Javascript

Introducción

- **Incluir javascript en una página web**

  - El código Javascript va entre etiquetas <*script*>. Puede ponerse en el <*head*> o en el <*body*>. Funciona como cualquier otra etiqueta y el navegador la interpreta cuando llega a ella
  - Es posible poner el código directamente entre la etiqueta <*script*> y su etiqueta de finalización pero lo correcto es que esté en un fichero externo (con extensión **.js**) que cargamos mediante el atributo *src* de la etiqueta. Así conseguimos que la página HTML cargue más rápido

```
<script src="./scripts/main.js"></script>
```

Introducción

- **Mostrar información**

Javascript permite mostrar al usuario ventanas modales para pedirle o mostrarle información. Las funciones que lo hacen son:

  - window.alert(mensaje): Muestra en una ventana modal mensaje con un botón de Aceptar para cierra la ventana.
  - window.confirm(mensaje): Muestra en una ventana modal mensaje con botones de Aceptar y Cancelar. La función devuelve true o false en función del botón pulsado por el usuario.
  - window.prompt(mensaje [, valor predeterminado]): Muestra en una ventana modal mensaje y debajo tiene un campo donde el usuario puede escribir, junto con botones de Aceptar y Cancelar.

Introducción

- La función devuelve el valor introducido por el usuario como texto (es decir que si introduce 54 lo que se obtiene es “54”) o false si el usuario pulsa Cancelar.
- También se pueden escribir las funciones sin window. (es decir alert('Hola') en vez de window.alert('Hola')) ya que en Javascript todos los métodos y propiedades de los que no se indica de qué objeto son se ejecutan en el objeto window.

- Si queremos mostrar una información para depurar nuestro código no utilizaremos alert(mensaje) sino console.log(mensaje) o console.error(mensaje). Estas funciones muestran la información pero en la consola del navegador. La diferencia es que console.error la muestra como si fuera un error de Javascript.

Variables

Javascript es un lenguaje débilmente tipado. Esto significa que no se indica de qué tipo es una variable al declararla e incluso puede cambiar su tipo a lo largo de la ejecución del programa

Ni siquiera estamos obligados a declarar una variable antes de usarla, aunque es recomendable para evitar errores que nos costará depurar.

```
let miVariable;          // declaro miVariable y como no se asigno un valor valdrá undefined
miVariable='Hola';       // ahora su valor es 'Hola', por tanto contiene una cadena de texto
miVariable=34;           // pero ahora contiene un número
miVariable=[3, 45, 2];   // y ahora un array
miVariable=undefined;    // para volver a valer el valor especial undefined
```

Variables

- Podemos hacer que se produzca un error si no declaramos una variable incluyendo al principio de nuestro código la instrucción

```
'use strict'
```

Variables

- Las variables de declaran con **let** (lo recomendado desde ES2015), aunque también pueden declararse con **var**. La diferencia es que con *let* la variable sólo existe en el bloque en que se declara mientras que con *var* la variable existe en toda la función en que se declara:

```
if (edad < 18) {
  let textoLet = 'Eres mayor de edad';
  var textoVar = 'Eres mayor de edad';
} else {
  let textoLet = 'Eres menor de edad';
  var textoVar = 'Eres menor de edad';
}
console.log(textoLet);  // mostrará undefined porque fuera del if no existe la variable
console.log(textoVar);  // mostrará la cadena
```

Variables

- Cualquier variable que no se declara dentro de una función (o si se usa sin declarar) es *global*. Debemos siempre intentar NO usar variables globales.
- Se recomienda que Los nombres de las variables sigan la sintaxis *camelCase* (ej.: *miPrimeraVariable*).
- Desde ES2015 también podemos declarar constantes con **const**. Se les debe dar un valor al declararlas y si intentamos modificarlo posteriorment se produce un error.

- Variables.
- Tipos de datos.
- Asignaciones.
- Operadores.
- Comentarios al código.
- Sentencias.
- Decisiones.
- Bucles.

Tipos de datos

- Para saber de qué tipo es el valor de una variable tenemos el operador typeof. Ej.:

  typeof 3 devuelve number

  typeof 'Hola' devuelve string

- En Javascript hay 2 valores especiales:
  - undefined: es lo que vale una variable a la que no se ha asignado ningún valor
  - null: es un tipo de valor especial que podemos asignar a una variable. Es como un objeto vacío (typeof null devuelve object)
- También hay otros valores especiales relacionados con operaciones con números:
  - NaN (Not a Number): indica que el resultado de la operación no puede ser convertido a un número (ej. 'Hola'*2, aunque '2'*2 daría 4 ya que se convierte la cadena ‘2’ al número 2)
  - Infinity y -Infinity: indica que el resultado es demasiado grande o demasiado pequeño (ej. 1/0 o -1/0)

Tipos de datos

- ***Casting* de variables**
- Como hemos dicho las variables pueden contener cualquier tipo de valor y, en las operaciones, Javascript realiza **automáticamente** las conversiones necesarias para, si es posible, realizar la operación. Por ejemplo:

  - `'4' / 2` devuelve 2 (convierte '4' en 4 y realiza la operación)
  - `'23' - null` devuelve 0 (hace 23 - 0)
  - `'23' - undefined` devuelve *NaN* (no puede convertir undefined a nada así que no puede hacer la operación)
  - `'23' * true` devuelve 23 (23 * 1)
  - `'23' * 'Hello'` devuelve *NaN* (no puede convertir 'Hello')
  - `23 + 'Hello'` devuelve '23Hello' (+ es el operador de concatenación así que convierte 23 a '23' y los concatena)
  - `23 + '23'` devuelve 2323 (OJO, convierte 23 a '23', no al revés)

Tipos de datos. String

- String
- Las String o cadenas de caracteres
- Cadenas literales (o primitivas)
  - Los cadenas literales se construyen asignando un valor entrecomillado a una variable. El tipo de comillas pueden ser tres: comillas simples ('), comillas dobles (") y comillas de ejecución o acentos graves (``).

```
var cad1 = "";        // cadena vacía
var cad2 = 'Uno';     // Cadena con 3 caracteres
var cad3 = `Hola`;    // Cadena con 4 caracteres
```

Tipos de datos. String

- Constructor de cadenas

También podemos usar el constructor String para crear cadenas, aunque esto rara vez es necesario. Para forzar la conversión a cadena se usa la función String(valor) (ej. String(23) devuelve ‘23’)

```
var cad5 = new String()     // Cadena vacía
var cad6 = new String("2")  // Cadena {"2"}
var cad7 = new String(123); // Cadena {"123"}
```

Tipos de datos. String

- Una diferencia importante a la hora de elegir qué método utilizamos para crear una cadena es si queremos compáralas con otra cadena. Si utilizamos new para crear las cadenas, estaremos comparando direcciones, no valores; si utilizamos literales, estaremos comparando valores:

```
var cad10 = new String("2");
var cad11 = new String("2");

var cad12 = String("2");
var cad13 = "2";

cad10 == cad11    // false
cad10 === cad11   // false

cad10 == cad12    // true (cad10 se convierte a literal para la operación)
cad10 === cad12   // false

cad12 == cad13    // true
cad12 === cad13   // true
```

Tipos de datos. String

- Accediendo a las cadenas
- Para acceder a una posición de la cadena podemos utilizar la notación corchetes como con los arrays.Pero solo para leer el valor de la posición no para modificarlo (RECUERDA: las cadenas son **inmutables**). La primera letra se encuentra en la posición 0; la última en la posición length - 1. También contamos con el método charAt(posición):

Accediendo (leer) a cadenas

```
var cad = "1 Cadena";
cad[0];    // "1"
cad[3];    // 'C'
cad[cad.length];      // undefined
cad[cad.length - 1];  // a
cad.charAt(0);    // "1"
cad.charAt(3)     // 'C'
cad.charAt(cad.length);      // undefined
cad.charAt(cad.length - 1);  // a
```

Tipos de datos. String

- Recorrer los elementos de una cadena
- Como las cadenas se comportan como arrays, se pueden utilizar los métodos vistos para recorrer arrays (salvo forEach)

```
var unaCad = "Una cadena";
// Mostrar los valores
for (let i = 0; i < unaCad.length; i++){
  console.log(unaCad[i]);
}
```

```
for (valor of unaCad){
  console.log(valor);
} // => U,n,a, ,c,a,d,e,n,a
```

for .. in

```
// Mostrar los valores
for (pos in unaCad){
  console.log(unaCad[pos]);
}
```

Tipos de datos. String

- Manipulación del contenido de las cadenas
  - Adición de cadenas El operados de adicción de cadenas es +:

```
var unaCad = "Una";
var otra = unaCad + " " + "cadena"; // "Una cadena"
```

También se puede utilizar el método concat:

```
var unaCad = "Una";
var otra = unaCad.concat(" ", "cadena"); // "Una cadena"
```

Tipos de datos. String

- Eliminación de espacios sobrantes

  JavaScript cuenta con el método trim para realizar este trabajo:

```
str.trim()
```

```
str = "   \tcadena\n  ";
str.trim(); // => 'cadena'
```

Tipos de datos. String

- **.length**: devuelve la longitud de una cadena. Ej.: `'Hola mundo'.length` devuelve 10
- **.charAt(posición)**: `'Hola mundo'.charAt(0)` devuelve 'H'
- **.indexOf(carácter)**: `'Hola mundo'.indexOf('o')` devuelve 1. Si no se encuentra devuelve -1
- **.lastIndexOf(carácter)**: `'Hola mundo'.lastIndexOf('o')` devuelve 9
- **.substring(desde, hasta)**: `'Hola mundo'.substring(2,4)` devuelve 'la'
- **.substr(desde, num caracteres)**: `'Hola mundo'.substr(2,4)` devuelve 'la m'
- **.replace(busco, reemplaza)**: `'Hola mundo'.replace('Hola', 'Adiós')` devuelve 'Adiós mundo'
- **.toLocaleLowerCase()**: `'Hola mundo'.toLocaleLowerCase()` devuelve 'hola mundo'
- **.toLocaleUpperCase()**: `'Hola mundo'.toLocaleUpperCase()` devuelve 'HOLA MUNDO'
- **.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

Tipos de datos. String

- .startsWith(cadena): `'Hola mundo'.startsWith('Hol')` devuelve *true*
- .endsWith(cadena): `'Hola mundo'.endsWith('Hol')` devuelve *false*
- .includes(cadena): `'Hola mundo'.includes('mun')` devuelve *true*
- .repeat(veces): `'Hola mundo'.repeat(3)` devuelve 'Hola mundoHola mundoHola mundo'
- .split(sepadaror): `'Hola mundo'.split(' ')` devuelve el array ['Hola', 'mundo']. `'Hola mundo'.split('')` devuelve el array ['H', 'o', 'l', 'a', ' ', 'm', 'u', 'n', 'd', 'o']

Tipos de datos. String

- **Template literals**
- Desde ES2015 también podemos poner una cadena entre ` (acento grave) y en ese caso podemos 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:

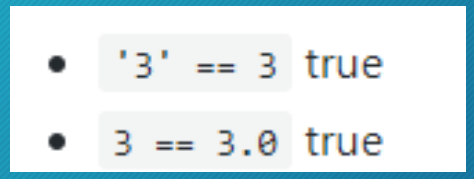
```
let edad=25;

console.log(\`El usuario tiene:
${edad} años\`)
```

Mostrará en la consola:

> El usuario tiene:
>
> 25 años

Tipos de datos. Boolean

- **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 (**!!**)
- Los operadores lógicos son ! (negación), && (and), || (or).
- Para comparar valores tenemos == y ===. La triple igualdad devuelve *true* si son igual valor y del mismo tipo. Como Javascript hace conversiones de tipos automáticas conviene usar la === para evitar cosas como:



Tipos de datos. Number

- **Number**
- Sólo hay un tipo de números, no existen enteros y decimales. El tipo de dato para cualquier número es number. El carácter para la coma decimal es el . (como en inglés, así que 23,12 debemos escribirlo como 23.12).

- Tenemos los operadores aritméticos +, -, *, / y % y los unarios ++ y – y existen los valores especiales Infinity y -Infinity (23 / 0 no produce un error sino que devuelve Infinity).

- Podemos usar los operadores aritméticos junto al operador de asignación = (+=, -=, *=, /= y %=).

Tipos de datos. Number

Algunos métodos útiles de los números son:

- **.toFixed(num)**: redondea el número a los decimales indicados. Ej. `23.2376.toFixed(2)` devuelve 23.24
- **.toLocaleString()**: devuelve el número convertido al formato local. Ej. `23.76.toLocaleString()` devuelve '23,76' (convierte el punto decimal en coma)

Algunos métodos útiles de los números son:

Otras funciones útiles son:

- **isNaN(valor)**: nos dice si el valor pasado es un número (false) o no (true)
- **isFinite(valor)**: devuelve *true* si el valor es finito (no es *Infinity* ni *-Infinity*).
- **parseInt(valor)**: convierte el valor pasado a un número entero. Siempre que compience por un número la conversión se podrá hacer. Ej.:

```
parseInt(3.65)      // Devuelve 3
parseInt('3.65')    // Devuelve 3
parseInt('3 manzanas')    // Devuelve 3, Number devolvería NaN
```

- **parseFloat(valor)**: como la anterior pero conserva los decimales

Operadores lógicos y relacionales

OPERADORES LÓGICOS Y RELACIONALES	DESCRIPCIÓN	EJEMPLO
==	Es igual	a == b
===	Es estrictamente igual	a === b
!=	Es distinto	a != b
!==	Es estrictamente distinto	a !== b
<, <=, >, >=	Menor, menor o igual, mayor, mayor o igual	a <=b
&&	Operador and (y)	a && b
||	Operador or (o)	a || b
!	Operador not (no)	!a

Operadores lógicos y relacionales básicos en JavaScript

Estructuras y bucles

- **Estructura condicional: if**
- El **if** es como en la mayoría de lenguajes. Puede tener asociado un **else** y pueden anidarse varios con **else if**.

```
if (condicion) {
    ...
} else if (condicion2) {
    ...
} else if (condicion3) {
    ...
} else {
    ...
}
```

Ejemplo:

```
if (edad < 18) {
    console.log('Es menor de edad');
} else if (edad > 65) {
    console.log('Está jubilado');
} else {
    console.log('Edad correcta');
}
```

Estructuras y bucles

- Se puede usar el operador **? :** que es como un *if* que devuelve un valor:

```
let esMayorDeEdad = (edad>18)?true:false;
```

Estructuras y bucles

- **Estructura condicional: switch**
- El **switch** también es como en la mayoría de lenguajes. Hay que poner *break* al final de cada bloque para que no continúe evaluando:

```
switch(color) {
    case 'blanco':
    case 'amarillo':    // Ambos colores entran aquí
        colorFondo='azul';
        break;
    case 'azul':
        colorFondo='amarillo';
        break;
    default:            // Para cualquier otro valor
        colorFondo='negro';
}
```

Estructuras y bucles

- Javascript permite que el *switch* en vez de evaluar valores pueda evaluar expresiones. En este caso se pone como condición *true*:

```
switch(true) {
    case age < 18:
        console.log('Eres muy joven para entrar');
        break;
    case age < 65:
        console.log('Puedes entrar');
        break;
    default:
        console.log('Eres muy mayor para entrar');
}
```

Estructuras y bucles

- **Bucle *while***
- Podemos usar el bucle *while…do*. Se ejecutará 0 o más veces.

```
while (condicion) {
    // sentencias
}
```

```
let nota=prompt('Introduce una nota (o cancela para finalizar)');
while (nota) {
    console.log('La nota introducida es: '+nota);
    nota=prompt('Introduce una nota (o cancela para finalizar)');
}
```

- O el bucle *do…while*: al menos se ejecutará 1 vez.

```
do {
    // sentencias
} while (condicion)
```

```
let nota;
do {
    nota=prompt('Introduce una nota (o cancela para finalizar)');
    console.log('La nota introducida es: '+nota);
} while (nota)
```

Ejercicio

- EJERCICIO: Haz un programa para que el usuario juegue a adivinar un número. Obtén un número al azar (busca por internet cómo se hace o simplemente guarda el número que quieras en una variable) y ve pidiendo al usuario que introduzca un número. Si es el que busca le dices que lo ha encontrado y si no le mostrarás si el número que busca el mayor o menor que el introducido. El juego acaba cuando el usuario encuentra el número o cuando pulsa en ‘Cancelar’ (en ese caso le mostraremos un mensaje de que ha cancelado el juego).

Estructuras y bucles

- **Bucle: for**
- **Bucle: for con contador**
- Creamos una variable contador que controla las veces que se ejecuta el for:

```
let datos=[5, 23, 12, 85]
let sumaDatos=0;

for (let i=0; i<datos.length; i++) {
    sumaDatos += datos[i];
}
// El valor de sumaDatos será 125
```

Estructuras y bucles

- **Bucle: for…in**
- El bucle se ejecuta una vez para cada elemento del array (o propiedad del objeto) y se crea una variable contador que toma como valores la posición del elemento en el array:

```
let datos=[5, 23, 12, 85]
let sumaDatos=0;

for (let indice in datos) {
    sumaDatos += datos[indice];     // los valores que toma indice son 0, 1, 2, 3
}
// El valor de sumaDatos será 125
```

Estructuras y bucles

- **Bucle: for…of**
- Es similar al *for…in* pero la variable contador en vez de tomar como valor cada índice toma cada elemento. Es nuevo en ES2015:

```
let datos = [5, 23, 12, 85]
let sumaDatos = 0;

for (let valor of datos) {
    sumaDatos += valor;       // los valores que toma valor son 5, 23, 12, 85
}
// El valor de sumaDatos será 125
```

Comentarios

- Los comentarios en el código ayudan a entenderlo. Como regla general, es mejor poner comentarios que no ponerlos. Si se ponen, deberían ir en el sentido de explicar lo que hace el código y **no** cómo lo hace ya que el propio código debe explicarse por sí mismo.
- Podemos poner comentarios tanto a nivel de línea como a nivel de varias líneas o de bloque:
  - Una línea (Se inicia con //, sin cierre)

```
// Comentario en una línea
variable = 5; // Comentario en línea
```

Comentarios

- Varias líneas (Se inicia con /* y se cierra con */)

```
/* Una línea
   Otra línea
   ...
*/
```