# UD3
# Utilización de los objetos predefinidos de JavaScript - Nativos

# Clasificación de Objetos Predefinidos en JavaScript

### Objetos Primitivos

Como hemos visto en la unidad anterior, en JavaScript tenemos los datos de **tipos primitivos** los cuales se clasifican básicamente en 5 tipos: String, Number, Bool, Undefined y Null:

```
//DATOS PRIMITIVOS
let cadena = "Hola Daw";
let pi = 3.14;
let bool = true;  //false
let noDefinidos = undefined;
let nulo = null;
        console.log(typeof cadena);       //string
        console.log(typeof pi);           //number
        console.log(typeof bool);         //boolean
        console.log(typeof noDefinidos);  //undefined
        console.log(typeof nulo);         //object
        console.log(typeof vector);       //object
```

→ Lectura recomendada: ¿Por qué “null” es de tipo “object”?

### Objetos Nativos

Luego por otro lado podemos definir los **objetos nativos**, los cuales no dependen del tipo del navegador y son los siguientes básicamente:

| String | Number | Boolean |
|---|---|---|
| Date | Math | RegExp |
| Array | Function | Object |

→ Lectura recomendada: ¿Qué diferencia hay entre una cadena primitiva “” y un objeto String?

### Objetos De Alto Nivel

Y luego tenemos otros objetos llamados “de alto nivel” lo cuales sí dependen del tipo de navegador (algunos son soportados por unos navegadores y otros no) los cuales se clasifican en:

| Window | Screen | Navigator |
|---|---|---|

| Location | History | Document |
|---|---|---|

## Constructores nativos

Muchos de los objetos predefinidos en JavaScript pueden ser instanciados mediante su constructor, y también asignando directamente un valor:

```
//1. String
let x1 = new String(); //No suele utilizarse
let y1 = "DAWCL"; // Se utiliza más frecuentemente

//2. Number:
let x2 = new Number(); //No utilizar
let y2 = 3.14; //Utilizar

//3. Boolean:
let x3 = new Boolean(); //No utilizar
let y3 = true; //Utilizar

//4. Array:
let x4 = new Array(); //No utilizar
let y4 = []; //Utilizar

//5. RegExp:
let x5 = new RegExp(); //No utilizar
let y5 = //; //Utilizar

//6. Function:
let x6 = new Function(); //No utilizar
let y6 = function (){}; //Utilizar

//7. Date:
let x7 = new Date();

//8. Math: no se puede declarar con "new" porque es un objeto global.

//9. Object:
let x8 = {};
```

**Importante:** Fíjese que el objeto **Math** es un objeto global y no requiere de inicialización.

## El objeto Math

Es un objeto que permite realizar operaciones matemática y no requiere de usar ningún constructor ni inicialización ya que este es un objeto glocal.

Algunas de las propiedades y métodos más comunes del objeto Math son los siguientes:

```
//PROPIEDADES:

// E, LN2, LN10, LOG2E, LOG10E, PI, SQRT1_2, SQRT2
let pi = Math.PI;
alert ("El número PI vale: "+ pi);

//MÉTODOS MÁS COMUNES:

//random(): Devuelve un número aleatorio entre 0 y 1
alert ("Número aleatorio: "+Math.random());

//max(<lista de valores>) // min (<lista de valores>)
alert ("Máximo de valores: "+Math.max(1,5,3,9,6));
alert ("Mínimo de valores: "+Math.min(1,5,3,9,6));

//round(<número>): redondeo al alza o a la baja (Según se aproxime)
let a = 4.3;
let b = 4.7;
let c = 4.5;
alert ("Redondeo de "+c+" es: "+Math.round(c));    //5

//ceil(<número>): redondea siempre al alza:
alert ("Redondeo de "+a+" es: "+Math.ceil(a));     //5

//floor(<número>): redondea siempre a la baja:
alert ("Redondeo de "+b+" es: "+Math.floor(b));    //4

//Número aleatorio entre 0 y 10:
let aleatorio = Math.floor(Math.random() * 11);
alert ("Aleatorio es: "+aleatorio);

//pow(<base>,<exponente>): devuelve el valor de la potencia
alert ("La potencia de 3 al cuadrado es: "+Math.pow(3,2));

//sqrt(<numero>): devuelve la raíz del número:
alert("Raíz de 36 es: "+Math.sqrt(36));

//abs(<número>): devuelve el valor absoluto de un número
alert ("Valor absoluto de -5: "+Math.abs(-5));

//Métodos trigonométricos:
//sin, cos, tan, asin, acos, atan, atan2
```

## El objeto Boolean

El objeto Boolean es un tipo de objeto el cual, como bien sabemos, puede tener 2 valores: true o false. Este es un objeto muy sencillo y principalmente vamos a comentar su método **Boolean()** el cual nos permite saber si un valor es equivalente a true o a false. En el siguiente código vamos a ver una lista de valores en los que como veremos, **todos los valores reales son true, y todos los no reales son false**:

```
//Función Boolean()
Alert (Boolean(10>9))     //  Tanto con la función Boolean como
alert (10>9);             //  sin ella el resultado es el mismo.

//Todos los valores reales son ciertos (true)
alert("100 es "+Boolean(100));   // Entero
alert("3.4 es "+Boolean(3.4));   // Decimales
alert("Hola es "+Boolean("Hola")); // Cualquier cadena de texto
alert("false "+Boolean("false"));  // Cualquier cadena de texto aunque sea false
alert("-15 es "+Boolean(-15));   // Enteros con signo
alert("5<6 es "+Boolean(5<6));   // Condición verdadera

//Todos los valores que no son reales son falsos
alert("0 es "+Boolean(0));
alert("-0 es "+Boolean(-0));
alert("undefined es "+Boolean(undefined));
alert("null es "+Boolean(null));
alert("false es "+Boolean(false));
alert("NaN es "+Boolean(NaN));

var g;
if (!g) // Es equivalente a g==undefined
{
      alert ("La g no está definida");  // Sí entraría en la condición
}
```

## El objeto String

```
//INSTANCIACIÓN:
let daw = "Desarrollo de aplicaciones web";
let dam = 'Desarrollo de aplicaciones multiplataforma';
let asir = "Administración de 'Sistemas Informáticos' en Red";
let smr = "Sistemas \"Microinformáticos\" y Redes";
let ciclos = new String("");


//CONCATENACIÓN DE CADENAS
```

```
ciclos += "Hay 3 ciclos de Grado Superior: \n";
ciclos += daw + ", " + dam + " y " + asir + "\n";
ciclos += " y 1 ciclo de Grado Medio: \n";
ciclos += smr;
alert (ciclos);



//PROPIEDADES: Realmente solo existe una propiedad significativa.
//length: devuelve la longitud de una cadena
alert ("La longitud de la cadena ciclos es: "+ciclos.length);

//////////////MÉTODOS:

//BÚSQUEDA
//CharAt(<num>): devuelve el carácter de una posición.
alert ("El carácter 16 de la cadena ciclos es: " + ciclos.charAt(16));

//indexOf(<caracter>): devuelve la primera posición de un carácter en una cadena.
alert ("La primera aparición de la letra a en ciclos es: "+ciclos.indexOf("a"));

//lastIndexOf(<carácter>): devuelve la última posición de un carácter en una cadena.
alert ("La última aparición de la letra a en ciclos es: "+ciclos.lastIndexOf("a"));

//search(<cadena|expresión>): buscar una cadena o expresión regular en otra cadena.
Devuelve la posición de la primera ocurrencia, si no devuelve -1.
alert ("Busca la cadena 'web' en la variable daw: "+daw.search("web"));

//match (<expresión regular>): busca una expresión regular en otra cadena. Devuelve
un objeto con 2 resultados/propiedades:
        index – la posición de la cadena encontrado.

        Input – la subcadena encontrada.
let cadena = "Fama es un programa de bailarines";
let result = str.match( /fama/i );      //   /i significa ignore case
alert( result[0] );     // Fama (the match)
alert( result.index );  // 0
alert( result.input );  // "Fama es un programa de bailarines" (la cadena de texto)

//COMPARACIÓN:
//localeCompare(<cadena>): devuelve -1 (antes), 0 (igual), 1 (después)
alert ("¿Es daw menor que dam? "+ daw.localeCompare(dam));  // Compara carácter a
carácter cuál es mayor o menor alfabéticamente (en orden secuencial).

//INCLUYE, EMPIEZA O TERMINA
//includes(<cadena>): devuelve true si la cadena incluye el parámetro.
alert ("¿Incluye 'aplicaciones' daw? "+daw.includes("aplicaciones"));
//startsWith(<cadena>): devuelve true si la cadena empieza con el parámetro.
```

```
alert ("Empieza daw con la palabra 'aplicaciones'?
"+daw.startsWith("aplicaciones"));
//endsWith(<cadena>): devuelve true si la cadena finaliza con el parámetro.
alert ("Acaba daw con la palabra 'aplicaciones'? "+daw.endsWith("aplicaciones"));

//CONCATENACIÓN Y REPETICIÓN
//concat(<cadena>): concatena a la cadena el parámetro:
alert ("Concatena daw con dam \n" + daw.concat(dam));
//repeat(<número>): repetir la cadena el número de veces que aparece como parámetro
alert ("Repetir daw \n" + daw.repeat(3));


//EXTRACCIÓN
//slice(<posición inicial>,<posición final>): devuelve la cadena comprendida entre
ambas posiciones:
alert ("La cadena que hay entre el 0 y el 1 en daw es: " + daw.slice(0,1));

//substring(<posición inicial>,<posición final>): devuelve la cadena comprendida
entre ambas posiciones:
alert ("La cadena que hay entre el 0 y el 1 en daw es: "+ daw.substring(0,1));

//substr(<posición inicial>,<número de caracteres>): devuelve la cadena comprendida
entre la posición inicial y el número de caracteres:
alert ("La cadena de 5 caracteres que hay a partir del carácter 7 es:
"+daw.substr(5,7));

//split(<carácter>,[<número de elementos]): divide la cadena en un array de
subcadenas separadas por el carácter del primer parámetro. El segundo parámetro
se utiliza para delimitar el número de elementos que contiene el array devuelto.
alert("La cadena daw separada por espacios es: "+daw.split(" ",2));
      // Resultado:  [La, cadena]

//trim(<cadena>): extrae los caracteres de la cadena eliminando los espacios del
principio y del final.
alert ("La cadena sin espacios quedaría: \n"+"Hola, caracola".trim());



//MAYÚSCULAS Y MINÚSCULAS:
//toLowerCase(): devuelve la cadena en minúsculas:
alert (daw.toLowerCase());
//toUpperCase(): devuelve la cadena en mayúsculas:
alert (daw.toUpperCase());

//MÉTODOS ESPECIALES:
//toString(): devuelve el valor del objeto String.
//valueOf: devuelve el valor primitivo del objeto. La única diferencia respecto a
toString() es que si el texto a convertir a String es “null”, toString() lanzaría
una excepción, pero valueOf() devuelve una cadena “null”;
```

## xEl objeto RexExp

Los **objetos RexExp** sirven para comprobar si una cadena de texto sigue un determinado patrón, o si contiene unos caracteres determinados.  Se emplea, por ejemplo para comprobar en un formulario si el texto pasado por el usuario es un e-mail, o un número de teléfono, etc.

La sintaxis que siguen son la siguiente:

```
/patrón/modificadores;
```

**Instanciar un objeto RegExp**

Para declarar un objeto RegExp podemos hacerlo simplemente asignando a una variable una expresión regular, tal como en el ejemplo anterior, o mediante el método general de crear objetos:

```
let reg = new RegExp("^IES, "i");  //Empiece por IES,ies,IeS,etc.
let reg = /^IES/i ;
let reg = /^IES/ ;     // Sin ningún modificador
```

Para ver el funcionamiento básico de las expresiones regulares veremos los métodos **test** y **match**:

- ***regexObject*.test( *String* )**: método que devuelve **true** o **false** si la cadena de texto cumple el patrón.

- **string.match( RegExp )**: método del objeto String que hace una búsqueda de la cadena contra un patrón, y devuelve un array con los resultados encontrados, o el valor primitivo **null** en caso de que no encuentre ninguna. Es muy importante mencionar que para que el método **match** devuelva todas las ocurrencias debemos de usar el **modificador "g".**  El método match, a su vez nos devuelve el array con 2 propiedades:
    - **Index:** la posición del elemento encontrado.
    - **Input:** la cadena de texto donde se ha encontrado.

Veamos un ejemplo sencillo de los métodos test y match:

```
var patron1 = /ioc/i
var patron2 = /ioc/
var cadena="Curso de JavaScript del IOC";

console.log(patron1.test(cadena));
console.log(patron2.test(cadena));

var res = cadena.match(patron1);
console.log(res.length); //1, una sola ocurrencia
console.log(res[0]);     //IOC
```

Ahora veremos otro ejemplo en el que tenemos 2 elementos encontrados del patrón con **match:**

```
var patron = /Q.ijote/ig ;
var cadena = "Don Quijote veía molinos gigantes, Quijote";
var array = cadena.match(patron);
console.log("Posicion encontrados: " + array.index);          //undefined
console.log("Cadena donde se ha encontrado: " + array.input);  //undefined
console.log("Longitud: " + array.length);                      //2
array.map( function(elemento, indice){
      console.log("Elemento "+ indice +": " + elemento);          //Quijote
      console.log("Posicion "+ indice +": " + elemento.index);    //undefined
      return elemento;
});
```

A la hora de usar **test** o **match** para verificar si una cadena cumple un patrón o no, debemos usar **test** ya que este es entre un 30% y un 60% más rápido, según el navegador donde se ejecute.

Respecto a las expresiones regulares y debido a su complejidad lo más cómodo es tener una lista de las expresiones regulares para los casos más comunes, tales como comprobar direcciones web, e-mail, núm teléfonos, fechas, etc.

### Búsqueda y Reemplazo de Cadenas

Para efectuar las búsquedas y reemplazos, este objeto tiene varios métodos, algunos de los cuales son iguales que para las cadenas de texto:

**cadena.search( regexp )** → Comprueba si la cadena se ajusta al patrón, en tal caso **devuelve la posición** del patrón encontrado. **Search** solo devuelve la posición de la primera coincidencia. Si quisiéramos buscar más ocurrencias usaríamos el método **match** como hemos visto anteriormente.

**¡¡¡ Si no encuentra el patrón, devuelve -1 !!!**

```
let str = "Me gustan los catalanes porque hacen cosas";
console.log( str.search( /cata/i ) ); // 14 (la primera posicion)
```

**cadena.replace(regexp , reemplazar )** → Reemplaza el trozo de cadena que se ajusta a la expresión regular por la cadena que se pasa como segundo argumento (reemplazar).

```
console.log('12-34-56'.replace("-", ":")) // 12:34-56
```

Si quisiéramos reemplazar todas las coincidencias usaríamos el modificador **"g"**:

```
console.log( '12-34-56'.replace( /-/g, ":" ) )  // 12:34:56
```

### Separar Cadenas

**cadena.split( regexp )** → Devuelve un array en el que la cadena se ha separado según las coincidencias con la expresión regular.

```
console.log('12-34-56'.split(/-/))   // [12, 34, 56]
```

### Modificadores

Algunas de los **modificadores** y **metacaracteres** más usados en RexExp se pueden resumir en los siguientes:

- ^ : Representa el carácter con el que debe "empezar" la cadena. (Acento circunflejo)

```
console.log(/a/.test("blah"));      // true
console.log(/^a/.test("blah"));     // false
console.log(/^a/.test("aaaablah"));   // true
```

- **$ :** Representa el carácter con el que debe de terminar la cadena.

```
console.log(/a/.test("blah"));      // true
console.log(/a$/.test("blah"));     // false
console.log(/a$/.test("aaaabla"));   // true
```

- . : El punto se considera como sustituto de cualquier carácter ( 1 solo carácter).

```
var patron = /Q.ijote/;
var cadena = "Don Quijote veía molinos gigantes";
console.log(cadena.search(patron));       // 4
```

- **\ :** Con la barra invertida se nos ofrece determinar determinados tipos de caracteres. Por ejemplo:
  - **\s :** un espacio en blanco.
  - **\d :** un dígito.
  - **\n :** un salto de línea.
  - **\t :** un tabulador
  - **\w:** representa cualquier carácter alfanumérico (incluidos los guiones bajos _).
  - **\r:** representa el "retorno de carro".

```
var digitoRodeadoDeEspacios= /\s\d\s/;
console.log("1a 2 3d".search(digitoRodeadoDeEspacios));     // 2
```

- **[ ] :** Los corchetes lo que nos dice es si se contiene el grupo de caracteres que referencia.

```
var llaveOAsterisco = /[{*]/;
var cadena = "Hola*, cómo estás?";
console.log(cadena.search(llaveOAsterisco));    // 4
```

- **([B-D]|[F-H]|[J-N]|[P-T]|[V-Z]) :** Otra forma de usar los corchetes es con guiones (-), lo que hace es agrupar conjuntos de caracteres. Por ejemplo, en este caso indica cualquier carácter entre la B mayúscula y la Z mayúscula, excepto las vocales.
- **[A-Z] :** cualquier carácter entre la A mayúscula y la Z mayúscula.

- **( ) :** Los paréntesis son parecidos a los corchetes pero normalmente se utilizan junto a la | **(barra)** para discernir entre diversos valores.

```
var holyCow = /(sacred|holy) (cow|bovine|bull|taurus)/i;
console.log(holyCow.test("Sacred bovine!"));    // true
console.log(holyCow.test("holy bull!"));        // true
console.log(holyCow.test("sacred holy!"));      // false
console.log(holyCow.test("sacred taurus!"));    // true
```

- **? :** El signo de interrogación especifica que una parte de la búsqueda es opcional. En conjunto con los paréntesis, permite especificar que un conjunto mayor de caracteres es opcional.

```
var nov = /Nov(\.|iembre|ember)?/;
console.log(nov.test("Nov"));              // true
console.log(nov.test("Nov."));             // true
console.log(nov.test("Noviembre"));        // true
console.log(nov.test("November"));         // true
```

- **{ } :** Comúnmente las llaves son caracteres literales cuando se utilizan por separado en una expresión regular. Para que adquieran su función de metacaracteres es necesario que encierren uno o varios números separados por coma y que estén colocados a la derecha de otra expresión regular de la siguiente forma:
  - **{1,2} :** Mínimo uno , máximo 2 caracteres.
  - **{4} :** Cuatro dígitos.

```
var fecha = /^\d{2}\/\d{2}\/\d{2,4}$/;
console.log(fecha.test("05/05/1982"));    // true
console.log(fecha.test("05/05/82"));      // true
```

- *** :** El asterisco sirve para encontrar algo que se encuentra repetido **0 o más veces**. Por ejemplo:

```
var exp = /[a-zA-Z]\d*/;
console.log(exp.test("A"));         // true
console.log(exp.test("B0"));        // true
console.log(exp.test("c01"));       // true
console.log(exp.test("abc01234"));  // true
console.log(exp.test("01234"));     // false
```

- **+ :** Se utiliza para encontrar una cadena que se encuentre repetida **una o más veces**. A diferencia del asterisco, la expresión [a-zA-Z]\d+ encontrará "H1" pero no encontrará "H".

```
var exp = /[a-zA-Z]\d+/;
console.log(exp.test("A"));         // false
console.log(exp.test("B0"));        // true
console.log(exp.test("c01"));       // true
console.log(exp.test("abc01234")); // true
console.log(exp.test("01234"));     // false
```

Vamos a ver un ejemplo “heavy metal”: Vamos a ver como controlar que una fecha esté correctamente introducida por un usuario con el formato dd/mm/yyyy:

```
regexp = /^(3[01]|[12][0-9]|0?[1-9])\/(1[0-2]|0?[1-9])\/([0-9]{2})?[0-9]{2}$/;
console.log(regexp.test( "01/01/2016"));  // true
console.log(regexp.test( "1/01/2016"));  // true
console.log(regexp.test( "01/1/16"));  // true
console.log(regexp.test( "01/01/16"));  // true
console.log(regexp.test( "IES"));  // false
```

→ **Curiosidad:** Existe una página en la que podemos hacer pruebas con expresiones regulares de forma online. https://www.regular-expressions.info/javascriptexample.html

# El objeto Date

Una instancia de un objeto **Date** representa un determinado instante de tiempo. Este tipo de objetos no solo es usado para **fechas** sino que también se utiliza para **horas**.

### Inicializar objetos Date

Para **inicializar** un objeto de tipo **Date** lo realizaremos de la siguiente forma:

```
let fecha = new Date();
```

Internamente, los objetos **Date** son expresados en **milisegundos** desde el **1 de Enero de 1970**.

Si se está familiarizado con fechas en Unix, o recibimos una fecha desde un servidor Unix, debemos de tener en cuenta que normalmente las fechas vienen expresadas en segundos, por lo que deberemos de instanciar nuestro objeto Date en JavaScript de la siguiente forma:

```
const timestamp = 1530826365;
```

```
let fecha = new Date(timestamp * 1000);
```

Si lo que queremos es instanciar nuestro objeto Date con una fecha/tiempo concreto se nos da mucha flexibilidad:

```
new Date('2018-07-22')
new Date('2018-07') // 1 de Julio 2018, 00:00:00
new Date('2018') // 1 de Julio 2018, 00:00:00
new Date('07/22/2018')
new Date('2018/07/22')
new Date('2018/7/22')
new Date('July 22, 2018')
new Date('July 22, 2018 07:22:13')
new Date('2018-07-22 07:22:13')
new Date('2018-07-22T07:22:13')
new Date('25 March 2018')
new Date('25 Mar 2018')
new Date('25 March, 2018')
new Date('March 25, 2018')
new Date('March 25 2018')
new Date('March 2018') //1 de Marzo de 2018, 00:00:00
new Date('2018 March') //1 de Marzo de 2018, 00:00:00
new Date('2018 MARCH') //1 de Marzo de 2018, 00:00:00
new Date('2018 march') //1 de Marzo de 2018, 00:00:00
```

Si lo que queremos es obtener directamente un **timestamp en milisegundos** como valor podemos usar el método **Date.parse()** :

```
Date.parse('2018-07-22')
Date.parse('2018-07') // 1 de Julio de 2018, 00:00:00
Date.parse('2018') // 1 de Julio de 2018, 00:00:00
Date.parse('07/22/2018')
Date.parse('2018/07/22')
Date.parse('2018/7/22')
Date.parse('July 22, 2018')
Date.parse('July 22, 2018 07:22:13')
Date.parse('2018-07-22 07:22:13')
Date.parse('2018-07-22T07:22:13')
```

Por último, también podríamos inicializar nuestro objeto Date haciendo uso de parámetros siempre ajustándonos al siguiente orden: **año , mes , día , hora, minutos, segundos** y **milisegundos.**

```
new Date(2018, 6, 22, 7, 22, 13, 0)
new Date(2018, 6, 22)
```

**Importante:** En cualquiera de las formas de inicializar un objeto Date mencionados, el objeto Date resultante siempre será relativo a la **timezone (zona horaria)** de tu ordenador. Esto significa que 2 ordenadores pueden tener diferentes valores para el mismo objeto.

### Zona horaria (Timezone)

Cuando se inicializa un objeto Date se le puede pasar la zona horaria a la que será ajustado. Si no se especifica ninguna, se ajustará a tu zona horaria local (de tu equipo).

Se puede especificar las zonas horarias de **2 formas** : Añadiendo esta en formato +HORA, o el nombre de la zona horaria.

```
new Date('July 22, 2018 07:22:13 +0700')
new Date('July 22, 2018 07:22:13 (CET)')
```

**Importante:**

- Si se especifica una zona horaria incorrecta entre paréntesis, JavaScript usará la de por defecto sin mostrar ningún mensaje de error.
- En cambio, si se especifica un número incorrecto como zona horaria, JavaScript mostrará un mensaje de error con el texto: "Invalid Date".

➞ Puedes buscar los nombres de las zonas horarias en el siguiente enlace: zonas horarias

### Formateado y Conversión de objetos Date

Existen multitud de métodos con los que podemos trabajar a la hora de aplicar un determinado formato o convertir nuestros objetos Date a un determinado formato. Algunos de ellos son:

```
const date = new Date('July 22, 2018 07:22:13')
date.toString() // "Sun Jul 22 2018 07:22:13 GMT+0200 (Central European Summer
Time)"
date.toTimeString() //"07:22:13 GMT+0200 (Central European Summer Time)"
date.toUTCString() //"Sun, 22 Jul 2018 05:22:13 GMT"
date.toDateString() //"Sun Jul 22 2018"
date.toISOString() //"2018-07-22T05:22:13.000Z" (ISO 8601 format)
date.toLocaleString() //"22/07/2018, 07:22:13"
date.toLocaleTimeString()     //"07:22:13"
date.getTime() //1532236933000
```

### Métodos Get para objetos Date

Algunos de los principales métodos de los objetos Date para obtener sus valores son:

```
const date = new Date('July 22, 2018 07:22:13')
date.getDate() //22  -- el día del mes.
date.getDay() //0 (0 means sunday, 1 means monday..)
date.getFullYear() //2018
date.getMonth() //6 (starts from 0)
date.getHours() //7
date.getMinutes() //22
```

```
date.getSeconds() //13
date.getMilliseconds() //0 (not specified)
date.getTime() //1532236933000
```

### Métodos Set para objetos Date

De la misma forma que los métodos Get, también podemos modificar puntualmente valores de objetos Date haciendo uso de los método Set de forma que podamos editar nuestro valores:

```
const date = new Date('July 22, 2018 07:22:13')
date.setDate(newValue)
date.setDay(newValue)
date.setFullYear(newValue) //note: avoid setYear(), it's deprecated
date.setMonth(newValue)
date.setHours(newValue)
date.setMinutes(newValue)
date.setSeconds(newValue)
date.setMilliseconds(newValue)
date.setTime(newValue)
date.setTimezoneOffset(newValue)
```

**Importante:** Los métodos **setDay** y **setMonth** empiezan a numerar por **el valor 0**. De esta forma, Marzo, por ejemplo, es el valor 2.  (Enero es el 0).

### Obtener la fecha/hora actual

Para obtener la fecha/hora actual basta con utilizar el siguiente método:

```
Date.now()
```

### Comparar 2 fechas

Se puede calcular la **diferencia entre 2 objetos Date** haciendo uso del método **Date.getTime() :**

```
const date1 = new Date('July 10, 2018 07:22:13')
const date2 = new Date('July 22, 2018 07:22:13')
const diff = date2.getTime() - date1.getTime() //diferencia en milisegundos
```

De la misma forma, se puede comprobar si 2 fechas son iguales :

```
const date1 = new Date('July 10, 2018 07:22:13')
const date2 = new Date('July 10, 2018 07:22:13')
if (date2.getTime() === date1.getTime()) {
  // fechas son iguales
}
```

**Overflow de valores**

Como JavaScript siempre trata de no generar ningún error y no dar ningún tipo de problemas (ironía) debemos de tener mucho cuidado que si por alguna razón nos pasamos en los valores de algunas de las cantidades a introducir en la instanciación de objetos Date, como por ejemplo, más horas que 24, JavaScript, automáticamente, sumaría un día más en nuestra fecha y ajustaría las horas sobrantes:

```
new Date(2018, 6, 40)  // Julio 10 2018
```

Lo mismo pasaría tanto con meses, horas, minutos, segundos y milisegundos.

**Formatos de Fecha según país:**

https://en.wikipedia.org/wiki/Date_format_by_country

## El objeto Number

- Normalmente no se suele utilizar ya que las variables numéricas se asignan directamente.

```
let  numero = new Number(3.18);
```

Algunos de los métodos más significativos son:

## Métodos

Number

| Método | Descripción |
|---|---|
| toExponential(x) | Convierte un número a su notación exponencial. |
| toFixed(x) | Formatea un número con x digitos decimales después del punto decimal. |
| toPrecision(x) | Formatea un número a la longitud x. |
| toString() | Convierte un objeto Number en una cadena.<br>•Si se pone 2 como parámetro se mostrará el número en binario.<br>•Si se pone 8 como parámetro se mostrará el número en octal.<br>•Si se pone 16 como parámetro se mostrará el número en hexadecimal. |
| valueOf() | Devuelve el valor primitivo de un objeto Number. |