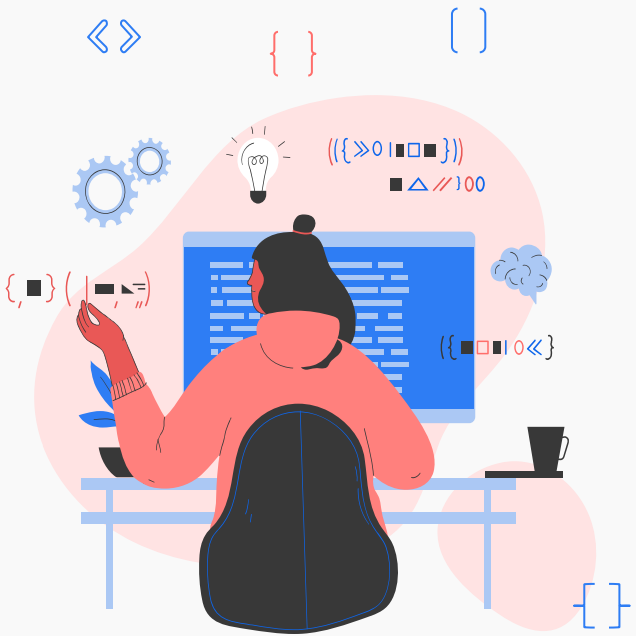
Desarrollo Web en Entorno Cliente

Tema 1 – Introducción a JavaScript

Marina Hurtado Rosales
marina.hurtado@escuelaartegranada.com



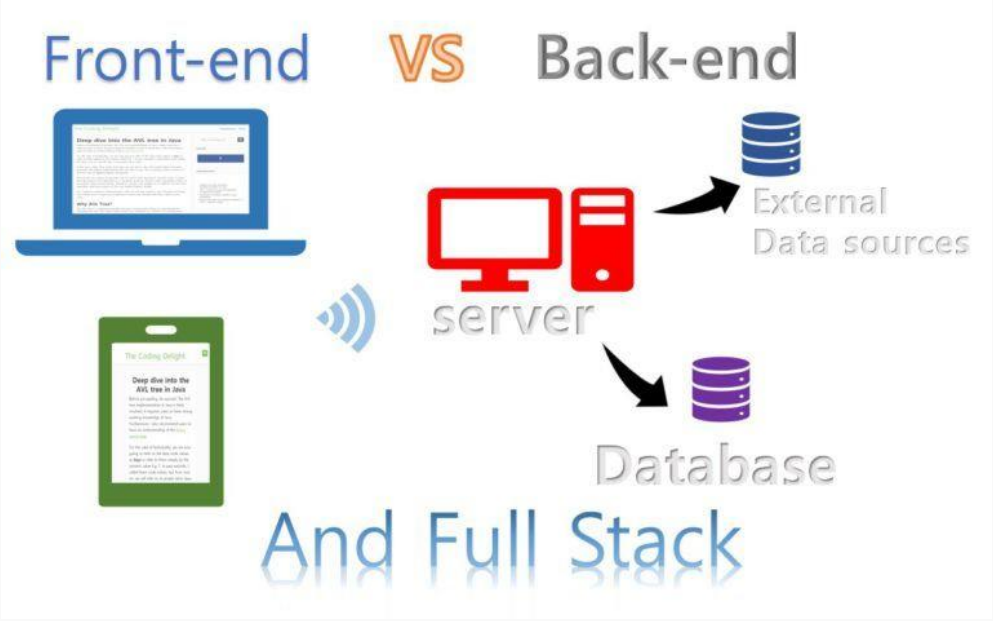
Introducción

Arquitectura C/S, navegadores web

Introducción

El desarrollo web es el trabajo necesario para crear un sitio web, tanto en Internet como en una red privada

Facilita la interacción con el usuario

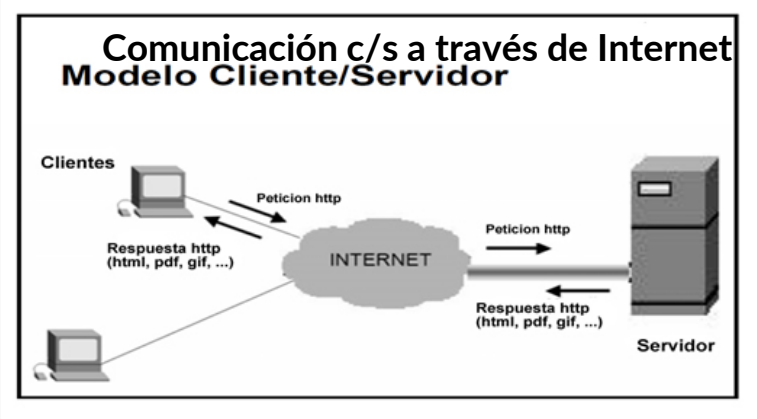


Programación más compleja, influye en el funcionamiento del sistema

Arquitectura cliente/servidor

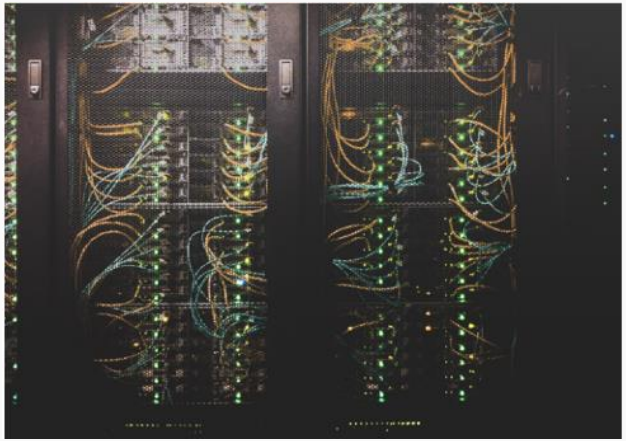
Es un modelo que crea aplicaciones que distribuyen las diferentes tareas, consiguiendo un balanceo en las tareas que realizan.

Comunicación c/s a través de Internet



Capa de presentación (lado cliente) = front-end

Comunicación c/s a través de la misma máquina

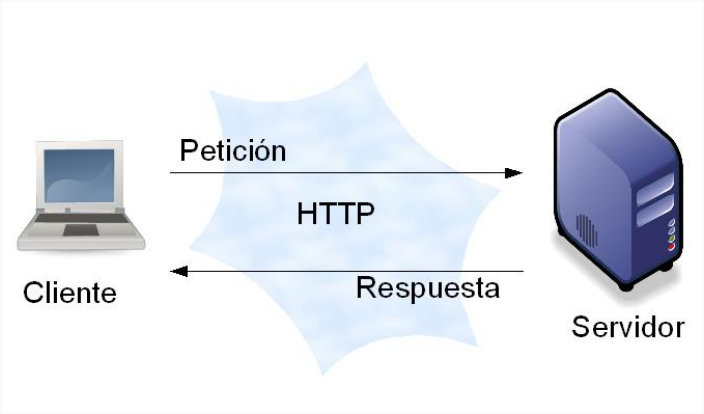


Capa de acceso a datos (lado servidor) = back-end

Tipos de arquitecturas

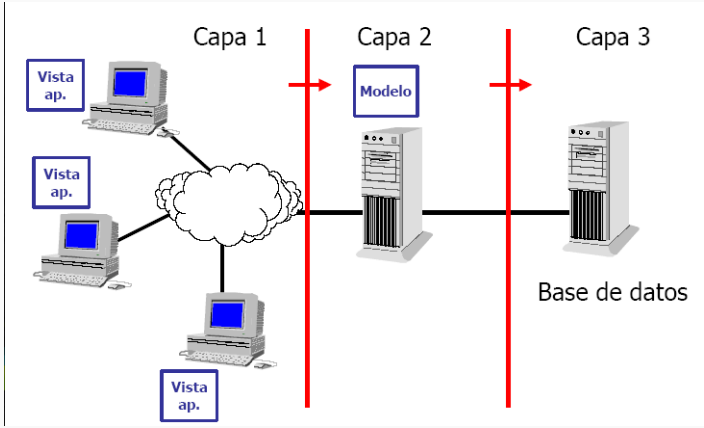
Se diferencian dependiendo de las tareas que realiza el servidor

Arquitectura en dos capas



El servidor al recibir una petición, responde con sus propios recursos.

Arquitectura en tres capas

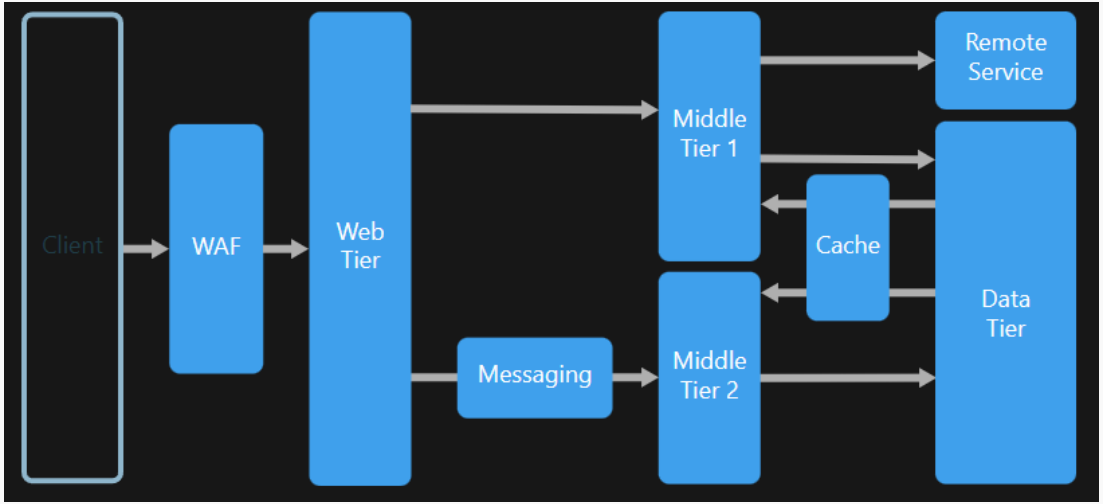


El servidor requiere del acceso de un componente externo (una BBDD en este caso)

Tipos de arquitecturas

Se diferencian dependiendo de las tareas que realiza el servidor

Arquitectura en N capas



Modelo con tantas capas se requiera para solucionar el problema

Ventajas y limitaciones C/S

Ventajas

- No es dependiente de una plataforma
- Alto nivel de interacción en las interfaces gráficas
- Fácil escalado sin modificar el lado cliente
- Estructura modular en sus componentes

Limitaciones

- El sistema falla si está todo centralizado en un servidor
- Alto nivel hardware, es muy costoso

Navegadores web

Cada navegador tiene como mínimo los siguientes componentes: **interfaz gráfica, motor de navegación, motor de visualización, intérprete Javascript, manejador de red y gestor de almacenamiento**.

El **motor de visualización** es el más importante ya que realiza las transformaciones para ver el código HTML..
Cada motor debe interpretar tanto el HTML como las reglas de estilo CSS.

Principales navegadores web

- **Chrome**. Inicialmente se basaba en WebKit. Sin embargo, actualmente utiliza un motor propio llamado Blink.
- **Firefox**. Se basa en el motor Gecko.
- **Safari**. Su motor se basa en WebKit.
- **Edge**. Posee su propio motor de visualización llamado EdgeHTML.

Entornos de programación

- **Sublime Text**. Es uno de los editores de texto más utilizados. Incorpora soporte para el reconocimiento de múltiples lenguajes (no solo de cliente). Está disponible para los principales sistemas operativos de propósito general.
- **Notepad++**. Es un editor de código abierto que reconoce la sintaxis de múltiples lenguajes (actualmente 27).
- **Vim**. Se trata de un editor de texto de propósito general disponible para sistemas UNIX.
- **Visual Studio Code**. Se trata de un editor de código abierto desarrollado por Microsoft. Se encuentra disponible para los sistemas operativos de propósito general. Además, ofrece una gran cantidad de complementos para realizar tareas más allá de las de un simple editor.

Ejercicio práctico: entornos de programación

Descarga e instala el entorno de programación que prefieras. Recomiendo Sublime Text o Visual Studio Code

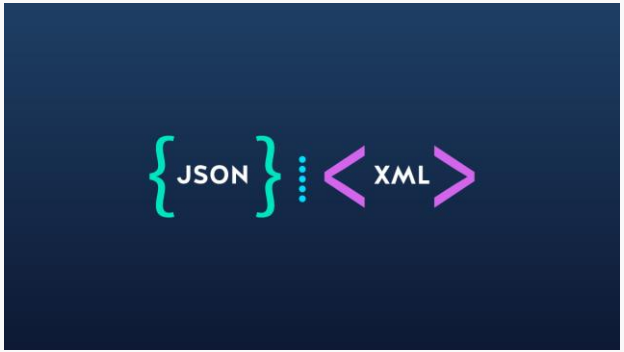
El lenguaje JavaScript

Lenguajes y frameworks, historia de JavaScript, funciones y limitaciones

Lenguajes más usados en clientes web

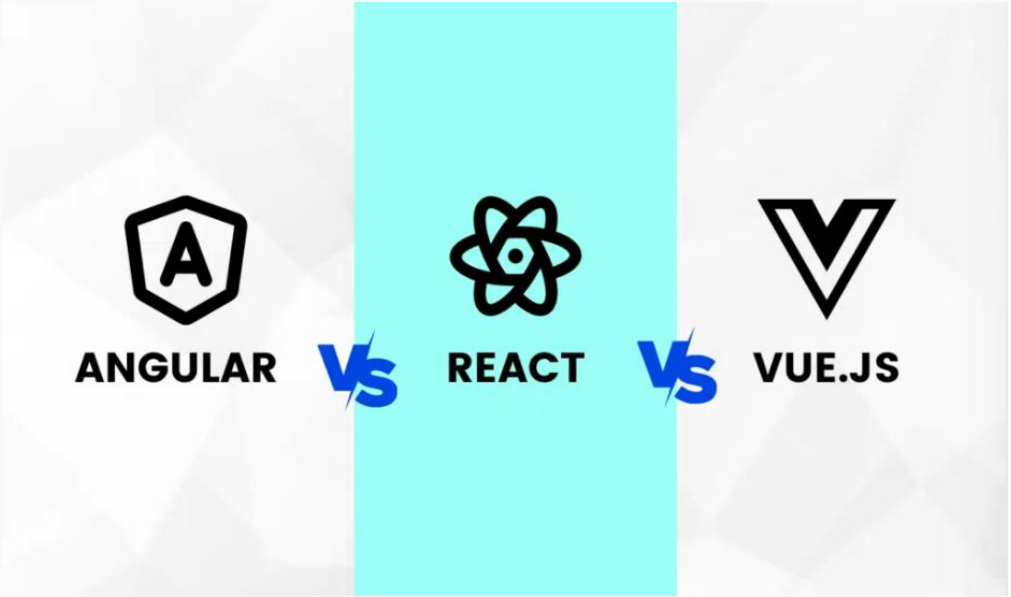
Lenguajes más usados en el desarrollo front-end

Lenguajes involucrados en el intercambio de datos



Frameworks más usados en el desarrollo cliente

Ofrece unas funcionalidades para realizar tareas de forma sencilla.



React: desarrollo reactivo y basado en Javascript. Interfaces sencillas e intuitivas

Angular: mecanismos para crear interfaces gráficas y apps de escritorio y móvil.

Vue.js: interfaces de usuario sencillas y progresivas

Integración de código mediante etiquetas

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>
<h1>My First Heading</h1>
<p>My first paragraph.</p>
</body>
</html>
```

Cabecera: etiqueta <head> incluye metadatos y enlaces a dependencias externas

Cuerpo: etiqueta <body> incluye todo el contenido web

Lenguajes script en entornos web

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Alert</h2>
<button onclick="myFunction()">Try it</button>
<script>
function myFunction() {
 alert("Hola Mundo!!!");
}
</script>
</body>
</html>
```

El código javascript se escribe entre las etiquetas <script>

Renacimiento de JavaScript con la UX

El objetivo de JavaScript es manipular el código HTML y CSS, borrando, mostrando, modificando o añadiendo elementos y su presentación como respuesta a una interacción del usuario con la página web.

Brendan Eich, que trabajaba en el navegador Netscape, desarrolló un lenguaje de programación llamado Mocha, que posteriormente pasó a denominarse LiveScript.

Coincidiendo con la popularidad del lenguaje de programación Java, LiveScript terminó llamándose JavaScript.

Entonces, ¿Qué es ECMA Script?

Renacimiento de JavaScript con la UX

Debido a que Microsoft (Jscript, VBScript) y otros sacaron sus propias versiones, los autores originales desarrollaron un estándar para la ECMA, que es adoptado en la actualidad por todos los navegadores.

TypeScript es un lenguaje desarrollado también por Microsoft que es compatible con JavaScript, ya que los dos se basan en ECMA Script.

Otro lenguaje ECMA Script es Babel, usado en ReactJS. No dejan de ser ampliaciones que convergen en JavaScript.

Fundamentos del lenguaje JavaScript

Desde la web 2.0 podemos crear páginas dinámicas en las que el usuario puede interactuar con facilidad, además, Javascript libera al servidor de ciertas tareas (ej: comparar un formulario en el cliente).

Cómo introducir Javascript en un documento Web

- **Embeber código en HTML**: añadimos Javascript con <script>
- **Archivos externos**: solo contendrían javascript

Se persigue la facilidad de mantenimiento. Se recomienda usar **archivos externos**

```
<body>

  <script src="./script.js"></script>


</body>
```

Dónde poner el código JavaScript

```html
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Mi pagina</title>
    <script src= "archivo.js"></script>
</head>
<body>
    <input type="button" name="boton" value="pulsa"
              onclick="miFuncion()" >
</body>
</html>
```

Estructura HTML/CSS con enlace a archivo Javascript

```javascript
//Este archivo solo tiene codigo Javascript
//No puedo poner nada de HTML o CSS aquí

function miFuncion(){
            codigo Javascript
}
```

archivo.js

Llamada al código .js

Código JavaScript

Otras tareas de Javascript

- Añadir HTML, cambiar contenido y modificar apariencia
- Reaccionar a eventos (clicks, pulsar teclas...)
- Comunicación asíncrono o síncrono
  - Asíncrono: las solicitudes se envían al servidor y la ejecución del código JavaScript continúa sin esperar a que se reciba la respuesta del servidor.
  - Síncrono: JavaScript envía una solicitud al servidor y espera de manera bloqueante hasta que se reciba la respuesta.
- Mecanismo para guardar cookies o almacenamiento local

Limitaciones de Javascript

Estas limitaciones son puramente por cuestiones de seguridad, para evitar el acceso malicioso a determinados datos

- Javascript no puede acceder al disco duro ni ejecutar un programa
- Para interactuar con la cámara o micrófono debe solicitar permiso al usuario
- JavaScript no puede acceder al contenido de una ventana o pestaña diferente a la que está ejecutando si no provienen del mismo dominio. Esto se conoce como Same Origin Policy.

Por ejemplo, sin la Same Origin Policy, un sitio web malicioso podría robar tus cookies de sesión o acceder a tu información en otros servicios en línea.

Sintaxis básica de JavaScript

Sintaxis básica, variables y operadores

Sintaxis básica de JavaScript

- La sintaxis de JavaScript es similar a la de C, C++, Java y otros lenguajes relacionados.
- No se tiene en cuenta espacios en blanco y saltos de línea.
- Distingue las mayúsculas de las minúsculas.
- Existen palabras reservadas del lenguaje (let, var...)
- El punto y coma al final de cada línea no es necesario, pero sí **RECOMENDABLE** para evitar errores.
- Los bloques de código se declaran utilizando llaves {}.

Palabras reservadas

Palabra reservada	Descripción
function	Permite declarar una función
for	Permite crear un bloque que se repita tantas veces como la condición sea cierta
break	Permite salir de un bloque ejecución
switch	Permite elegir qué bloque de sentencias ejecutar a partir de una condición
do...while	Ejecuta un bloque de sentencias y lo repite tantas veces como la condición sea cierta
If...else	Permite ejecutar un bloque de sentencias u otro dependiendo de una condición

Variables

- Una variable es un espacio reservado en el sistema de almacenaje a la cual se le asigna un identificador.
- Utilizamos variables para guardar datos o para asignar valores de unas variables a otra.
- No es necesario establecer el tipo de datos en la declaración de las variables, se ajusta automáticamente.
- El nombre de las variables solo puede contener: letras, números y los símbolos “_” y “$”.
- Los identificadores de las variables no pueden empezar por un número.
- Para declarar las variables se usan: **let** nombreVariable (ámbito local), **var** nombreVariable (ámbito global).
- Para declarar constantes se usa: **const** NOMBRE_CONSTANTE

Sistema de tipado

No es necesario declarar el tipo de variable, por ello, la variable puede tener diferentes tipos

Variable “x” con distintos tipos de datos

```
var x = 10; // x es un número
x = "Hola"; // x ahora es una cadena
x = [1, 2, 3]; // x ahora es un arreglo
```

Tipos primitivos de Javascript

- Boolean: var x = true;
- Null: var x = null;
- Undefined= var x;
- Number: var x = 7;
- String: var x = “Hola Mundo”;

Uso de “typeof”

```
var x = 42;
console.log(typeof x); // Imprime "number"

var nombre = "Juan";
console.log(typeof nombre); // Imprime "string"

var esVerdadero = true;
console.log(typeof esVerdadero); // Imprime "boolean"

var persona = { nombre: "Maria", edad: 30 };
console.log(typeof persona); // Imprime "object"
```

Cuando queramos saber qué dato está usando una variable, usamos **typeof**

Conversión de tipos

Parsing de String a Number

```
var stringValue = "42";
var age = parseInt(stringValue);
```

Comentarios

Comentario de línea

```
var z = 2 ; // Declara z y le asigna 2.
var y = z + 2; // Declara y, y le asigna z + 2
```

Código 6. Comentarios de línea

Comentario multilínea

```
/* 1. Declara z y le asigna 2.
2. Declara y, y le asigna z + 2
/*
var z = 2 ;
var y = z + 2;
```

Código 7. Comentario en bloque

Operadores aritméticos

Operador	Descripción
+	Operador suma
-	Comprador resta
*	Operador de producto
**	Operador de potencia
/	Operador división
%	Operador módulo.
++	Operador de incremento
--	Operador de decremento

Operadores de comparación

Operador	Descripción
==	Operador de igualdad
===	Igualdad de valor y de tipo
!=	Operador distinto
!==	Operador distinto valor o tipo
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que

Operador == Compara si tienen el mismo valor, aunque tengan distinto tipo de dato

```
5 == "5" // true (la cadena "5" se convierte en número)
true == 1  // true (el booleano true se convierte en número)
null == undefined // true
```

Operador === Requiere que los datos tengan el mismo valor y tipo de dato

```
5 === 5 // true
"5" === "5" // true
true === true // true
```

```
5 === "5" // false
true === 1 // false
```

Operadores de asignación y lógicos

Operador abreviado	Ejemplo	Equivalencia
=	A = B	A = B
+=	A += B	A = A + B
-=	A -= B	A = A – B
*=	A *= B	A = A * B
/=	A /= B	A = A / B
%=	A %= B	A = A % B
**=	A **= B	A = A ** B

Operadores de asignación

Operador	Descripción
&&	Y lógico
||	O lógico
!	Negación lógica

Operadores lógicos

Sintaxis básica de JavaScript

- En ciertos aspectos JavaScript es muy flexible, tal vez demasiado.
- A lo largo de los años esa flexibilidad da lugar al uso de malas prácticas de programación que introducen errores que se van arrastrando toda la vida del software.
- Para ello surgió el ECMA Script y existe una manera de obligar a usar ECMA Script aunque JavaScript no obligue explícitamente a ello.
- Tenemos que usar la directiva al principio del fichero

'use strict'

¿Cuál es el valor de cada línea al ejecutar el código?

```
let A, B, C;
A = 5;
B = 3;
C = "5";
(A > B);
(A < B);
(A >= B);
(A <= B);
(A == C);
(A != C);
(A === C);
(A !== C);
(A === B);
(A !== B);
```

```
let A, B, C, D;
A = 5;
B = 3;
C = 6;
D = 1;
(C > A && D < B);
(C > A || D > B);
(!(D > B));
```

Algunos ejercicios para practicar

1. Declara cinco variables numéricas para los valores de la operación y una variable más para el resultado. Genera un script que ejecute la siguiente operación y muestre el valor final de la variable resultado en pantalla: 23 + (15 * (3 / 2)) - 10.

    1.1. Réstale 5 al valor de resultado mediante un operador compuesto y muestra el valor final de la variable resultado en pantalla.

    1.2. Decrementa en uno el valor de resultado mediante el operador decremento y muestra el valor final de la variable resultado en pantalla.

2. Declarar en un script 12 variables con los meses del año y su valor numérico asociado (para enero el valor 1, para febrero el valor 2, etc.)

    2.1. Mostrar por consola el resultado de comparar si enero es mayor que diciembre.

    2.2. Mostrar por consola el resultado de comparar si junio es menor que julio.

    2.3. Mostrar por consola el resultado de comparar si marzo es mayor que febrero y septiembre es mayor que octubre.

    2.4. Mostrar por consola el resultado de comparar si marzo es mayor que febrero o septiembre es mayor que octubre.

3. Crea un script que devuelva la media aritmética de 3 valores, los cuales deben estar guardados en una variable cada uno.

Entrada/Salida de datos

console, alert, prompt, confirm

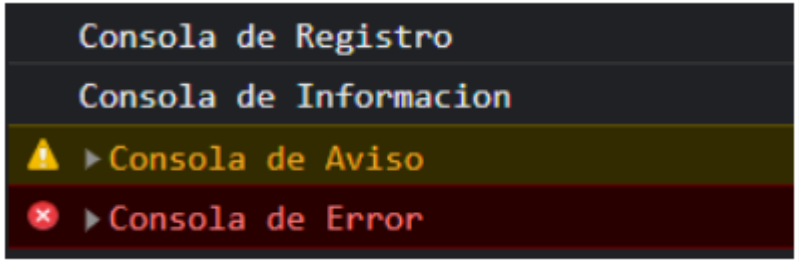
console.log

- Para escribir algo en la consola, existe la función **console.log()** / **console.table()** / etc...
- Los usuarios regulares no ven este output, ya que está en la consola. Para verlo, debemos abrir la consola de desarrolladores.

```
let x = 1;
let y = 2;
let z = 3;
console.log("x:", x, "y:", y, "z:", z);
console.log("Quince es " + (a+b) + " y \nno " + (2*a+b) + ".");
console.log(`Quince es ${a+b} y
    no ${2*a+b}.`);
```

Otras variantes de console.log

```
console.log("Consola de Registro");
console.info("Consola de Información");
console.debug("Consola de Depuración");
console.warn("Consola de Aviso");
console.error("Consola de Error");
```



Falta un mensaje, ¿por qué?

alert

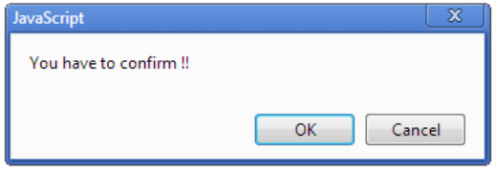
- Muestra una ventana con el mensaje que le haya indicado y es una forma básica de comunicación.

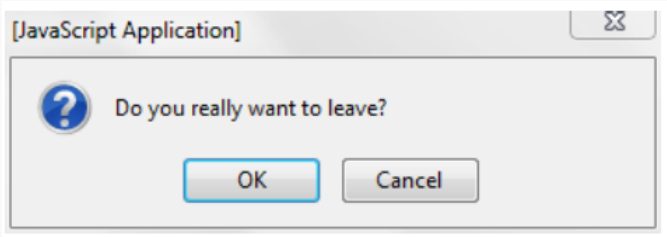
prompt

- La instrucción **prompt** además de mostrar un mensaje permite recoger información del usuario.

confirm

- La orden **confirm** es parecida a la anterior, pero ofreciendo solamente la respuesta de sí o no.

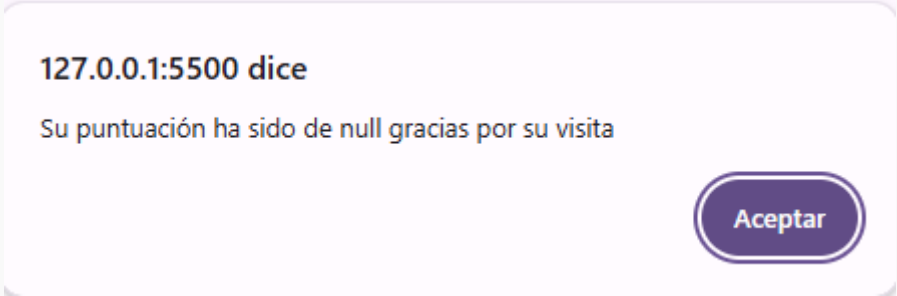




Entrada y salida de datos

- Si en el prompt no escribimos nada se devuelve la cadena vacía o si le damos a cancelar se devuelve null.

```
let respuesta = confirm("Esta satisfecho con la aplicación");
alert(respuesta);
let puntuacion = prompt("Puntue la aplicación del 1 al 10");
alert("Su puntuación ha sido de " + puntuacion + " gracias por su visita");
```



Entrada y salida de datos

- **document** es una variable de tipo objeto que tiene un método llamado **writeln** que permite escribir código HTML en el **body de la página**.
- Por lo tango, cualquier etiqueta HTML que incluyamos se **interpretará** en el navegador. La idea es mezclar información de las variables con el código HTML.

```
document.writeln("Frase desde document write");
document.writeln("<h1>Hola mundo desde un h1</h1>");
let nota = 7;
document.writeln("Tu nota es un <strong>" + nota + "</strong>");
```

Entrada y salida de datos

El código se entiende, pero la concatenación es un poco tediosa.

```
document.writeln("Bienvenido a mi pagina web");
document.writeln("<h1>Bienvenido a mi pagina web</h1>");
let mensaje = prompt("Escriba una frase:");
document.writeln("<h1>" + mensaje + "</h1>");
let nivel = prompt("¿Qué nivel de enbezado desea? (1-6)");
document.writeln("<h" + nivel + "> Bienvenido a mi página web </h" + nivel + ">");
```

Template o plantilla

- Si usamos la comilla backticks o acento grave permite escribir en varias líneas.
- Son muy versátiles porque además pueden contener marcadores, identificados por el signo de dólar y envueltos en llaves **${expresión}.**
- Las expresiones contenidas en los marcadores, junto con el texto entre ellas, son enviados como argumentos a una función.

Template o plantilla

```
let a = 5;
let b = 10;
"Quince es " + (a + b) + " y no " + (2 * a + b) + ".";
// Con templates es más sencillo
`Quince es ${a + b} y no ${2 * a + b}.`;

// Si queremos salto de línea
"Quince es " + (a + b) + " y \nno " + (2 * a + b) + ".";
// Con templates es más sencillo
`Quince es ${a + b} y
no ${2 * a + b}.`;
```

Template o plantilla

Los templates mejoran el código.

```
document.writeln("Bienvenido a mi pagina web");
document.writeln("<h1>Bienvenido a mi pagina web</h1>");
let mensaje = prompt("Escriba una frase:");
document.writeln("<h1>" + mensaje + "</h1>");
let nivel = prompt("¿Qué nivel de enbezado desea? (1-6)");
document.writeln("<h" + nivel + "> Bienvenido a mi página web </h" + nivel + ">");
```

Ejercicio práctico: ejercicios varios

Crea una página web con una hoja de estilos (css). Debes incluir en esa página estos ejercicios de JavaScript

Ejercicio 1: Crear una página con una imagen y un botón. Añadir el código necesario para que cuando pulse el botón salga un `alert ("hola")` y cuando pase el ratón encima de la foto *(onmouseover)* salga un `alert("adios")`.

Ejercicio 2: A través de un prompt, pide el nombre al usuario y saludalo con un alert de la siguiente forma: *"Bienvenid@ a mi página XXXXXX"* (siendo XXXXXX el nombre que ha introducido el usuario).

Ejercicio 3: Crear una página en JavaScript que:

- Pida al usuario mediante prompt el nombre del producto que quiere comprar.
- Pida el precio del producto.
- Pida la cantidad que desea comprar.
- Calcule el total a pagar usando operadores matemáticos.
- Muestre por consola la información completa (producto, cantidad y total).
- Use confirm para preguntar al usuario si desea confirmar la compra.
- Muestre si el usuario ha confirmado o no la compra, junto con la información de lo que se está comprando, en un alert.
- Añada una sección en la página web en la que se muestre la información de la compra (usa document.writeln).

Nota: haz el código JS en un archivo aparte, no lo hagas en el archivo.html

String

Operaciones básicas de String

Operaciones básicas de String

```
let texto = "Un texto con 'comillas' dentro";
let otro_texto = 'Otro texto con "comillas" dentro';
let numero_letras = texto.length;

let prueba1 = "Hola " + "Mundo";
let prueba2 = "Nota: " + 10;
let prueba3 = "7" + 5;
```

Operaciones básicas de String

- Para acceder a un carácter en la posición pos, se debe usar **[pos]** o llamar al método **str.charAt(pos)**.
- El primer carácter comienza desde la posición cero:

```
let str = "Hola";

// El primer carácter
str[0]; // H
str.charAt(0); // H

// El último carácter
str[str.length -1]; // a
```

Operaciones de String

- **length**: Devuelve la longitud de la cadena.
- **charAt(indice):** Devuelve el carácter en una posición específica de la cadena.
- **charCodeAt(indice):** Devuelve el valor Unicode del carácter en una posición específica de la cadena.
- **concat():** Combina dos o más cadenas y devuelve una nueva cadena.
- **indexOf():** Devuelve el primer índice en el que se encuentra un substring dado, o -1 si no se encuentra.
- **lastIndexOf():** Devuelve el último índice en el que se encuentra un substring dado, o -1 si no se encuentra.
- **slice():** Extrae una porción de la cadena y devuelve una nueva cadena.
- **substring(indiceInicio, longitud):** Similar a slice(), pero no acepta índices negativos.
- **substr():** Extrae una cantidad específica de caracteres a partir de una posición dada.
- **replace(patron, reemplazo):** Reemplaza un substring con otro en la cadena.
- **toUpperCase():** Convierte la cadena a mayúsculas.
- **toLowerCase():** Convierte la cadena a minúsculas.
- **trim():** Elimina los espacios en blanco al principio y al final de la cadena.
- **split(delimitador):** Divide la cadena en un array de substrings en función de un delimitador.
- **includes(subcadena):**comprueba si una cadena contiene una subcadena y devuelve un valor booleano (true o false).
- **match(patron):** Busca una expresión regular en la cadena y devuelve un array de resultados.

Arrays

Operaciones básicas de Arrays

Arrays

- JavaScript cuenta con el tipo **Array** como un tipo básico.
- Un **Array** es una **colección de variables**, que se engloba en una misma variable.
- Los Arrays **son dinámicos** por lo que pueden cambiar de tamaño y pueden incluir cualquier tipo de datos.
- Los Arrays empiezan a numerarse en 0 y tiene la propiedad **length** como el tipo string.
- De hecho, un **string** es un caso particular de un **Array**.
- De manera práctica, son conjuntos de datos almacenados bajo el mismo nombre.
- Una representación gráfica de un array sería la siguiente

Posición	0	1	2	3
Valor	25	36	21	58

Arrays

- Para crear el array representado en la imagen anterior es necesario este código.

```
let numeros = [];
numeros[0] = 25;
numeros[1] = 36;
numeros[2] = 21;
numeros[3] = 58;
```

- Un array en JavaScript no tiene un tamaño predefinido, con lo que puede tener tantos valores como sea necesario.

Operaciones básicas de un Array

```
let nombreArray = [];
let diasLectivos = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes"];
alert(dias[0]);
alert(dias.length);
let variado = ["Dia", 45, 32.78, true];
let elMartes = diasLectivos[1];

// Un string también es un array
let nombre = "Pepe"
alert(nombre[2]);
```

Crear HTML desde un array

Posición	0	1	2	3
Valor	León	Seat	5	V6

```
let coche = [];
coche[0] = "Leon";
coche[1] = "Seat";
coche[2] = 5;
coche[3] = "V6";

document.writeln("<ul>" +
    "<li>" + coche[0] + "</li>" +
    "<li>" + coche[1] + "</li>" +
    "<li>" + coche[2] + "</li>" +
    "<li>" + coche[3] + "</li>" +
    "</ul>"
);
```

Crear HTML desde un array

- Los templates funcionan muy bien con los arrays.

```
let coche = ["Leon", "Seat", 5, "V6"];
document.writeln(`<ul>
    <li>${coche[0]}</li>
    <li>${coche[1]}</li>
    <li>${coche[2]}</li>
    <li>${coche[3]}</li>
    </ul>
`);
```

Operaciones de Array

- **length**: Devuelve la cantidad de elementos en un array.
- **push():** Agrega uno o más elementos al final del array y devuelve la nueva longitud.
- **pop():** Elimina el último elemento del array y lo devuelve.
- shift(): Elimina el primer elemento del array y lo devuelve.
- **unshift():** Agrega uno o más elementos al principio del array y devuelve la nueva longitud.
- **concat():** Combina dos o más arrays y devuelve un nuevo array.
- **join(separador):** Une todos los elementos de un array en una cadena.
- **slice(indiceInicio, indiceFin):** Retorna una copia superficial de una porción del array en un nuevo array.
- **splice(indice, cantidad, elemento1, elemento2, ...):** Cambia el contenido de un array eliminando, reemplazando o agregando elementos.
- **indexOf():** Devuelve el primer índice en el que se encuentra un elemento dado, o -1 si no se encuentra.
- **includes(subcadena):**comprueba si una cadena contiene una subcadena y devuelve un valor booleano (true o false).
- **forEach():** Ejecuta una función en cada elemento del array.
- **map():** Crea un nuevo array con los resultados de aplicar una función a cada elemento del array original.
- **filter():** Crea un nuevo array con todos los elementos que cumplan una condición.
- **reduce():** Aplica una función a un acumulador y a cada elemento del array (de izquierda a derecha) para reducirlo a un solo valor.
- **sort():** Ordena los elementos de un array en su lugar y devuelve el array.
- **reverse():** Invierte el orden de los elementos de un array en su lugar.

Estructuras condicionales

If-elseif-else, switch

Condicionales (if-else)

```
var result = 0
if (today > 10) {
 result++ ;
}else{
 result--;
}
```

Código 4. Sentencia if-else

Condicionales (Switch)

```
switch (today) {
  case 0:
    day = “Lunes”;
    break;
  case 1:
    day = “Martes”;
    break;
  default: {
        day =”error”;
        break;
  }
}
```

Código 5. Ejemplo de funcionamiento sentencia switch

Estructuras repetitivas

Bucles for, bucles while, bucles do while

Bucles (for)

```
for (i = 0; i < 5; i++) {
  text += “El número es “ + i + “<br>”;
}
```

Código 6. Bucle for

Bucles (while)

```
while (i < 10) {
  text += “El número es “ + i;
  i++;
}
```

Código 7. Ejemplo texto de tipo de bucle

El código se ejecuta de 0 a N veces

Bucles (while)

```
do {
  text += “El número es “ + i;
  i++;
}
while (i < 10);
```

Código 8. Ejemplo bucle do-while

El código se ejecuta de 1 a N veces

Ejercicio práctico: ejercicios varios

Ejercicio 1: Encuesta de nombres

Crea un programa en JavaScript que:

- Pida al usuario con prompt cuántos nombres quiere ingresar.
- Pida uno a uno los nombres con un prompt
- Guarde esos nombres en un array.
- Use un condicional para verificar si el usuario no ingresó nada y mostrar un alert de error.
- Al final, muestre en consola la lista completa de nombres y también la cantidad total ingresada.
- Muestra con alert un mensaje con el primer y el último nombre de la lista.

Ejercicio 2: Tienda virtual simple

Crea un programa que:

- Tenga un array con al menos 3 productos (ejemplo: ["camisa", "pantalón", "zapatos"]).
- Pida al usuario con prompt cuál producto quiere comprar.
- Use un condicional para verificar si el producto está en el array:
- Si existe, pide con otro prompt la cantidad y calcula el total (suponiendo un precio fijo).
- Si no existe, muestra un alert que diga “Producto no disponible”.
- Usa confirm para preguntar si desea confirmar la compra y muestra el resultado en un alert.

Ejercicio 4: Juego del número secreto

Crea un programa que:

- Genere un número secreto entre 1 y 10.
- Dé al usuario 3 intentos para adivinarlo usando un bucle.
- En cada intento, pide un número con prompt y:
- Si acierta → muestra un alert con “¡Ganaste!” y termina el juego.
- Si no acierta → muestra un alert indicando si el número secreto es mayor o menor.
- Si se acaban los intentos, muestra un alert con el número correcto.
- Al final, muestra en console.log el número secreto y todos los intentos del usuario (guardados en un array).

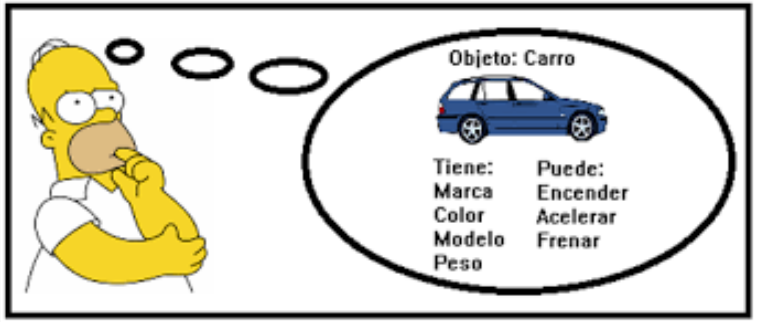
POO y clases core de JavaScript

POO y clases core de JavaScript

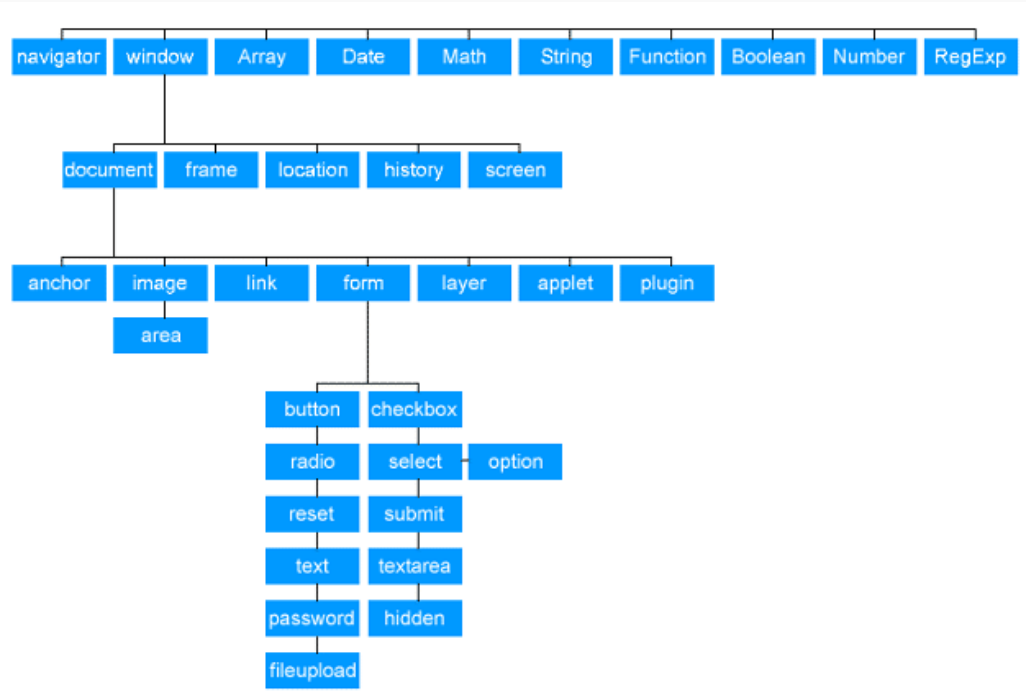
- Aunque no lo creamos, hemos estado usando programación orientada a objetos en JavaScript sin darnos cuenta.
- **alert, prompt y confirm** son órdenes pertenecientes al objeto window de JavaScript.
- Por supuesto **document.writeln** también hace uso de la POO, indicando que el método writeln pertenece al objeto document.
- **JavaScript** es un lenguaje **orientado a objetos** (no es puro, pero casi), y por tanto necesitamos familiarizarnos con sus objetos.

¿Qué es la POO?

- Paradigma de la programación -> aplicaciones más organizadas, escalables y mantenibles.
- Se basa en la creación de objetos que representan entidades del mundo real.
- Se sustenta en 4 pilares fundamentales: abstracción, encapsulamiento, herencia y polimorfismo.
- Hasta ahora conocéis el modelo de clases. En JavaScript se utilizan sobre todo objetos.



Jerarquía de objetos JavaScript



Operador . (punto)

```
//Caracteristicas
document.body
document.title
document.URL
document.links //array de enlaces
document.images //array de imagenes
document.cookie

//Capacidades
document.close();
document.open();
document.getElementByName("a");
document.write("<h1>Hola Mundo</h1>");

//Modificar caracteristicas
document.title="Titulo cambiado con JavaScript";
document.body.style.backGroundColor="cyan";
document.images[0].style.width="500px";
```

La realidad del const

```
//Probar el siguiente codigo
const lista=[1,2,3,4,5];
lista.push(6);
console.log(lista);
//El array muta aun estando declarado con const

//En cambio si probamos
const lista=[1,2,3,4,5];
lista=[9,8,7];
//Da error

//const realmente es una variable igual que let
//pero que solo permite una sola asignacion
//Si vamos a tener un objeto javascript en una variable
//si solo si unicamente vamos a usar sus metodos
//y no reasignamos a esas variable lo mejor sin duda
//ES DECLARARLA const que es más eficiente que let
//En el tema 3 ahondaremos sobre ello
```

Clase Math

```
//Caracteristicas
Math.PI
Math.E

//Capacidades
Math.sqrt(4); //Raiz cuadrada
Math.pow(2,5); //Potencia base exponente
Math.abs(-7); //Valor absoluto
Math.random(); //Numero aleatorio decimales entre 0 y 0,9999
Math.floor(Math.random()*100); //Numero entero entre 0 y 99
Math.floor(Math.random()*100+1); //Numero entero entre 1 y 100
```

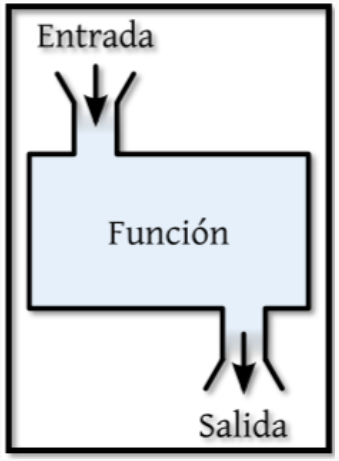
Clase Number

```
//Son metodos relacionados a números
let num = 12.36;
console.log(num.toFixed(1)); // "12.4"
let a = 3.3445;
let c = a.toFixed(1); /* resultado -> 3.3 */
let g = a.toFixed(4); /* resultado -> 3.3445 */
let g = a.toFixed(7); /* resultado -> 3.3445000 */
console.log( isNaN(NaN) ); // true
console.log( isNaN("str") ); // true
console.log( isNaN(NaN) ); // true
console.log( isNaN("str") ); // true
console.log( parseInt('100px') ); // 100
console.log( parseFloat('12.5em') ); // 12.5
console.log( parseInt('12.3') ); // 12
console.log( parseFloat('12.3.4') ); // 12.3
let entero = 10;
let decimal = parseFloat(int).toFixed(2); // 10.00
```

Modularización de código.
Las funciones

Funciones

- Una función es una herramienta que nos permite definir una serie de instrucciones como si fuera solo una evitando así duplicar código.
- Utilizar funciones permite modularizar el código y hacer que sea más entendible, además de ser necesario para la gestión de eventos.



Funciones

```
function nombreFuncion(parametro1,parametro2,...)
{
    ...
}
```

```
function SumarIVA(cantidad,porcentaje)
{
   let  total;

   total=cantidad+cantidad*porcentaje/100;
   alert(total);
}
SumarIVA(400,18);
```

```
function ElMayor(num1,num2)
{
   if(num1>num2)
   {
      alert(num1+" es el mayor");
   }else{
      alert(num2+" es el mayor");
   }
}
ElMayor(4,7);
```

Invocando funciones

Javascript permite invocar funciones y variables antes de su declaración ya que Javascript puede mover las declaraciones al principio del código. Esto es el llamado “hoisting”

```
ejemploFuncion(5);

// Declaración de la función
function ejemploFuncion(x){
    return x*x
}
```

Diseño modular de funciones

- Se recomienda no incluir entrada/salida en funciones para que se puedan reutilizar.

```
function SumarIVA(cantidad,porcentaje)
{
   let  total;

   total=cantidad+cantidad*porcentaje/100;
   return total;
}

let resultado=SumarIVA(400,18);
//Puedo sacarla por pantalla por HTML o lo que quiera
```

```
function ElMayor(num1,num2)
{
   let mayor;
   if(num1>num2)
   {
      mayor=num1;
   }else{
      mayor=num2;
   }
   return mayor;
}

let ganador=ElMayor(4,7);
//Puedo sacarla por pantalla por HTML o lo que quiera
```

Creando HTML con funciones modulares

```
function ejemploModular(datos, tipo){
    let res;
    if (tipo === "button"){
        res = "";
        for(let valor of datos){
            res += `<input type="button" value="${valor}"><br>`;
        }
    } else if (tipo === "select"){
        res="<select>";
        for(let valor of datos){
            res += `<option value="${valor}">${valor}</option>`;
        }
        res += "</select>";
    } else {
        res = "<h2>Tipo incorrecto</h2>";
    }

    return res;
}
```

Parámetros opcionales

```
//Maneras tradicionales
function mostrarMensaje(persona) {
  if(texto === undefined){
    texto="sin texto dado";
  }
  console.log(persona + ": " + texto);
}

function mostrarMensaje(persona) {
  //texto undefined se considera false
  texto=texto || "sin texto dado";
  console.log(persona + ": " + texto);
}

//Con ECMA6

function mostrarMensaje(persona, texto = "sin texto dado") {
  console.log(persona + ": " + texto);
}

mostrarMensaje("Daniel"); // Daniel: sin texto dado
```

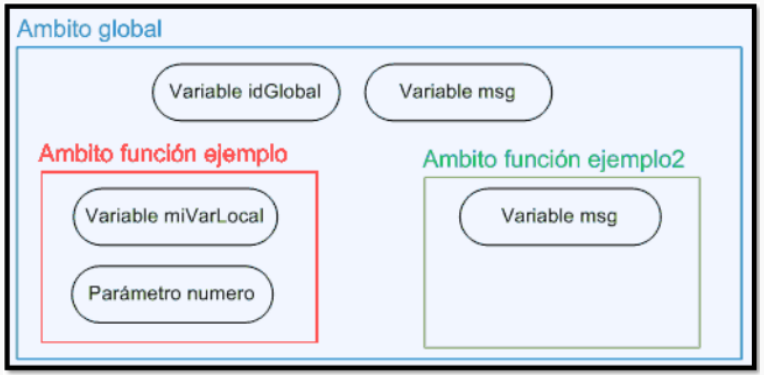
Ámbito de las variables

- El ámbito de una variable, es la zona del programa donde existe la variable y se puede operar con ella.
- **Variables locales:** Son variables como el contador de un bucle for o declaradas dentro de una función. No se pueden acceder desde fuera de su bloque.
- **Variables globales:** Son variables que no se han definido dentro de ningún bloque y por tanto están disponibles en cualquier parte de la aplicación.

```
let idGlobal=33;
let msg = 'Variable global';

function ejemplo(numero) {
 let miVarLocal = 'Soy una variable local';
 ejemplo2();
}

function ejemplo2(){
  let msg = 'Mensaje: '+idGlobal;
}
```



Programación funcional

La programación funcional es un paradigma de programación que se basa en gran medida en el uso de funciones puras y aisladas.

Una **función pura** es una función que no tiene efectos secundarios.

Algunos tipos de funciones en JavaScript

- **Funciones de primera clase:** en JavaScript todas las funciones se consideran de primera clase. Las funciones de primera clase son funciones que pueden asignarse como valores a variables, devolverse desde otras funciones y pasarse como argumentos a otras funciones.
- **Funciones Callback:** las funciones callback son funciones que se pasan a otras funciones como argumentos y son llamadas por la función en la que se pasan.
  Simplemente, las funciones callback son funciones que escribimos como argumentos en otras funciones.
  No podemos invocar funciones callback. Se invocan cuando se llama a la función principal en la que se pasaron como argumentos.
- **Funciones de orden superior:** son funciones que reciben otras funciones como argumentos o devuelven una función.

Programación funcional

```
//Las funciones pueden ser parametros de otra funcion
function hablar(entonacion){
    let frase=...;
    entonacion(frase);
}

hablar(exclamar);

//No es necesario que este en una variable o const
hablar(function (mensaje){
    console.log("¡¡"+mensaje+"!!")
});
```

Programación funcional

```
//Podriamos tener distintas funciones
const preguntar=function (mensaje){
    console.log("¿¿"+mensaje+"??")
}

const neutro=function (mensaje){
    console.log(mensaje);
}

const html=function (mensaje){
    console.log("<h1>"+mensaje+"</h1>");
}
```

Objetos JavaScript vs JSON

Orientación a objetos en JavaScript

Javascript no está basado en el concepto de clase, su estilo se basa en definir las propiedades de un nuevo objeto.
Soporta la herencia entre objetos, haciendo uso del puntero “this” (que hace referencia al objeto actual).
Se puede crear un objeto de diferentes formas:

- Definiendo un constructor.
- Utilizando el objeto genérico “Object”.
- Usando llaves y pares-valor (objeto literal).

Objetos JavaScript

```
// Definición de un constructor llamado "Persona"
function Persona(nombre, edad) {
    this.nombre = nombre;
    this.edad = edad;
}

// Crear una instancia de Persona
let persona1 = new Persona("Juan", 30);
console.log(persona1.nombre); // Salida: "Juan"
console.log(persona1.edad);   // Salida: 30

// Crear otra instancia de Persona
let persona2 = new Persona("María", 25);
console.log(persona2.nombre); // Salida: "María"
console.log(persona2.edad);   // Salida: 25
```

Crear objetos usando un constructor

Objetos JavaScript

```
// Crear un objeto utilizando Object
let coche = new Object();
coche.marca = "Toyota";
coche.modelo = "Corolla";
coche.año = 2022;

console.log(coche.marca); // Salida: "Toyota"
console.log(coche.modelo); // Salida: "Corolla"
console.log(coche.año);   // Salida: 2022
```

Crear objetos usando “Object”

Objetos JavaScript

```
var persona = { nombre: "Fran", apellido: "Pérez" };

console.log(persona.nombre); // Salida: "Fran"
console.log(persona.apellido); // Salida: "Pérez"
```

Crear objetos usando llaves pares-valor


```
const micoche={
    name:"Fiat",
    model:"500",
    weight: 850,
    color:"white"
};
```

Manejo básico de un objeto

```
//Mostrar informacion del objeto
console.log(micoche.name);
console.log(micoche["color"]);

//Modificar informacion del objeto
micoche.model="500C";
micoche["model"]="500C";
//Propiedades nuevas
micoche.doors=3;
micoche["owner"]="Juan"
//borrar propiedades
delete micoche["owner"]
delete micoche.doors

//Objeto sin propiedades
const bolsa={};
//Propiedades nuevas
bolsa.tomate=10
fruta="manzana";
bolsa[fruta]=6;
```

Más formas de crear objetos

```
let fruta="manzana";

{  //la propiedad se obtiene de fruta
    fruta:6,
}

let cantidad=90;

{   //El valor se obtiene de cantidad
    manzana:cantidad,
}

{   //la propiedad y valor se obtienen de varibles
    fruta:cantidad,
}
```

Más formas de crear objetos

```
let nombre="Alfonso";
let edad=39;
const persona={
    nombre,edad
}
```

```
//construirá
{
  nombre:"Alfonso",
  edad:23
}
```

Operador in

```
const user = { name: "John", age: 30 };
console.log( "age" in user );
// mostrará "true", porque user.age sí existe
console.log( "blabla" in user );
// mostrará false, porque user.blabla no existe
```

Propiedades y métodos

```
let persona = { nombre: "Fran", apellido: "Pérez", edad: 30 };

//Borrar propiedad edad
delete persona.edad

//Métodos en objetos (usando par-valor)
let persona = {
        name:”Fran”,
        surname:”Pérez”,
        age:36,
        state:”Spain”,
        fullInfo: function(){ //Muestra la info del objeto
                return this.name + “ “+ surname + “ “ + age+ “ “+ state;
        }
};
```

Se pueden almacenar funciones como valores de propiedad

Iterar sobre las propiedades de un objeto

```
const usuario = {
  name: "John",
  age: 30,
  isAdmin: true
};

for (let clave in usuario) {
  // claves
  console.log(clave);  // name, age, isAdmin
  // valores de las claves
  console.log(usuario[clave]); // John, 30, true
}
```

Ideas de uso de un objeto

```
const trabajadores = {
  John: 100,
  Ann: 160,
  Pete: 130
};

let suma = 0;
for (let trabajador in trabajadores) {
  suma += trabajadores[trabajador];
}

console.log(suma); // 390
```

```
const prefijos={
    españa:"+0034",
    spain:"+0034",
    es:"+0034",
    portugal:...
}
```

Recorrido de un array de objetos

```
const almacen=[
    {
     name:"Fiat",
     model:"500",
     weight: 850,
     price:100000,
     color:"white"
    },
    {
     name:"Renault",
     model:"R5",
     weight: 1000,
     price: 50000,
     color:"yellow"
    }
    ...
```

```
//Mostrar informacion objeto de un array
for (let i=0;i<almacen.length;i++)
{
    console.log(`Coche numero: ${i} se llama ${almacen[i]["name"]}`);
}

//Con ecma 6 algo más simplificado
for (let coche of almacen)
{
    console.log(`Coche: ${coche["name"]}`);
}
```

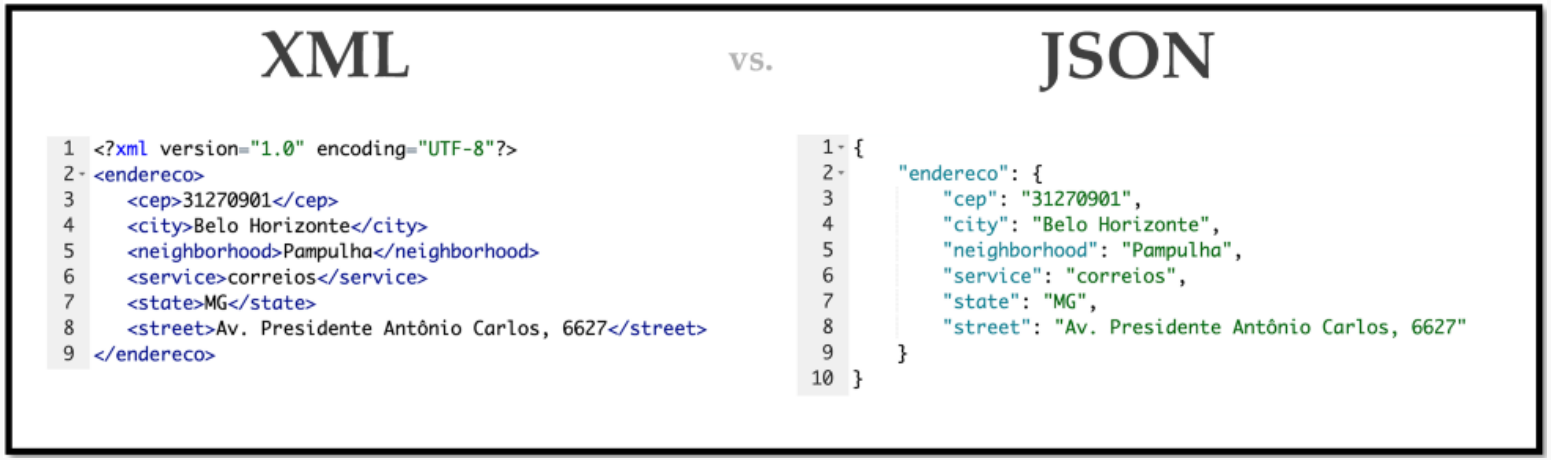
Recorrido de un array de objetos

```
//Recorrido completo
for (let coche of almacen)
{
    for (let propiedad of coche){
        ...
        switch(propiedad){
         case "name":
           ...
         case "weight":
           console.log(coche[propiedad]+"Kg");
           break;
         case "price":
           console.log(coche[propiedad]+"€");
           break;
        }
    }
}

//En el siguiente tema veremos formas
//mas compactas de refactorizar este codigo
```

¿Qué es JSON?

- JavaScript Object Notation
- Notación de Objetos de JavaScript es un formato ligero de intercambio de datos.
- Por lo tanto, tiene el mismo objetivo que XML, o sea, es texto que es un formato universal y está soportado por virtualmente todos los lenguajes de programación



¿Qué es JSON?

No deja de ser un **String** que contiene **información estructurada** como en el lenguaje JavaScript.

JSON está constituido por dos estructuras:

- Una colección de **pares de nombre/valor**. En varios lenguajes esto es conocido como un objeto, registro, estructura, diccionario, tabla hash, lista de claves o un arreglo asociativo.
- Una **lista ordenada de valores**. En la mayoría de los lenguajes, esto se implementa como arreglos, vectores, listas o secuencias.

Lo único que no pueden contener son funciones, por cuestiones lógicas (todo se debe transformar a String)

Objetos vs JSON

```
const almacen=[
    {
     name:"Fiat",
     model:"500",
     weight: 850,
     price:100000,
     color:"white"
    },
    {
     name:"Renault",
     model:"R5",
     weight: 1000,
     price: 50000,
     color:"yellow"
    }
    ...
```

```
`[{"name":"Fiat","model":"500","weight":850,"price":100000,"color":"white"},
  {"name":"Renault","model":"R5","weight":1000,"price":50000,"color":"yellow"},
  {"name":"Peugeot","model":"307","weight":600,"price":35000,"color":"black"}]`
```

JSON.stringify

```
const almacen=[
    {
     name:"Fiat",
     model:"500",
     weight: 850,
     price:100000,
     color:"white"
    },
    {
     name:"Renault",
     model:"R5",
     weight: 1000,
     price: 50000,
     color:"yellow"
    }
    ...
```

```
JSON.stringify(almacen);

`[{"name":"Fiat","model":"500","weight":850,"price":100000,"color":"white"},
 {"name":"Renault","model":"R5","weight":1000,"price":50000,"color":"yellow"},
 {"name":"Peugeot","model":"307","weight":600,"price":35000,"color":"black"}]`
```

JSON.parse

```
JSON.parse(`[{"name":"Fiat","model":"500","weight":850,"price":100000,"color":"white"},
             {"name":"Renault","model":"R5","weight":1000,"price":50000,"color":"yellow"},
             {"name":"Peugeot","model":"307","weight":600,"price":35000,"color":"black"}]`);
```

```
const almacen=[
  {
   name:"Fiat",
   model:"500",
   weight: 850,
   price:100000,
   color:"white"
  },
  {
   name:"Renault",
   model:"R5",
   weight: 1000,
   price: 50000,
   color:"yellow"
  }
  ...
```

¿Para qué se utiliza JSON?

La fuente de datos en formato JSON suelen ser servidores de bases de datos que transforman bases de datos de SQL a JSON.

El hecho de que varios sistemas entiendan JSON al ser información en formato texto le convierten en la mejor forma de intercambia información entre aplicaciones.

Esto da lugar al concepto de API (interfaz de programación de aplicaciones) que son mecanismo ordenados para leer/escribir información en servidores en formato JSON.

Ejemplos de API

- https://www.rtve.es/api/noticias.json
- https://pokeapi.co/
- https://api.chucknorris.io/
- https://openweathermap.org/api
- https://github.com/justadudewhohacks/face-api.js
-

¿Para qué se utiliza JSON?

Hasta que no veamos AJAX y la comunicación asíncrona no podemos utilizar todas las posibilidades.

En este caso lo que nos va a interesar es manejarnos con JSON y lo que haremos es usar ficheros locales que tiene los datos en formato JSON.

Otras variaciones del console para depurar código rapidamente

```
for(i=0; i<10; i++){
  console.count('Contador')
}
console.countReset('Contador')
```

```
Contador: 1
Contador: 2
Contador: 3
Contador: 4
Contador: 5
Contador: 6
Contador: 7
Contador: 8
Contador: 9
Contador: 10
```

```
array= [{nombre:"pedro",edad:20},{nombre:"ana",edad:30}];
console.table(array);
```

(índice)	nombre	edad
0	'pedro'	20
1	'ana'	30

▶ Array(2)

Bibliografía

- Mozilla MDN WebDocs
  https://developer.mozilla.org/es/
- Scrimba.
  https://www.scrimba.com