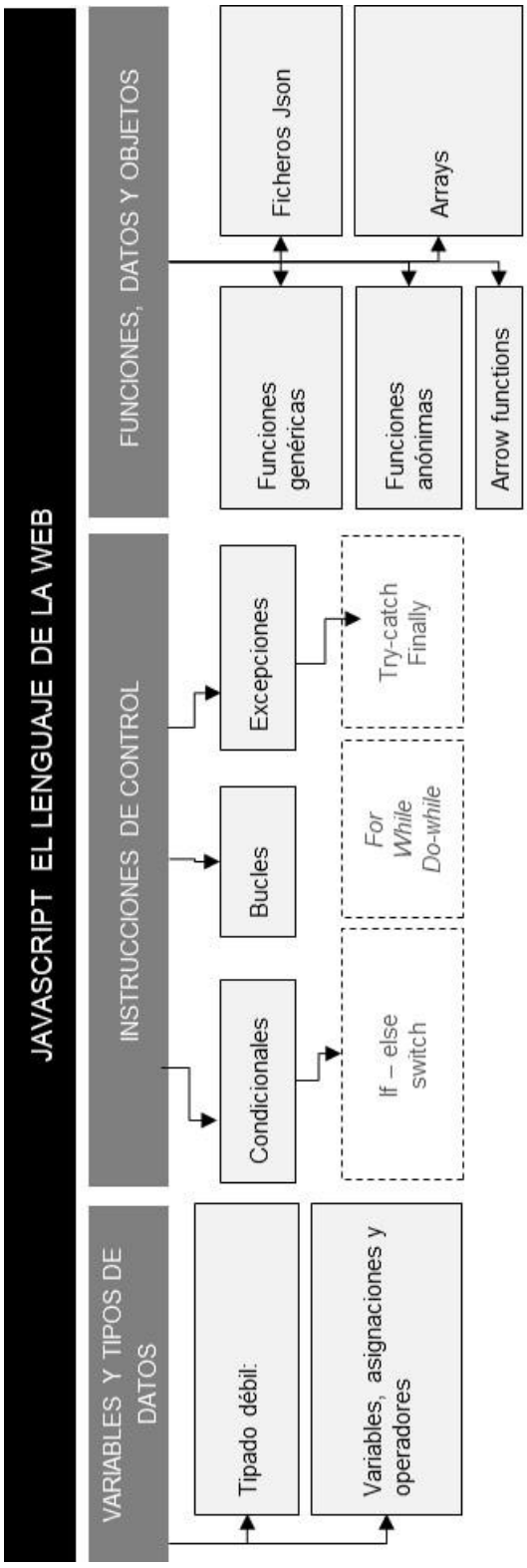
Desarrollo Web en Entorno Cliente

JavaScript: el lenguaje de la web

Índice

Esquema


JAVASCRIPT EL LENGUAJE DE LA WEB
VARIABLES Y TIPOS DE DATOS
Tipado débil:
Variables, asignaciones y operadores
INSTRUCCIONES DE CONTROL
Condicionales
If – else
switch
Bucles
For
While
Do-while
Excepciones
Try-catch
Finally
FUNCIONES, DATOS Y OBJETOS
Funciones genéricas
Funciones anónimas
Arrow functions
Ficheros Json
Arrays

Material de estudio

2.1. Introducción y objetivos

En el transcurso de este tema se realizará una introducción a JavaScript, el lenguaje de programación más utilizado en el desarrollo web de entorno cliente, analizando sus características y componentes principales. Fue creado por Netscape en 1995 como una extensión de HTML para Netscape Navigator 2.0. y su primer nombre fue Mocha.

Los estándares en tecnologías para la web se pueden distribuir en tres capas, que se complementan unas a otras:

1. HTML es un lenguaje de marcado que usa la estructura para dar un sentido al contenido web.
2. CSS es un lenguaje de reglas en cascada que usamos para aplicar un estilo a nuestro contenido en HTML.
3. JavaScript es un lenguaje de programación que permite crear contenido nuevo y dinámico, controlar archivos multimedia, crear imágenes animadas, etc.

Durante el desarrollo de este tema se deben conseguir los siguientes objetivos:

- Conocer los fundamentos principales de JavaScript.
- Aprender a codificar correctamente, sabiendo qué uso podemos dar a las variables y estructuras de control.
- Aprender qué son y cómo se manejan los operadores.
- Conocer herramientas útiles en JS.
- Manejar correctamente las funciones y ámbitos de las variables.

2.2. Variables, tipos de datos, comentarios y asignaciones

Cada una de las instrucciones que forman un programa se llama sentencia. Un programa será entonces una lista de sentencias.

Cada sentencia en JavaScript debe ser terminada con punto y coma (realmente no es imprescindible, pero es muy recomendable) y por mejorar la legibilidad del código (salvo excepciones) debe escribirse una sentencia por línea.

```
  const a; // Sentencia de declaración de variable a
  const b; // Sentencia de declaraciónd de variable b
  a = 3; // Sentencia de asignación del valor 3 en la variable a
  b = 2; // Sentencia de asignación del valor 2 en la variable b
  alert(a + b); // Sentencia de función alert que mostrará una alerta con el contenido '5'
```

Nuestra sentencia básica para mostrar información por pantalla será:

```
console.log(2);
```

Comentarios

Como cualquier lenguaje de programación, JavaScript incluye la posibilidad de añadir comentarios a su código.

Los comentarios son caracteres que no se ejecutarán y son útiles para documentar el código, explicar funcionalidades a futuros desarrolladores u ocultar al intérprete de código partes del programa mientras se está desarrollando. El código debe ser los suficientemente claro para que se pueda explicar por sí mismo, si esto no es posible se debe recurrir a los comentarios.

JavaScript tiene dos maneras de escribir comentarios: comentarios de **una línea** que empiezan con // o comentarios de **bloque** que empiezan con /* y acaban con */.

```
let b;  // Declaración de variable b
let a = 3;  // Asignación del valor 3 en la variable a

/*  Este código muestra la suma de dos números que se han declarado antes en una alerta del navegador.
*/
a = 3;
b = 2;
alert(a + b);
```

Tipos de datos

El tipo de dato es un concepto muy común en programación, como su propio nombre indica, son las diferentes clases de datos que JavaScript puede manejar. Cada tipo de dato tiene sus características y operaciones permitidas, por ejemplo, se pueden multiplicar dos datos que sean números, sin embargo, no se pueden multiplicar cadenas de texto.

JavaScript maneja los siguientes tipos de datos:

- Cadenas de texto.
- Números.
- Booleano.
- Array.
- Objetos.
- Indefinido (undefined).

Gracias a la palabra reservada **typeof** se puede preguntar a JavaScript por el tipo de dato de una variable.

```
typeof "Hola Mundo!"; // Devuelve 'string'
typeof 3;             // Devuelve 'number`
```

El tipo de datos especial undefined significa que esa variable todavía no alberga nada o lo que albergaba se ha borrado pero la variable sigue definida:

```
let a;
typeof a; // undefined
a = 3;
typeof a; // number
a = undefined;
typeof a // undefined
```

Variables

Una variable permite almacenar valores para poder ser leídos en otro momento. Una variable en JavaScript se declara con la palabra reservada: let y el identificador de la variable.

```
let a; // Declaración de variable
a = 3; // Asignación de valor en variable
console.log(a); // la consola mostrará: 3
a = 2; // Reasignación de valor en variable
console.log(a); // la consola mostrará: 2
```

El identificador debe ser único, es decir, no puede haber dos variables con el mismo identificador/nombre. Conviene utilizar identificadores cortos, pero a la vez descriptivos para hacer la labor de desarrollo más sencilla. Al elegir un identificador, además se debe saber que:

- Se puede utilizar cualquier combinación de letras, números y el guion bajo.
- El primer carácter de un identificador no puede ser un número.
- JavaScript es case-sensitive, es decir, distingue entre minúsculas y mayúsculas por lo que la variable nombre y Nombre son dos variables distintas al tener distinto identificador.
- No se pueden utilizar palabras reservadas del lenguaje como identificador de variable, es decir, aquellas palabras que el lenguaje utiliza para sí mismo como this, let, const, if, etc.

Se puede declarar una variable y asignarle un valor en la misma sentencia:

```
let a = 3;
```

Si el valor de la variable no va a cambiar a lo largo de la ejecución, es conveniente utilizar la palabra reservada const para declararla.

```
const pi = 3.14;
pi = 33; // Esta sentencia dará un error ya que no puede reasignarse una variable declarada utilizando const
```

Comprobar el tipo de variable puede ser útil en determinados casos ya que **JavaScript es un lenguaje de programación de tipado débil**, esto quiere decir que para declarar una variable no hace falta indicar el tipo de dato que va a albergar y en la vida de esa variable el tipo de dato puede cambiar:

```
// Simplemente se declara la variable a, no hace falta indicar qué tipo de dato va a albergar, en otros lenguajes de programación si hace falta
let a;
typeof a; // undefined
a = 3;
typeof a; // number
a = "3";
typeof a; // string
```

Este tipado débil puede acarrear problemas y comportamientos inesperados ya que JavaScript al poder cambiar el tipo de una variable sin consultar con el programador, puede dar lugar a resultados que no son intuitivos.

2.3. Operadores

Los operadores permiten hacer diferentes operaciones con variables.

Operadores aritméticos

Estos operadores permiten realizar operaciones aritméticas básicas, algunos de ellos son:

- Suma: +
- Resta: -
- Multiplicación: *
- División: /
- Resto: %: Devuelve el resto de una división entera.

Operadores de comparación

Estos operadores permiten comparar dos expresiones. JavaScript hará lo posible por compararlas y devolver un valor booleano: true o false.

- == igual.
- === igual estricto (compara tipo de la variable).
- != distinto.
- !== distinto estricto (compara tipo de la variable).
- > mayor que.
- < menor que.
- >= mayor o igual.
- <= menor o igual.

Operadores lógicos

Estos operadores permiten realizar operaciones lógicas, JavaScript hará lo posible por comparar los valores y realizar la operación lógica entre ellos.

- Operador lógico and: &&.
- Operador lógico or: ||.
- Operador lógico not: !.

Operadores de asignación

Con un operador de asignación se asigna el valor de la derecha en el operador de la izquierda. Por ejemplo: x = 3 se asigna el valor 3 a x.

Algunos operadores de asignación:

- =: Asignación simple: x = 3.
- +=: Asignación de suma: x += 3 es lo mismo que: x = x + 3.
- -=: Asignación de resta: x -=3 es lo mismo que: x = x - 3.

Operadores de cadena

El operador + si es utilizado con una cadena de texto no realizará una suma (ya que no es posible sumar cadenas de texto), si no que realizará una concatenación (unión de los dos textos).

2.4. Instrucciones de control

El flujo del programa es la línea que sigue el dispositivo ejecutando código. Si no ocurre nada que lo altere, el flujo empezará en la primera sentencia e irá ejecutando una a una cada sentencia de arriba a abajo hasta llegar al final.

Sin embargo, es complicado entender un programa sin que el flujo pueda variar. Por ejemplo, en una web de la previsión meteorológica habrá condiciones en el flujo: si hace buen tiempo ocurre una cosa y si hace mal tiempo ocurre otra cosa. De esta manera el flujo tiene caminos diferentes que recorre hasta llegar al final de la ejecución.

Instrucciones de control condicional: if /if-else

La estructura de control condicional if es la estructura más sencilla, con ella mediante una condición booleana el flujo puede tomar un camino específico para esa condición.

```
// Suponiendo la variable temperatura como temperatura actual
let mensaje = temperatura + "º";
```

```
  // Si la temperatura es menor que 0, se añaden unos caracteres haciendo
de copo de nieve
  if (temperatura < 0) {
    mensaje = "***" + mensaje + "***";
  }
  console.log(mensaje);
```

La condición booleana aparece entre paréntesis (). Esta condición será evaluada por JavaScript y si el resultado es true, el flujo entrará a ejecutar las sentencias que hay dentro de las llaves {}, en caso contrario, saltará dichas sentencias.

Se puede complicar un poco más añadiendo otro camino para el flujo mediante la estructura if-else: Si se cumple la condición el flujo seguirá un camino y si no, se irá a otro camino:

```
  // Suponiendo la variable edad declarada con un número que representa la
edad de una persona
  if (edad >= 18) {
    console.log("mayor de edad");
  } else {
    console.log("menor de edad");
  }
```

Operador ternario

El operador ternario recrea la estructura de if-else pero de una manera muy simple y para casos en los que solo hay una sentencia para cada condición. Es simplemente una sintaxis diferente para esta estructura de control en el que se prescinde del if, else y de las llaves y se separa la condición de la primera sentencia con «?» y la primera sentencia de la segunda con «:».

```
  // Todos los bloques siguientes imprimen por pantalla lo mismo
  // Utilizando estructura if-else
  if (edad >= 18) {
    console.log("mayor de edad");
  } else {
```

```
    console.log("menor de edad");
  }

  // Utilizando operador ternario
  (edad >=18) ? console.log("mayor de edad") : console.log("menor de edad");

  // Utilizando el operador ternario en una asignación
  const mensaje = (edad >=18) ? "mayor de edad" : "menor de edad";
  console.log(mensaje);
  // Utilizando el operador ternario directamente como parámetro de función
  console.log((edad >=18) ? "mayor de edad" : "menor de edad");
```

Instrucción de control condicional: switch

La estructura switch resuelve un flujo condicional cuando este tiene muchas opciones. Podría resolverse con un if-else muy grande, pero la estructura switch es más fácil de leer y mantener.

```
  // Convierte una variable donde está el día de la semana en número en la palabra en castellano para ese día
  let dia;
  switch (diaNumero) {
    case 0:
      dia = "Lunes";
      break;
    case 1:
      dia = "Martes";
      break;
    case 2:
      dia = "Miércoles";
      break;
    case 3:
      dia = "Jueves";
      break;
    default:
      dia = "Error";
  }
  console.log(dia);
```

La sentencia break es parte de la estructura switch, es necesario añadirla ya que sin ella se evaluaría las sentencias del siguiente **case**, aunque la condición no fuese verdadera.

Por otro lado, default se puede utilizar como caso especial, el flujo evaluará las sentencias que estén dentro de él cuando ningún caso anterior haya sido seleccionado.

Instrucciones de control iterativas: for /for in / foro f

Estas estructuras es la forma más sencilla de manejar un bucle. Un bucle es una estructura de control que ejecuta una y otra vez un mismo conjunto de sentencias.

for

La estructura for funciona utilizando un índice (un número entero que va cambiando de valor) como control, de tal manera que se establece:

- Un punto de inicio.
- La manera en la que ese índice va a cambiar.
- Un punto de final.

A cada vuelta se puede consultar el índice. Utilizando la sintaxis del for se indica esas condiciones para el bucle, dentro de los paréntesis () y separado por «;» se indica:

- El índice que se va a utilizar.
- La condición para la que el bucle seguirá repitiéndose.
- Qué transformación se hace del índice a cada iteración (vuelta del bucle).

```
//For para rellenar lista
for (i = 0; i < 10; i++) {
  console.log(i);
}
```

for in

Funciona de una manera similar al bucle for, sin embargo, en lugar de utilizar un índice, recorre un array o un objeto y deja disponible un índice correspondiente a cada uno de los elementos del array u objeto:

```
const arr = ["a", "b", "5"];
for (indice in arr) {
  console.log(indice); // Cada uno de los índices / posiciones del array
  console.log(arr[indice]); // Cada uno de los elementos que ocupan posiciones en el array
}
// Se mostrará por consola
// 0 a 1 b 2 c
```

for of

Esta estructura itera un objeto iterable como un array y deja en cada vuelta del bucle disponible para una de las partes que componen ese objeto iterable:

```
let array = ["a", "b", "c"];
for (let value of array) {
  value = value + "33";
  console.log(value);
}
// a33 b33 c33

// Si el valor no se va a modificar dentro del bucle, se puede declarar la variable con const
let array = ["a", "b", "c"];

for (const value of array) {
  console.log(value);
}
// a b c
```

Instrucciones de control iterativas: while / do-while

La estructura while como su nombre indica realiza un bucle mientras una condición se cumpla, en el momento que esa condición se evalúe como falsa, el bucle acabará.

```
let i = 0;
while (i < 4) {
  console.log(i);
  i++;
}
// Se mostrará por consola
// 0 1 2 3 4

let i = 10;
while (i < 4) {
  console.log(i);
  i++;
}
// No se mostrará nada por consola porque en la primera vuelta del bucle se evalúa ( 10 < 4) a
 por lo que no entra en el bucle
```

Similar al while, la estructura do-while que es similar con una salvedad: la primera vuelta se ejecutará siempre:

```
let i = 1;
const n = 5;

do {
  console.log(i);
  i++;
} while(i <= n);

// Se mostrará por consola
// 1 2 3 4 5
```

Interrumpir o abandonar un bucle

La sentencia break sirve para terminar bucles, hacer que el flujo no respete el bucle y salte como si el bucle hubiese acabado. La utilización de break debe hacerse en muy determinados casos, siempre es más recomendable diseñar correctamente un bucle antes de tener que recurrir a esta sentencia, a excepción de la sentencia switch donde la utilización de break es requerida.

Por ejemplo, el siguiente bucle está diseñado para ir del 1 al 5, sin embargo, en su ejecución cuando el valor del índice es 3, la sentencia break hace que el flujo salte, evitando el resto del bucle.

```
// Bucle con un índice del 1 al 5
for (let i = 1; i <= 5; i++) {
  if (i === 3) {
    break;
  }
  console.log(i);
}

// Se mostrará por consola
// 1 2
```

Gestión de excepciones: try-catch

Una excepción es un error que se produce (por la razón que sea) cuando un programa se está ejecutando, y salvo que se gestione esa excepción, la ejecución del programa no puede seguir.

Para manejar excepciones y que, aunque ocurra una el programa pueda seguir ejecutándose, se utiliza la estructura try/catch.

```
try {
  console.log("pre error");
  console.log(a);
  console.log("post error");
}

catch(error) {
  console.log('Error: ' + error);
}

// Mostará por consola
// pre error
// Error: ReferenceError: a is not defined
```

En el anterior ejemplo el flujo de ejecución entra en el try, cualquier excepción que ocurra dentro de ese bloque try será manejado por el bloque catch, por donde continuará la ejecución (saltando el resto de las sentencias que pudiese haber en el bloque try).

Como complemento al try-catch existe finally, gracias a finally se pueden agrupar sentencias de manera que se **ejecutarán tanto ocurra una excepción como si no**.

```
// Se produce excepción
try {
  console.log("pre error");
  console.log(a);
  console.log("post error");
}
catch(error) {
  console.log('Error: ' + error);
}
finally {
  console.log("finally");
}

// pre error
// Error: ReferenceError: a is not defined
// finally
// No se produce excepción
try {
  console.log("pre error");
  console.log("post error");
}
catch(error) {
  console.log('Error: ' + error);
}
finally {
  console.log("finally");
}
// pre error
// post error
// finally
```

2.5. Funciones

Una función es un bloque de código que realiza una determinada tarea. A base de esos pequeños bloques de código es como se construyen los programas (dividen un problema grande en otros más pequeños).

```
function nombreDeLaFuncion () {
  // Contenido de la función
}
```

Las funciones se declaran en JavaScript utilizando la palabra reservada function seguida del nombre que se desea dar a la función y entre llaves {} ese trozo de código que debe ejecutar la función.

Una vez declarada la función, se puede llamar a su ejecución utilizando el nombre de la función seguido por ().

Parámetros de una función

Con el fin de hacer esos trozos de código más útiles y reutilizables, las funciones pueden recibir parámetros.

Los parámetros son valores que se reciben en la función y actúan como variables predeclaradas solo en el ámbito (entre llave y llave) de la función.

```
function saludar(nombre) {
  console.log("Hola " + nombre);
}

saludar("Félix");
// Mostrará por consola: Hola Félix

saludar("Fran");
// Mostrará por consola: Hola Fran
```

En el ejemplo anterior, se ha definido la función saludar que acepta el parámetro nombre de manera que puede haber muchos tipos de saludos, tantos como parámetros nombre reciba la función. De esta manera podemos saludar a diferentes nombres pasando como argumento de la función una cadena de texto en el momento de llamar a la función: saludar ("Félix").

Las funciones pueden tener varios parámetros, cada uno de esos parámetros se separa con «,».

```
function sumar(a, b) {
  console.log(a + b);
}

sumar(2,3);
// Mostrará por consola 5
```

Los parámetros de una función pueden tener un valor por defecto, de esta manera ese parámetro se convierte en opcional, ya que no es necesario pasarlo al llamar la función (hay un valor por defecto en tal caso).

```
function saludar(nombre = "Desconocido") {
  console.log("Hola " + nombre);
}

saludar(); // Hola Desconocido
saludar("Víctor") // Hola Víctor
```

Retorno de una función

Ese conjunto de sentencias que es una función puede además devolver un valor, de esta manera es muy útil para abstraer cálculos y simplemente llamando a una función, se obtiene un resultado.

```
function sumar(a, b) {
  return a + b;
}
```

```
const x = sumar(2,3);
console.log("La suma es: " + x);
// Mostrará por consola
// La suma es: 5
```

En el ejemplo anterior la función sumar devuelve la suma de los dos números que entran como parámetro. Ese resultado se guarda en la constante **x** que posteriormente se utiliza.

Conviene saber que:

- Una función no ejecuta más código tras encontrar la palabra return.
- Si una función no tiene la palabra return, devolverá siempre undefined.

Ámbito de variables

El ámbito de una variable define las partes del código en la que una variable está disponible para utilizarse.

JavaScript tiene dos ámbitos de variable:

- Ámbito de variables global: Las variables declaradas fuera de una función se consideran variables de ámbito global. Al declararse fuera de una función estas variables están disponibles para utilizarse en cualquier parte del programa.
  - Ámbito de variables local: Si una variable se declara dentro de una función, esa variable estará únicamente disponible dentro de esa función. Cada función tiene su ámbito propio llamado ámbito local.

```
const nombre = "Víctor";

function saludar() {
  const prefijo = "Hola ";
  return prefijo + nombre;
}
```

```
console.log(saludar());
// Hola Víctor

console.log(prefijo);
// ReferenceError: prefijo is not defined
```

En el ejemplo anterior:

- La variable nombre es de ámbito global y por lo tanto está disponible para usar dentro de todo el código, incluido dentro de la función saludar.
- La variable prefijo es de ámbito local ya que se ha declarado dentro de la función saludar y por lo tanto solo estará disponible dentro de dicha función. Al intentar utilizarla fuera de la función JavaScript no sabe de ella y da error.

Con la declaración de variables con let y const se puede hablar incluso de ámbito de bloque. Toda variable declarada de esta manera estará disponible entre las llaves {} en la que se encuentre.

Funciones anónimas

Las funciones se pueden definir como expresión, es decir, se pueden guardar en una variable. Al guardar una función en una variable **ya no es necesario asignarle nombre** por lo que se convierte en una función anónima. Para llamar a una función anónima guardada en una variable simplemente se debe escribir el nombre de esa variable seguido de ().

```
// La constante saludar guarda una función anónima
const saludar = function() {
  console.log("hola!");
}

// Llamada a la función en la variable saludar
saludar(); // hola!
```

Arrow functions

Esta nueva sintaxis para funciones se introdujo en la revisión ES6 de JavaScript, la cual hace más sencillo escribir funciones.

```
// La función sumar funciona de la misma manera en los siguientes ejemplos
const sumar = function(a, b) {
  return a + b;
}

// Equivalencia en arrow function
const sumar = (a, b) => {
  return a + b
}

// Como únicamente tiene una sentencia return, se puede quitar junto con las llaves
const sumar = (a, b) => a + b
```

2.6. Manejo de datos

Gracias a determinado tipo de datos que incluye JavaScript como los arrays y objetos se hace muy sencillo para el programador manejar gran cantidad de datos.

Arrays

Array es una colección de elementos que se guardan juntos con el fin de hacer operaciones como recorridos, mutaciones u ordenación.

La manera más sencilla de crear un array es utilizar los caracteres [] y asignar dicho array a una variable.

```
const arrayVacio = [];
const colores = ["rojo","verde","azul"];

// Otra manera de crear un array
const arrayVacioOldSchool = new Array();
const numerosOldSchool = new Array(1, 2, 3);
```

Una vez declarado un array, se puede acceder a sus elementos utilizando un índice. Para ello, basta con escribir el nombre del array y el índice entre dos []. Los arrays en JavaScript empiezan en el índice 0 y tendrán tantos índices como elementos - 1:

```
// Hay 3 colores en el array ocupando los índices 0, 1 y 2
const colores = ["rojo", "azul", "verde"];
console.log(colores[0]); // rojo
console.log(colores[1]); // azul
console.log(colores[2]); // verde
console.log(colores[3]); // undefined
```

Una vez creado un array es posible añadir o quitar elementos con los métodos propios del array para tal caso:

- push() Añade un elemento al final del array.
- unshift() Añade un elemento al principio del array.
- pop() Quita el último elemento de un array.
- shift() Quita el primer elemento de un array.

```
const array = [];
array.push("rojo");
array.push("azul"); // push Añade un elemento al final de array
console.log(array); // rojo, azul
array.unshift("verde");
console.log(array) // verde, rojo, azul
array.pop();
console.log(array) // verde, rojo
array.shift();
console.log(array) // rojo
```

Además, es posible reemplazar un elemento que ya pertenezca al array sabiendo su índice, simplemente se debe realizar una asignación:

```
const colores = ["rojo", "azul", "verde"];
colores[1] = "violeta";
console.log(colores) // rojo, violeta, verde
```

Métodos y propiedades de los arrays

length

Gracias a la propiedad length se puede consultar el tamaño de un array. Como los arrays empiezan en el índice 0, se puede decir que el último índice disponible de un array será su tamaño - 1.

```
let array = [];
console.log(array.length); // 0
array = [1, 2, 3, 4, 33];
console.log(array.length); // 5
console.log(array[array.length - 1]); // 33
```

foreach

Gracias al método forEach incluido en los arrays de JavaScript, se puede iterar un array de una manera muy sencilla, al igual que se hace con un bucle for pero sin tener que mantener el índice.

forEach es un método que necesita como argumento una función. Dicha función recibe como parámetro cada uno de los elementos del array en cada una de las vueltas, es decir, esa función se ejecuta tantas veces como elementos tenga el array y en cada ejecución su primer parámetro es el elemento actual.

```
const colores = ["rojo", "verde", "azul"];
colores.forEach(function(value) {
  // En value está el elemento del array correspondiente
  // En la primera ejecución de esta función value = rojo
  // Segunda ejecución: value = verde
  // Tercera ejecución: value = azul
  console.log(value);
});

// Mostrará por consola
// rojo verde azul
```

El método forEach recorrerá desde el primer hasta el último elemento del array ejecutando la función para cada elemento, de manera que el programador no debe preocuparse por el índice. Ese índice se puede consultar ya que le llega a la función como el segundo parámetro (es opcional utilizarlo).

```
const colores = ["rojo", "verde", "azul"];
colores.forEach(function(value, i) {
  console.log(value);
  console.log(i);
});

// Mostrará por consola
// rojo 0 verde 1 azul 2
```

map

El método map mapea un array: para cada elemento lo transforma con una función y devuelve el resultado en otro array.

Este método se utiliza asignando el resultado, ya que **no transforma el array original**. Recibe como argumento una función que se utilizará para transformar cada elemento del array. Dicha función recibe como parámetro cada elemento y mediante una sentencia return devuelve el elemento transformado.

```
const numeros = [1, 2, 3, 4, 33];

const numerosPor2 = numeros.map(function(elemento) {
  return elemento * 2;
})

console.log(numerosPor2); // 2, 4, 6, 8, 66
```

find

El método find permite buscar elementos dentro de un array. Este método revisará cada uno de los elementos y parará de buscar una vez haya encontrado aquel

elemento que está buscando, esto quiere decir que o bien no encuentra nada o bien encuentra un elemento, pero no encontrará más de un elemento. Como otros métodos de este tipo, no modifica el array, si no que devuelve el resultado.

Este método necesita una función como argumento que utilizará por cada uno de los elementos del array. Esa función tiene como parámetro cada uno de los elementos del array y se puede comprobar por cada elemento si cumple la condición de búsqueda: si la función devuelve true, dará por encontrado el elemento, sin embargo, si la función devuelve seguirá buscando.

```
const numeros = [1, 2, 3, 4, 33];

// Búsqueda con fin y functión tradicional
const numeroMayorQueCuatro = numeros.find(function (elemento) {
  return elemento > 4;
});

console.log(numerMayorQueCuatro); // 33

// Utilizando arrow function
const numeroMayorQueCuatro = numeros.find(elemento => elemento > 4);
console.log(numerMayorQueCuatro); // 33

// Dejando la función en una constante
const mayorQueCuatro = elemento => elemento > 4;
const numeroMayorQueCuatro = numeros.find(mayorQueCuatro);
console.log(numerMayorQueCuatro); // 33
```

filter

El método filter es similar a find. Su diferencia es que puede encontrar todos los elementos que cumplan una condición: esa condición es la que decida la función que se le pasa como argumento.

```
const numeros = [1, 2, 3, 4, 16, 33, 1502];

const pares = numeros.filter(function(numero) {
```

```
  return numero % 2 === 0;
});

console.log(pares); // 2, 4, 16, 1502
```

sort

El método sort ordena un array siguiendo la condición que marque la función que se le pasa como argumento. La función de ordenación recibe dos parámetros que son utilizados para hacer precisamente la comparación como si fuesen dos elementos del array.

```
const numeros = [33, 45, 1, 100, 2, 4];

const ordenados = numeros.sort(function(a, b) {
  return a - b;
});

console.log(ordenados); // 1, 2, 4, 33, 45, 100
```

Una función de comparación function(a,b) devuelve:

- < 0. Si el valor devuelto por la función de comparación es menor que 0, entonces el elemento b es mayor que a.
- Si el valor devuelto por la función de comparación es 0, entonces a y b deben tener la misma posición.
- 0. Si el valor devuelto por la función de comparación es mayor que 1, entonces el elemento.

reduce

El método reduce como su nombre indica, reduce un array a una variable, es decir, de una forma similar al map, pasa por todos los elementos de un array y ejecuta una función.

La función recibe por parámetro el elemento actual del array y un acumulador, ese acumulador es lo que al final se reduce del array, es decir, para cada elemento del

array transforma el acumulador que ya previamente han transformado los elementos anteriores y que transformarán los elementos siguientes. Para que el acumulador pase al siguiente elemento, la función debe devolver la transformación que le ha hecho al acumulador.

```
const numeros = [33, 45, 1, 100, 2, 4];

const suma = numeros.reduce(function(acumulador, numero) {
  return acumulador + numero;
});

console.log(suma); // 185
```

2.7. JavaScript orientado a objetos

Los objetos son un tipo de dato para almacenar múltiples colecciones de datos mediante clave/valor.

```
const persona = {
  nombre: 'Mercer',
  apellido: 'Roussel',
  direccion: 'Cruce Casa de Postas, 33'
};
```

Cada una de las propiedades del objeto: *claves*, funciona como una variable declarada *dentro* del objeto, de manera que la clave es el nombre de la variable y el valor es el contenido de la variable que puede ser de cualquier tipo: un string, booleano, un array, incluso otro objeto.

Para acceder a las propiedades de un objeto se puede utilizar la misma sintaxis que para acceder a un array o mediante el carácter «:».

```
const persona = {
  nombre: 'Mercer',
```

```
  apellido: 'Roussel',
  direccion: 'Cruce Casa de Postas, 33'
};
console.log(persona['nombre']); // Mercer
console.log(persona.apellido); // Roussel

// Si la propiedad no existe, se devuelve undefined
console.log(persona['xyz']); // undefined
```

Métodos de objetos

En JavaScript una variable puede contener una función, por lo que una propiedad de un objeto puede contener una función:

```
const persona = {
  nombre: 'Mercer',
  apellido: 'Roussel',
  saludar: function() { console.log('Soy Mercer') }
}

persona.saludar(); // Soy Mercer
```

De esta manera se añaden nuevas funcionalidades al objeto para hacerlo un poco más eficaz al utilizarse. Esas funciones predefinidas pueden incluso acceder al propio objeto en sí, utilizando para ello la palabra reservada this. this es una palabra reservada en JavaScript que se refiere al contexto de ejecución en el que se realiza una sentencia, en el caso de funciones dentro de objetos, this hace referencia al mismo objeto ya que es el contexto en el que se ejecuta dicha sentencia.

```
const persona = {
  nombre: 'Mercer',
  apellido: 'Roussel',
  saludar: function() { console.log('Soy Mercer') }
}

persona.saludar(); // Soy Mercer
```

Recorrer un objeto

Aunque un objeto está diseñado para albergar información de todo tipo, en determinados casos puede ser interesante recorrerlo (por ejemplo, para buscar información).

Es interesante destacar que JavaScript no asegura el orden de las claves de un objeto, porque un objeto en realidad está pensado para albergar información y recuperarla con una clave y no para almacenar información de manera ordenada (para eso están los arrays), por lo tanto, el orden en el que aparecen las claves en una iteración puede no ser el mismo orden en el que se definieron en el objeto.

```
// Bucle for ... in
const numeros = { a: 1, b: 2, c: 3 };

for (const clave in numeros) {
  console.log(`${clave}: ${numeros[clave]}`);
}
// Muestra por consola:
// a: 1
// b: 2
// c: 3

// Object.keys
// Devuelve un array con las claves del objeto de manera que se puede iterar
const numeros = { a: 1, b: 2, c: 3 };
Object.keys(numeros).forEach(function(clave){
  console.log(`${clave}: ${numeros[clave]}`);
});

// Muestra por consola:
// a: 1
// b: 2
// c: 3
```

2.8. Referencias bibliográficas

Azaustre, C. (2021). *Aprendiendo JavaScript: Desde cero hasta ECMAScript 6+.* Independently published.

Gómez, M. R. (2021). *Curso de desarrollo Web. HTML, CSS y JavaScript.* ANAYA MULTIMEDIA.

Microsoft. (15 de 07 de 2024). *TypeScript official web site*. Obtenido de https://www.typescriptlang.org/

A fondo

Primeros pasos con JavaScript

MDN Contributors. (2024). Primeros pasos con JavaScript [documentación en línea]. https://developer.mozilla.org/es/docs/Learn/JavaScript/First_steps

Introducción a JavaScript de la mano de MDN Web Docs, conviene tener a mano este tipo de documentaciones ya que suelen estar actualizadas, pero no se debe tomar como documentación de estudio.

Guardar datos en el navegador

Kantor, I. (2022). Cookies, document.cookies [documentación en línea]. https://es.javascript.info/cookie

Kantor, I. (2022). LocalStorage, sessionStorage [documentación en línea]. https://es.javascript.info/localstorage

Mediante el uso de cookies o localStorage, el programador puede guardar información en el navegador del usuario, con el fin de tenerla disponible cuando el usuario entre otra vez en el sitio web, realizar cálculos en el lado del cliente u otros usos.

Propiedades y métodos estáticos

Kantor, I. (2022). Propiedades y métodos estáticos [documentación en línea]. https://es.javascript.info/static-properties-methods

Las clases en JavaScript soportan un tipo de propiedades y métodos útiles en determinadas situaciones llamados *estáticos* para los que no hace falta instanciar la clase para acceder a ellos.

Códigos de respuesta HTTP

Silva, C. (2023). Códigos de estado HTTP: una guía sin tecnicismos [artículo en línea]. https://es.semrush.com/blog/codigos-de-estado-http/

Los códigos de respuesta del protocolo HTTP proveen de mucha información cuando se realiza una llamada. Conocer su naturaleza y al menos sus rangos es algo que todo desarrollador web debe conocer.

Entrenamientos

Entrenamiento 1

- Escribe un programa en JavaScript que dada una calificación numérica entre 0 y 10 y la transforma en calificación alfabética, escribiendo el resultado.
    - de 0 a <3 Muy Deficiente.
    - de 3 a <5 Insuficiente.
    - de 5 a <6 Bien.
    - de 6 a <9 Notable
    - de 9 a 10 Sobresaliente.
- Desarrollo paso a paso:
    - Introducir un valor numérico entre 1 y 10 (se puede generar aleatoriamente)
    - Definir mediante una instrucción de control los posibles casos.
    - Generar la calificación en formato de texto.
    - Mostrar por consola la calificación.
- Solución: https://github.com/Anuar-UNIR/DWEC_2024_2025/tree/main/Tema%202/Entrenamiento%201

Entrenamiento 2

- Escribe un programa en JavaScript que dada una hora expresada en horas, minutos y segundos que nos calcula y escribe la hora, minutos y segundos que serán, transcurrido un segundo.
- Desarrollo paso a paso:
    - Generar una hora en formato HH:MM:SS (de 0 a 24h).
    - Sumar un segundo a la hora anterior.
    - Calcular los cambios y las diversas posibilidades (horas,minutos).
    - Mostrar la hora correcta en el formato HH:MM:SS

- Solución: https://github.com/Anuar-UNIR/DWEC_2024_2025/tree/main/Tema%202/Entrenamiento%202

Entrenamiento 3

- Escribe un programa en JavaScript para crear el juego de "Piedra, papel o tijera", el programa debe de seguir la siguiente estructura.
  - Explicarle al jugador cómo se juega.
  - Generar la jugada aleatoria de cada jugador (usar una función).
  - Decidir quién ha ganado.
- Desarrollo paso a paso:
  - Explicarle al jugador cómo se juega.
  - Generar la jugada aleatoria del ordenador.
  - Pedir al jugador su jugada mediante una letra (P para piedra, L para papel, T para tijeras):
  - Decidir quién ha ganado (comparación de las jugadas).
  - Mostrar la jugada y el ganador.
- Solución: https://github.com/Anuar-UNIR/DWEC_2024_2025/tree/main/Tema%202/Entrenamiento%203

Entrenamiento 4

- Crea un programa que cree un array con 100 números reales aleatorios entre 0.0 y 1.0, utilizando Math.random(). Define una función que dado un array y un numero muestre cuántos valores del array son igual o superiores al número dado.
- Desarrollo paso a paso:
  - Generar un array de dimensión 100
  - Rellenar con valores aleatorios usando Math.random()
  - Definir una función que tenga un número y un array como parámetros.
  - Implementar la función para que muestre cuantos valores son iguales al número pasado por parámetro.

- Mostar en la consola la información.

- Solución: https://github.com/Anuar-UNIR/DWEC_2024_2025/tree/main/Tema%202/Entrenamiento%204

Entrenamiento 5

- Desarrollar un programa que cree un array con 100 números reales aleatorios entre 0.0 y 10.0, utilizando Math.random(), nos calcule la media, mediana, la suma, el máximo, el mínimo y el valor más repetido.
- Desarrollo paso a paso
  - Generar un array de dimensión 100
  - Rellenar con valores aleatorios usando Math.random()
  - Calcular media, mediana, la suma, el máximo, el mínimo y el valor más repetido (se pueden usar funciones independientes o funciones que calculen más de una cosa a la vez)
  - Mostar en la consola la información.
- Solución: https://github.com/Anuar-UNIR/DWEC_2024_2025/tree/main/Tema%202/Entrenamiento%205