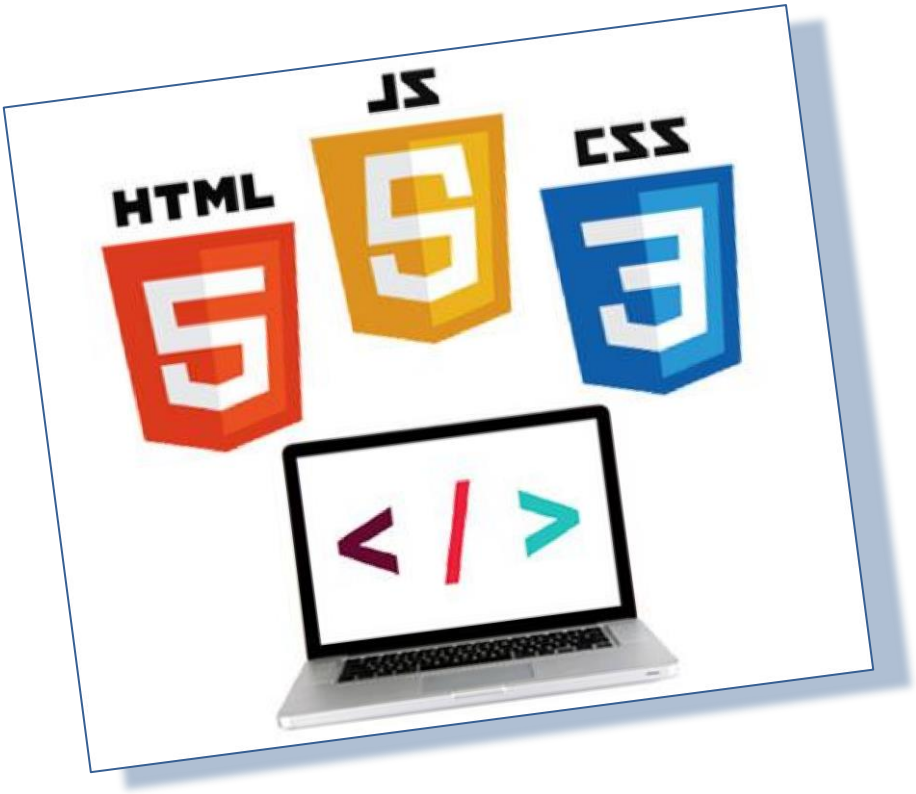
Desarrollo web en entorno cliente



Tema 4: Programación Orientada a Objetos y Objetos Nativos

1. Programación orientada a objetos

Aunque JavaScript no es un lenguaje orientado a objetos, sino más bien un lenguaje basado en prototipos, desde la versión ES2015(ECMAScript6) JavaScript permite el uso de clases y objetos, aunque con algunas diferencias respecto a lo que es la programación orientada a objetos de lenguajes como Java o PHP.

En los lenguajes de programación orientados a objetos basados en clases, un objeto no puede existir a menos que se haya definido primero la clase. En JavaScript, sin embargo, cada objeto se puede crear directamente, sin necesidad de definir antes una clase.

1.1. Propiedades de un objeto

Como se ha visto en los ejemplos del punto anterior, se puede acceder a las propiedades de un objeto con . o con []. Siguiendo el ejemplo anterior, las dos sentencias que vienen a continuación devolverían el mismo resultado:

```
console.log(casa.ciudad);
console.log(casa["ciudad"]);
```

Con la instrucción **for..in** se puede iterar sobre las propiedades de un objeto y acceder a ellas. Lo que devuelve cada iteración es el nombre de la propiedad a la que se está accediendo en ese momento, por lo que para acceder al valor de la misma habría que utilizar dicha propiedad como índice:

```
for (const propiedad in casa) {
  console.log(propiedad + ": " + casa[propiedad]);
}
```

1.2. Definición de clases

Como se ha indicado al comienzo de la unidad, JavaScript permite crear objetos sin clases definidas, pero también es posible seguir la estructura clásica de la programación orientada a objetos en cuanto a clases e instancias de las mismas.

Estas clases se definirán con la palabra reservada class. También se pueden ver definidas con la palabra function, aunque está es una solución anterior a ES2015 para suplir esa falta de orientación a objetos que tenía JavaScript.

Una clase se definiría como en el ejemplo que sigue:

```
class Casa {
  constructor(metros, direccion, ciudad) {
    this.metros = metros;
    this.direccion = direccion;
    this.ciudad = ciudad;
  }
  getDireccion() {
    return "Calle " + " " + this.direccion + " " + this.ciudad;
  }
}
```

Y, cuando se quiera crear una instancia de la misma(un objeto):

```
let casaPlaya = new Casa(65,"Miramar 5 4A", "Benidorm");
console.log(casaPlaya.getDireccion());
```

1.3. Variable this

En Javascript, dentro de una función se crea un nuevo contexto y this pasa a hacer referencia al mismo. Este problema es muy común en las funciones de manejo de eventos.

1.4. Herencia

Al igual que en otros leguajes orientados a objetos, en JavaScript una clase puede heredar de otra. Se hará mediante la palabra **extends** y heredará todos sus métodos y propiedades.

Esos métodos pueden ser sobrescritos en la clase hija aunque se podrá seguir accediendo a los métodos de la clase padre a través de la palabra reservada **super**, por ejemplo, al crear un constructor en la clase hija:

```
class CasaIndividual extends Casa{
    constructor(metros, direccion, ciudad, metrosJardin, piscina) {
      super(metros, direccion, ciudad);
      this.metrosJardin = metrosJardin;
      this.piscina = piscina;
    }
    getDireccion () {
        return super.getDireccion() + " y con " + this.metrosJardin +
" metros de jardín";
    }
    getPiscina() { return this.piscina; }
}
```

1.5. Métodos estáticos

Se puede usar sin instanciar la clase

ES2015 permite declarar métodos estáticos, que se llaman haciendo uso directo del nombre de la clase y, al no haber instancia de la misma, no tendrán acceso al objeto this.

```
class Casa {
    …
    static getTiposCalefaccion() {
        return ["gas natural", "electrica", "pellets"];
    }
}
console.log(Casa.getTiposCalefaccion()); //Devolverá los tipos
let casa = new Casa(68, "Los Cedros 25 4ºC", "Salamanca");
console.log(casa.getTiposCalefaccion()); //Dará error
```

Se utilizan habitualmente para crear funciones para una aplicación.

1.6. toString()

(texto)

Al convertir un objeto a string (por ejemplo al hacer un alert o al concatenarlo) internamente se está llamando al método .toString() del mismo, que devuelve [object Object]. Es decir, este método es el llamado de forma automática cuando un objeto tiene que ser representado como una cadena. Es posible sobrescribirlo para que devuelva lo que se quiera:

```
class Casa {
    …
    toString() {
        return "Esta casa tiene" + this.metros + "metros cuadrados";
    }
}
```

Este método también es el que se usará si se quiere ordenar una array de objetos (.sort() ordena alfabéticamente para lo que llama al método .toString() del objeto a ordenar).

2. Funciones globales

Javascript, de forma nativa, ofrece gran cantidad de funciones globales y objetos nativos. Este tipo de funciones y objetos son de uso muy extendido y facilitan el desarrollo de forma eficiente y clara. En este y siguientes apartados se van a ver funciones globales (las cuales ya se han visto anteriormente) y objetos predefinidos que facilitarán el trabajo (por ejemplo, al trabajar con fechas o expresiones regulares).

- **parseInt(valor):** Método que devuelve el valor pasado como parámetro convertido a entero o NaN si no es posible la conversión.
- **parseFloat(valor):** igual que el anterior pero devolviendo un número decimal.
- **Number(valor)**: convierte el valor a un número. Es como parseFloat pero más estricto y si no puede convertir todo el valor devuelve NaN.
- **String(valor)**: convierte el valor pasado a una cadena de texto. Si le pasamos un objeto llama al método .toString() del objeto.
- **isNaN(valor)**: devuelve true si el parámetro pasado no es un número o no puede convertirse en un número.
- **isFinite(valor)**: devuelve false si el número pasado como parámetro es infinito (o demasiado grande):

```
console.log( isFinite(5.78) ); // true
console.log( isFinite(5.78 / 0) ); // false
```

3. Objetos nativos u objetos predefinidos

Como ya se ha visto en temas anteriores, en Javascript casi todo son objetos. Algunos muy utilizados, que se verán en temas posteriores, son:

- window
- screen
- navigator
- location
- history

RELACIONADO CON EL NAVEGADOR

- document

RELACIONADO CON HTML

De estos, los cinco primeros se corresponden al modelo de objetos del navegador y el sexto se corresponde al modelo de objetos del documento. Todos permiten interactuar con el navegador para realizar distintas acciones, y se verán en detalle cuando se vea DOM y BOM.

A parte de objetos dependientes del navegador, existen objetos nativos que no dependen de aquel. Además de los tipos primitivos de número, cadena, booleano, undefined y null, Javascript, existen los objetos nativos Number, String, Boolean, Array, Function, Object, Math, Date y RegExp. Ya se ha visto que se puede crear un número usando su tipo primitivo o su objeto, pero que lo recomendable es usar los primitivos. Se declare un objeto de una manera u otra, siempre será un objeto y se tiene acceso a todas sus propiedades y métodos.

3.1. Number

El tipo de dato para cualquier número es Number, sea entero o decimal. Se le pueden aplicar los operadores aritméticos +, -, *, / y % y los unarios **++** y **--,** y existen los valores especiales **Infinity** y **-Infinity**. Además, existe la posibilidad de usar los operadores aritméticos junto al operador de asignación **=** (+=, -=, *=, /= y %=).

Algunos métodos propios de un objeto Number ya se han visto, como Number(cadena); para la conversión a número. Mas métodos útiles de los números son:

- **.toFixed(num)**: Método que redondea el número a los decimales indicados:

```
let numero =23.2376;
numero.toFixed(2); //devuelve 23.24
```

- **.toLocaleString()**: Método que devuelve el número convertido al formato local:

```
let numero2 = 23.76;
numero2.toLocaleString(); // devolverá '23,76'
```

Hay que tener cuidado al trabajar con decimales, ya que pueden dar problemas:

```
console.log(0.1 + 0.2) // imprime 0.30000000000000004
//Se puede evitar redondeando los resultados
```

3.2. String

Una cadena de texto se puede entrecomillar con comillas simples o dobles.

- Es posible escapar caracteres con el símbolo \. Por ejemplo, si se quiere mantener en una cadena las comillas como parte de la misma: *'Hola \'Mundo\''* devolverá *Hola 'Mundo'*.
- Para forzar la conversión a cadena se usa la función **String(valor)**. Por ejemplo, String(23) devolverá la cadena "23".

Algunos métodos y propiedades de las cadenas son:

- **.length:** devuelve la longitud de una cadena.
- **.charAt(posición)**: Devuelve el carácter que se encuentra en la posición indicada.
- **.substring(posInicio, posFin)**: Devuelve una cadena que contiene desde el carácter de la posición posInicio hasta el carácter de la posición posFin-1.
- **.substr(desde, N)**: Devuelve una cadena de N caracteres y cuyo primer carácter será el que se encuentre en la posición posInicio.
- **.replaceAll(palabraBuscar,** cadenaNueva**)**: Método que busca las ocurrencias de **palabraBuscar** en la cadena y devolverá una nueva con dichas ocurrencias sustituidas por cadenaNueva.
- **.toLocaleLowerCase(locale)**: Método que convierte toda la cadena a minúsculas. locale indica la localización que se utilizará para la conversión de acuerdo con cualquier correspondencia de mayúsculas y minúsculas.
- **.toLocaleUpperCase(locale)**: Método que convierte toda la cadena a mayúsculas.
- **.localeCompare(cadena)**: devuelve -1 si la cadena a que se aplica el método es anterior alfabéticamente a 'cadena', 1 si es posterior y 0 si ambas son iguales. Tiene en cuenta caracteres locales como acentos ñ, ç, etc.
- **.trim(cadena)**: Elimina los espacios en blanco a los extremos de la cadena.
- **.startsWith(caracteres)**: Devuelve true en caso de que el comienzo de la cadena coincida con el valor pasado como parámetro. Devolverá false en caso contrario.
- **.endsWith(cadena)**: Devuelve true en caso de que el final de la cadena coincida con el valor pasado como parámetro. Devolverá false en caso contrario.
- **.repeat(N)**: Concatenará N veces la cadena consigo misma.

También se puede trabajar con cadenas con los métodos ya vistos para arrays.

3.3. Boolean

Los valores booleanos son **true** y **false**. Para convertir algo a booleano se usar **Boolean(valor)** aunque también puede hacerse con la doble negación !! .

EJERCICIO 6

- Crea una función que:
  - Pida al usuario que introduzca notas separadas por comas.
  - Calcule la nota media de dichas notas y la muestre por consola.
  - Seguidamente mostrará de nuevo por consola la media pero con 1 solo decimal.
- Crea una función que devuelva el cubo de un número comprobando primero si el parámetro pasado es un número entero:
  - Si no es un entero o no es un número mostrará un alert indicando cuál es el problema y devolverá false.
  - En caso contrario mostrará el resultado de la operación.
- Implementa una función a la que se le pasa como parámetro un DNI y devolverá true en caso de que sea correcto y false en caso contrario. **La letra del DNI se calcula dividiendo la parte numérica del mismo entre 23 y cogiendo de la cadena 'TRWAGMYFPDXBNJZSQVHLCKE' la letra correspondiente al resto de la división.
- Crea una función que, a partir de una cadena de caracteres pasada como parámetro:
  - La pase a minúsculas.
  - Forme una nueva con los valores de las posiciones pares.
  - Busque en la nueva cadena la palabra "aviko".

3.4. Math

Proporciona constantes y métodos para trabajar con valores numéricos:

- constantes: Math define ciertas constantes que son de utilidad, entre ellas las más conocidas quizá sean .PI (número pi) y .SQRT2 (raíz cuadrada de 2):

```
console.log( Math.PI );// imprime 3.141592653589793
console.log( Math.SQRT2 );// imprime 1.4142135623730951
```

- **Math.round(x)**: redondea x al entero más cercano.

```
console.log( Math.round(3.21) ); // devolverá 3
console.log( Math.round(3.67) ); // devolverá 4
```

- **Math.floor(x)**: redondea x hacia abajo.
- **Math.ceil(x)**: redondea x hacia arriba.
- **Math.min(x1,x2,...)**: devuelve el número más bajo de los pasados.
- **Math.max(x1,x2,...)**: devuelve el número más alto de los pasados.
- **Math.pow(x, y)**: devuelve x elevado a y.
- **Math.abs(x)**: devuelve el valor absoluto de x.
- **Math.random()**: devuelve un número aleatorio entre 0 y 1 (no incluido).
- **Math.sqrt(x)**: devuelve la raíz cuadrada de x.

EJERCICIO 7

- Método Math.random():
  - Investiga de qué manera se puede utilizar el método Math.random() para que devuelva un número aleatorio en otro rango diferente a 0-1.
  - Crea una función que saque tres números aleatorios entre 1 y 100 y muestre por pantalla el mayor de ellos.
- Completa la función que has codificado en el primero punto del ejercicio 7 para que, después de la funcionalidad indicada en dicho ejercicio, haga lo siguiente:
  - Redondee cada nota y las muestre todas en un array.
  - Pregunte al usuario si quiere redondear hacia arriba las notas.
  - En caso de que su respuesta sea afirmativa, redondeará todas las notas hacia arriba y mostrará por consola la menor de ellas.
  - En caso contrario, lo hará hacia abajo y lo que se muestre por consola será el valor mas grande.
- Después del último redondeo, calculará de nuevo la nota media y la mostrará con 3 decimales.

3.5. Date

Objeto que se usará siempre que se quiera trabajar con fechas. Al crear una instancia de la clase se le puede pasar la fecha que se quiere crear o, sin parámetros, crear la fecha actual. Si se opta por pasarle la fecha a crear, se puede hacer pasándole:

- milisegundos, desde la fecha EPOCH.
- cadena de fecha, con distintos formatos permitidos.
- valor para año, mes (entre 0 y 11), día, hora, minutos, segundos, milisegundos.

Ejemplos:

```
let date1=new Date(); //Creará un objeto fecha con los datos de la fecha actual
let date2=new Date(1577836801000); // Creará un objeto para la fecha 1/1/2020
let date3=new Date('2018-07-30');
let date4=new Date('2018-07-30 05:30');
let date5=new Date('07-30-2020'); // formato MM-DD-AAAA
let date7=new Date('30-Ago-2020'); //también se podría indicar como Agosto
```

Cuando se indica la fecha en modo texto, como separador de los datos se puede usar -, / o espacio en blanco. Como separador de las horas, sin embargo, hay que usar : .

Cuando la fecha se introduce como parámetros numéricos (separados por ,) es posible poner valores fuera de rango que se sumarán al valor anterior.

```
let date=new Date(2020,7,41); // 10/09/2020 porque 41=31Ago+10Sep
let date=new Date(2020,7,0); // 31/07/2020 porque 0=0Ago=31Jul (el anterior)
let date=new Date(2020,7,-1); // 30/07/2020 porque -1=0Ago-1=31Jul-1=30Jul
```

Cabe destacar que el rango de los meses empieza en 0, por lo que hay que ser especialmente cuidadosos con ese dato.

Los siguientes métodos permiten obtener o modificar los distintos valores de una fecha:

- **getFullYear()/setFullYear()**: permite ver o cambiar el año de la fecha.
- **getMonth()/setMonth()**: devuelve/cambia el número de mes (0=Enero, 1=Febrero,…,11=Diciembre).
- **setDate(fecha)/getDate()**: devuelve/cambia el número de día.

```
let fecha=new Date('2020-07-30');
console.log( fecha.getDate() ); // imprimirá 30
fecha.setDate(-2); // El día pasará a ser el 28
```

- **getDay()**: devuelve el número de día de la semana (0->Dom, 1->Lun, …, 6->Sáb):

```
let fecha=new Date('2018-07-30');
console.log( fecha.getDay() ); // imprime 1
```

- **(get/set)Hours(),(get/set)Minutes(),(get/set)Seconds(),get/set)milliseconds()**: devuelve/cambia el número de la hora, minuto, segundo o milisegundo, respectivamente.
- **setTime(milisegundos)/getTime()**: establece/devuelve el número de milisegundos desde Epoch:

```
let fecha=new Date('2018-07-30');
console.log( fecha.getTime() ); // imprime
1532908800000
fecha.setTime(1332403882588);//Suma 1332403882588
milisegundos a epoch
```

Una fecha se puede mostrar en diferentes formatos, algunos de ellos a través de los siguientes métodos:

- **.toString()**: “Mon Jul 30 2018 02:00:00 GMT+0200 (CEST)”
- **.toUTCString()**: “Mon, 30 Jul 2018 00:00:00 GMT”
- **.toDateString()**: “Mon, 30 Jul 2018”
- **.toTimeString()**: “02:00:00 GMT+0200 (hora de verano de Europa central)”
- **.toISOString()**: “2018-07-30T00:00:00.000Z”
- **.toLocaleString()**: “30/7/2018 2:00:00”
- **.toLocaleDateString()**: “30/7/2018”
- **.toLocaleTimeString()**: “2:00:00”

Hay que tener en cuenta que cuando se copia una variable de tipo Date a otra variable, como pasa con cualquier objeto, ambas variables pasan a referenciar a la misma fecha. Por ello, si se modifica después la copia, ese cambio también se realizará en la variable original (porque ambas variables apuntan al mismo valor).

Por ello, para copiar una fecha lo más sencillo es crear una nueva pasándole el tiempo EPOCH de la que se quiere copiar:

```
let fecha=new Date('2018-07-30');
let otraFecha=new Date(fecha.getTime());
otraFecha.setDate(28);
console.log( fecha.getDate() ); // imprimirá 30
```

Por último, en cuanto a la comparación de fechas, es posible comparar dos objetos Date con los operadores >, >=, < y <= pero no con ==, ===, != ni !===. La mejor forma de comparar dos fechas es utilizando su método getTime() y el operador ===.

EJERCICIO 8

- ¿Qué es el formato EPOCH?
- Crea dos variables fechaNacimiento1 y fechaNacimiento2 que contengan las dos tu fecha de nacimiento. La primera se creará pasándole una cadena y la segunda pasándole año, mes y día.
- A partir de las fechas creadas en el punto anterior, implementa el código necesario para mostrar por consola el resultado de:
  - El día de la semana de una de ellas.
  - Modifica una de ellas para que contenga la fecha del cumpleaños de este año (cambia sólo el año).
  - El día de la semana de la fecha en este año.
  - Calcular el nº de días que han pasado desde el día que naciste hasta hoy.
- Crea una función que:
  - Cree una fecha con la hora de la cena de Nochevieja de un año cualquiera.
  - Recupere de dicha fecha un valor con el método getDate().
  - Asigne a la fecha creada, el valor recuperado en el punto anterior +1, a través del método setDate().
  - Muestre por consola la fecha.
  - Añade al ejercicio mediante un comentario la explicación del resultado mostrado por consola.
- Muestra la fecha de hoy y la hora actual en 5 formatos distintos.
- Comprueba si es mayor tu fecha de nacimiento o el 1 de enero de este año.
- Escribe una función que devuelva el número de días que tiene un mes concreto. Como parámetros se le pasarán el año y el mes de forma numérica(1: Enero, 2:Febrero, etc.)
- Crea una función que compare dos fechas llegadas por parámetros:
  - Las fechas serán objetos de tipo Date.
  - El texto que devolverá la función indicará que la primera es mayor que la segunda, que la segunda es mayor que la primera o que ambas son iguales.

4. Template literals

Desde ES2015 también está permitido poner una cadena entre ` (acento grave) y en ese caso se pueden poner dentro variables y expresiones que serán evaluadas al ponerlas dentro de **${}**. También se respetan los saltos de línea, tabuladores, etc que haya dentro. Ejemplo:

```
let edad=25;
console.log(`El usuario tiene: ${edad} años`); //Saldrá por consola:
El usuario tiene: 25 años
console.log(`El usuario tiene: \n ${edad} años`); //Y por consola:
El usuario tiene:
25 años
```