMBER **Date.now()** | Devuelve el **Tiempo UNIX** de la fecha actual. Equivalente a **+new Date()**. |
| NUMBER **Date.parse(str)** | Convierte un STRING de fecha a **Tiempo UNIX**. Equivalente a **+new Date(str)**. |

Veamos algunos ejemplos aplicados para entenderlos:

// Estas tres operaciones son equivalentes

var f = Date.now();

var f = + new Date();

var f = new Date().getTime();

// Estas tres operaciones son equivalentes

var f = Date.parse('2018/10/30');

var f = + new Date('2018/10/30');

var f = new Date('2018/10/30').getTime();

* En el primer caso, utilizamos directamente los métodos estáticos **now()** y **parse()** para obtener el número con el **tiempo Unix**.
* En el segundo caso, creamos un objeto Date con **new Date()**, lo que devuelve una fecha. Sin embargo, en Javascript, podemos preceder a esa fecha con el símbolo **+**, obligándolo a evaluarlo de forma numérica, lo que hace que obtenga el **tiempo Unix** a partir de la fecha.
* En el tercer caso, escribimos en una línea dos acciones: crear la variable de fecha con **new Date()** y posteriormente, sobre esa fecha, hacemos un **getTime()**, que nos devuelve el **tiempo Unix** de un Date, como veremos más adelante.

**Getters: Obtener fechas**

Una vez hemos creado una fecha y tenemos el objeto DATE, podemos trabajar muy facilmente con estas variables a través de sus sencillos métodos. Los siguientes son una lista de **getters**, funciones para obtener información, sobre la fecha almacenada:

| Método | Descripción |
| --- | --- |
| NUMBER **.getDay()** | Devuelve el día de la semana: **OJO:** **0** Domingo, **6** Sábado. |
| NUMBER **.getFullYear()** | Devuelve el año con 4 cifras. |
| NUMBER **.getMonth()** | Devuelve la representación interna del mes. **OJO:** **0** Enero - **11** Diciembre. |
| NUMBER **.getDate()** | Devuelve el día del mes. |
| NUMBER **.getHours()** | Devuelve la hora. **OJO:** Formato militar; **23** en lugar de **11**. |
| NUMBER **.getMinutes()** | Devuelve los minutos. |
| NUMBER **.getSeconds()** | Devuelve los segundos. |
| NUMBER **.getMilliseconds()** | Devuelve los milisegundos. |
| NUMBER **.getTime()** | Devuelve el [UNIX Timestamp](https://es.wikipedia.org/wiki/Tiempo_Unix): segundos transcurridos desde **1/1/1970**. |
| NUMBER **.getTimezoneOffset()** | Diferencia horaria (*en min*) de la hora local respecto a UTC (ver más adelante). |

Algunos ejemplos del uso de estos métodos serían los siguientes:

var f = new Date('2018/01/30 15:30:10.999');

f.getDay(); // 2 (Martes)

f.getDate(); // 30

f.getMonth(); // 0 (Enero)

f.getFullYear(); // 2018

f.getHours(); // 15

f.getMinutes(); // 30

f.getSeconds(); // 10

f.getMilliseconds(); // 999

f.getTimezoneOffset(); // 0

f.getTime(); // 1517326210999 (Tiempo Unix)

Observa que Javascript no tiene forma de devolver, por ejemplo, el día de la semana o el mes en formato de texto. Sin embargo, ello se puede hacer de forma muy **sencilla** utilizando un array:

var f = new Date();

const MESES = ['Enero', 'Febrero', 'Marzo', 'Abril', 'Mayo', 'Junio', 'Julio',

'Agosto', 'Septiembre', 'Octubre', 'Noviembre', 'Diciembre'];

MESES[f.getMonth()]; // Devuelve el mes actual en formato de texto

**Curiosidad**: El método **getYear()** se utilizaba para devolver la fecha con formato de 2 cifras. Se dejó de usar debido a la llegada del [efecto 2000](https://es.wikipedia.org/wiki/Problema_del_a%C3%B1o_2000), reemplazándose por el método **getFullYear()** que usa el formato de 4 cifras.

**Setters: Cambiar fechas**

De la misma forma que en el apartado anterior podemos obtener fechas específicas, con los **setters** podemos modificarlas o alterarlas.

| Método | Descripción |
| --- | --- |
| NUMBER **.setFullYear(year)** | Altera el año de la fecha, cambiándolo por **year**. Formato de 4 dígitos. |
| NUMBER **.setMonth(month)** | Altera el mes de la fecha, cambiándolo por **month**. **Ojo**: **0-11** (Ene-Dic). |
| NUMBER **.setDate(day)** | Altera el día de la fecha, cambiándolo por **day**. |
| NUMBER **.setHour(hour)** | Altera la hora de la fecha, cambiándola por **hour**. |
| NUMBER **.setMinutes(min)** | Altera los minutos de la fecha, cambiándolos por **min**. |
| NUMBER **.setSeconds(sec)** | Altera los segundos de la fecha, cambiándolos por **sec**. |
| NUMBER **.setMilliseconds(ms)** | Altera los milisegundos de la fecha, cambiándolos por **ms**. |

**Nota**: Además de cambiar la fecha del objeto en cuestión, estos métodos devuelven un NUMBERen **tiempo Unix**, con la fecha modificada.

De la misma forma que mencionamos anteriormente, hay que tener en cuenta que los valores pasados a estos métodos pueden recalcular fechas: **setMonth(0)** implica **Enero**, **setDate(0)** implica último día del mes anterior, **setDate(-1)** implica penúltimo día del mes anterior, etc...

var f = new Date('2018/01/30 15:30:10.999');

f.setDate(15); // Cambia a 15/01/2018 15:30:10.999 (Devuelve 1516030210999)

f.setMonth(1); // Cambia a 15/02/2018 15:30:10.999 (Devuelve 1518708610999)

f.setFullYear(2020); // Cambia a 15/02/2020 15:30:10.999 (Devuelve 1581780610999)

f.setHours(21); // Cambia a 15/02/2020 21:30:10.999 (Devuelve 1581802210999)

f.setMinutes(00); // Cambia a 15/02/2020 21:00:10.999 (Devuelve 1581800410999)

f.setSeconds(3); // Cambia a 15/02/2020 21:00:03.999 (Devuelve 1581800403999)

f.setMilliseconds(79); // Cambia a 15/02/2020 21:00:03.079 (Devuelve 1581800403079)

f.setTime(872817240000); // Cambia a 29/08/1997 02:14:00.000 (Devuelve 872817240000)

Ten en cuenta que los mismos métodos anteriores, soportan varios parámetros, para hacer más cómodo su utilización y no tener que estar usándolos uno por uno. Así por ejemplo, se puede usar **setFullYear()** para cambiar año, año y mes, o año, mes y día:

| Método | Descripción |
| --- | --- |
| NUMBER **.setFullYear(y, m, d)** | Altera el año, mes y día de una fecha. |
| NUMBER **.setMonth(m, d)** | Altera el mes y día de una fecha. |
| NUMBER **.setHour(h, m, s, ms)** | Altera la hora, minutos, segundos y milisegundos. |
| NUMBER **.setMinutes(m, s, ms)** | Altera los minutos, segundos y milisegundos. |
| NUMBER **.setSeconds(s, ms)** | Altera los segundos y milisegundos. |
| NUMBER **.setTime(ts)** | Establece una fecha a partir del **tiempo Unix** **ts**. |

Por último, también tenemos a nuestra disposición el método setter **setTime(ts)**, que nos permite establecer una fecha a partir de un **tiempo Unix** **ts**. Es el equivalente a hacer un **new Date(ts)**.

**Representación de fechas**

Otro detalle más delicado dentro del trabajo con fechas es a la hora de mostrar una fecha en un formato específico. Podemos utilizar el formato por defecto que nos ofrece Javascript, pero lo más habitual es que queramos hacerlo en otro diferente, utilizando nuestra configuración horaria, abreviaturas u otros detalles. En Javascript tenemos varios métodos (*muy limitados, eso sí*) para representar las fechas:

| Método | Descripción |
| --- | --- |
| **Formato por defecto** | **Fri Aug 24 2018 00:23:31 GMT+0100** |
| STRING **.toDateString()** | Devuelve formato sólo de fecha: **Fri Aug 24 2018** |
| STRING**.toLocaleDateString()** | Idem al anterior, pero en el formato regional actual: **24/8/2018** |
| STRING **.toTimeString()** | Devuelve formato sólo de hora: **00:23:24 GMT+0100** ... |
| STRING**.toLocaleTimeString()** | Idem al anterior, pero en el formato regional actual: **0:26:37** |
| STRING **.toISOString()** | Devuelve la fecha en el formato [ISO 8601](https://en.wikipedia.org/wiki/ISO_8601): **2018-08-23T23:27:29.380Z** |
| STRING **.toJSON()** | Idem al anterior, pero asegurándose que será compatible con JSON. |
| STRING **.toUTCString()** | Devuelve la fecha, utilizando UTC (*ver más adelante*). |

El método **toISOString()** devuelve un formato que intenta ser el estándar a la hora de manejar fechas en Internet o documentos en general. Básicamente, una regla mnemotécnica es pensar que se escriben desde magnitudes más altas (*año*) hasta las magnitudes más bajas (*milisegundos*). La fecha y la hora se separa con una **T** y cada componente con un **-** o un **:**dependiendo de si es fecha o es hora. Los milisegundos se separan siempre con **.**. La **Z** del final indica que es una fecha **UTC**.

Existen algunas librerías interesantes que permiten trabajar muy fácilmente con la representación de fechas o incluso otros detalles. Lo comentaremos un poco más adelante.

**UTC y zonas horarias**

Para facilitar las cosas, hasta ahora hemos obviado un tema bastante importante: las **zonas horarias**. Probablemente te habrás percatado de que podría ser necesario trabajar y manejar **diferencias horarias**, porque por ejemplo necesitamos utilizar diferentes horas locales de distintos lugares del planeta. Para manejar esto, Javascript permite indicar también la diferencia horaria respecto al [meridiano Greenwich](https://es.wikipedia.org/wiki/Meridiano_de_Greenwich), indicada como **GMT**.

Por ejemplo, podemos tener una fecha A **2018/01/01 15:30:00 GMT+0000**, y una fecha B **2018/01/02 18:30:00 GMT+0200**. La fecha A no tiene diferencia horaria (GMT+0000), sin embargo, la fecha B si la tiene (GMT+0200), por lo que en Javascript al hacer un **new Date(str)** de dicha fecha, se recalcula automáticamente y se guarda la fecha modificada y sin diferencia horaria. En este caso, se le restan 2 horas a la fecha B.

Es importante darse cuenta que, en Javascript, cuando usamos **new Date()** sin parámetros, obtendremos una fecha que **puede** incluir diferencias horarias:

var f = new Date(); // Mon Aug 27 2018 01:39:21 GMT+0100

f.getTimezoneOffset(); // -60 (1 hora menos)

Sin embargo, si hacemos uso de **new Date(str)**, pasándole una fecha como parámetro de texto, automáticamente se recalcula la diferencia horaria, sumando/restando las horas, y se obtiene finalmente la fecha con diferencia horaria **cero**:

var f = new Date('2018/01/30 23:15:30 GMT+0100');

f; // Tue Jan 30 2018 22:15:30 GMT+0000

En este caso, al recalcular y guardar la fecha/hora sin diferencias horarias, decimos que estamos utilizando el **Tiempo Universal Coordinado** o [UTC](https://es.wikipedia.org/wiki/Tiempo_universal_coordinado). También podemos utilizar el método estático **Date.UTC()** donde le podemos pasar los parámetros de los componentes numéricos de la fecha, tal cómo lo hacemos en el constructor **new Date(y, m, d, h, min, s, ms)** anteriormente mencionado.

Además de lo anterior, recuerda que también tenemos todos los métodos **Getters** y **Setters**mencionados en su versión UTC. Por ejemplo, en el caso de **getHours()** tenemos una versión **getUTCHours()** que devuelve las horas de acuerdo al **UTC** (*Tiempo Universal Coordinado*).

**Librerías alternativas para fechas**

En muchos casos, el objeto DATE de Javascript puede quedarse corto para el usuario, que echa mucho de menos algunas funciones específicas, mayor comodidad al trabajar con fechas, o que simplemente necesita un mayor control.

Existen múltiples librerías para trabajar con fechas, citamos algunas de ellas:

| Librería | Descripción | GitHub |
| --- | --- | --- |
| [Moment.js](https://momentjs.com/) | Manejo de fechas y horas en JS. | [moment/moment](https://github.com/moment/moment/) |
| [date-fns](https://date-fns.org/) | Moderna librería de fechas. | [date-fns/date-fns](https://github.com/date-fns/date-fns) |
| Day.js | Librería inmutable (2KB) alternativa a Moment.js | [iamkun/dayjs](https://github.com/iamkun/dayjs) |
| [js-joda](https://js-joda.github.io/js-joda/) | Librería inmutable de fechas para JS. | [js-joda/js-joda](https://github.com/js-joda/js-joda) |
| [Datejs](http://www.datejs.com/) | Librería alternativa para trabajar con fechas. | [datejs/Datejs](https://github.com/datejs/Datejs) |
| fecha | Librería ligera para formatear y parsear fechas. | [taylorhakes/fecha](https://github.com/taylorhakes/fecha) |

Con ellas, podrás trabajar con mayor comodidad o profundidad con fechas y horas en Javascript.