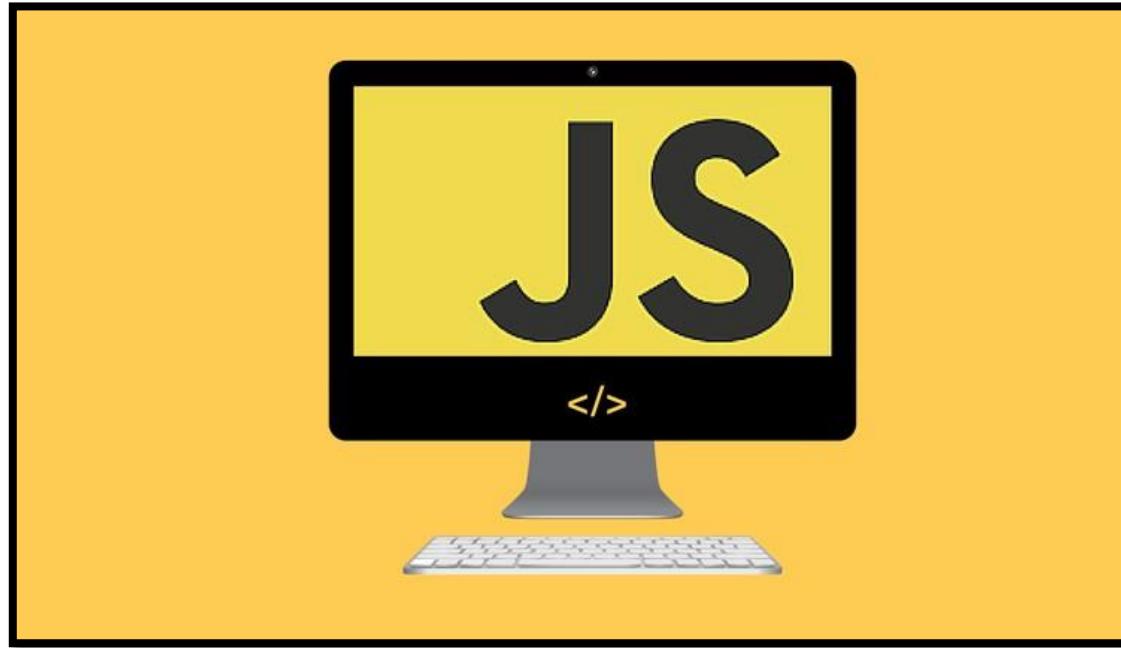


Desarrollo de Aplicaciones Web



Tema 1. Introducción a JavaScript

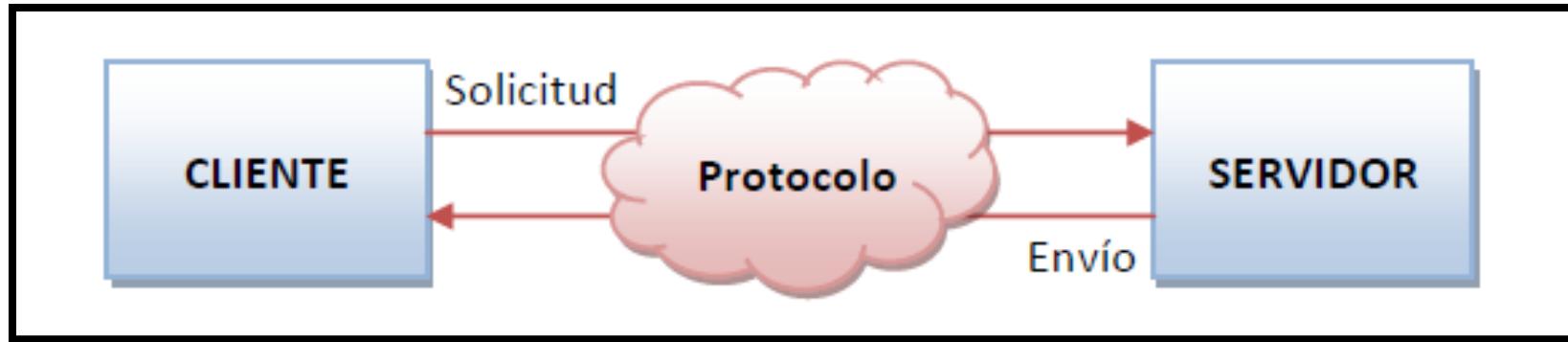
Índice de contenidos

- Front-End VS Back-End
- El lenguaje JavaScript. Entorno RunJS
- Sintaxis básica. Variables y operadores
- String y Arrays
- POO. Clases Core de JavaScript
- if - elseif - else y switch
- while, do while y for
- Funciones
- Objetos JavaScript vs JSON. Bucles for-each

Front-End VS Back-End

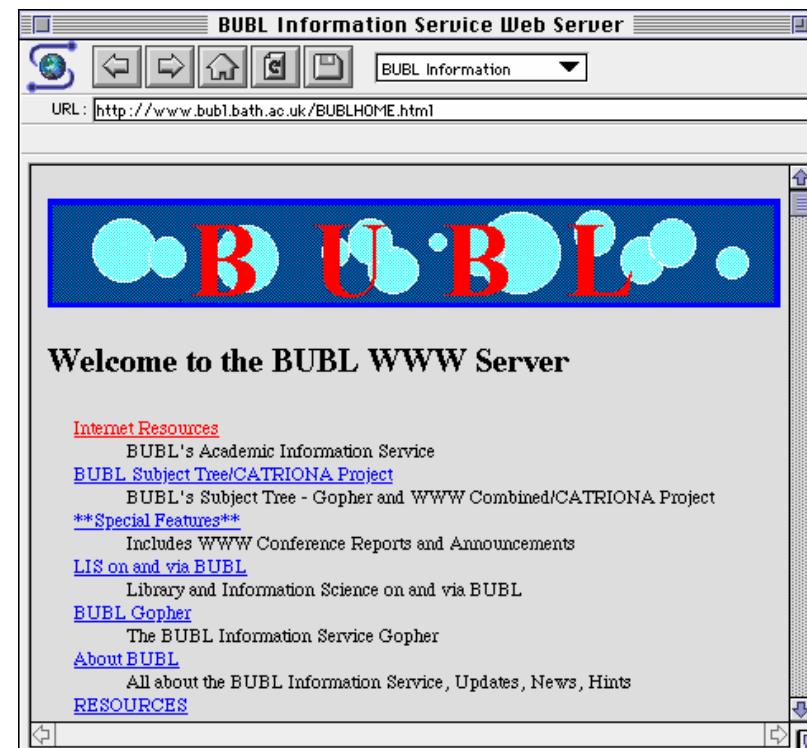


- La configuración arquitectónica más habitual es la denominada **Cliente-Servidor**.
- El **cliente** es un componente *consumidor* de servicios.
- El **servidor** es un proceso *proveedor* de servicios.

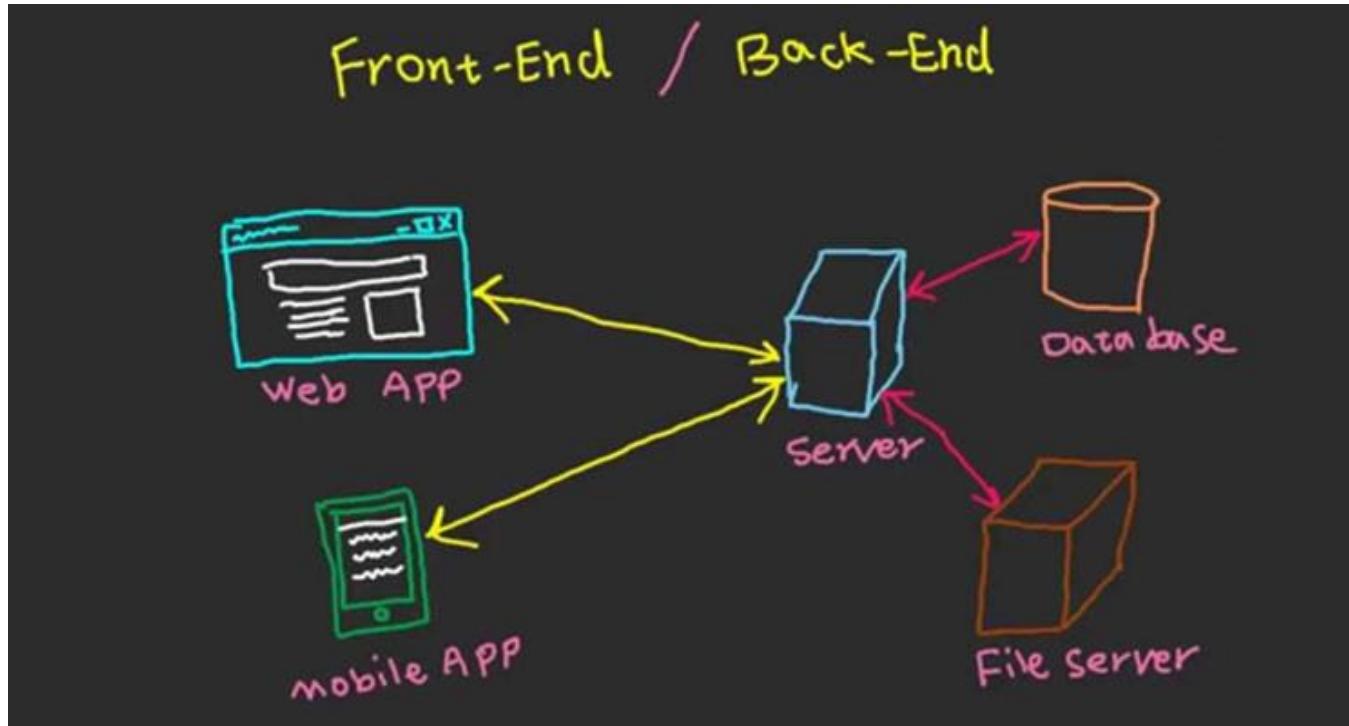


- El componente software que se utiliza en el cliente es el **navegador web**.
- Permite acceder al contenido ofrecido por los **servidores de Internet** sin la necesidad de que el usuario instale un nuevo programa.
- El **navegador web** permite a un usuario acceder y visualizar a un recurso de un servidor Web mediante una **dirección URL (Universal Resource Locator)**.
- En la actualidad dicho recurso puede incluir **programas ejecutables**.

- Existen multitud de navegadores web disponibles:
 - Mosaic.
 - Netscape Navigator (después Communicator).
 - Internet Explorer.
 - Mozilla Firefox.
 - Google Chrome.
 - Safari.
 - Opera.

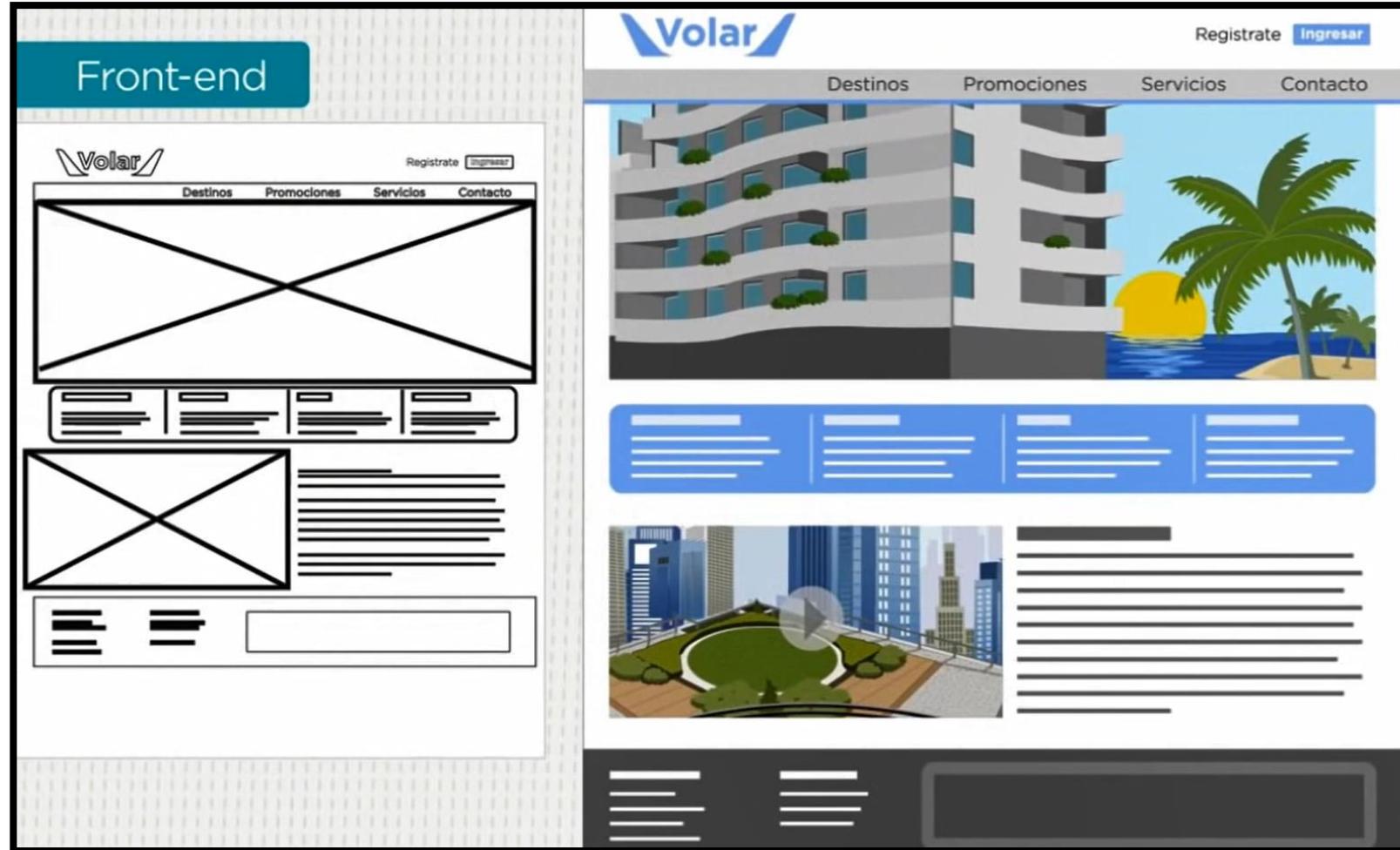


- En toda aplicación web existen dos partes distinguibles. Se trata del **front-end** y el **back-end**.
- El front-end es la parte del software que **interactúa con los usuarios** y el back-end es la parte que **procesa la información** que entra desde el front-end.
- El front-end es el responsable de recolectar los datos de entrada del usuario, y **los transforma ajustándolos a las especificaciones** que demanda el back-end para poder procesarlos, devolviendo generalmente una respuesta que el front-end recibe y expone al usuario.

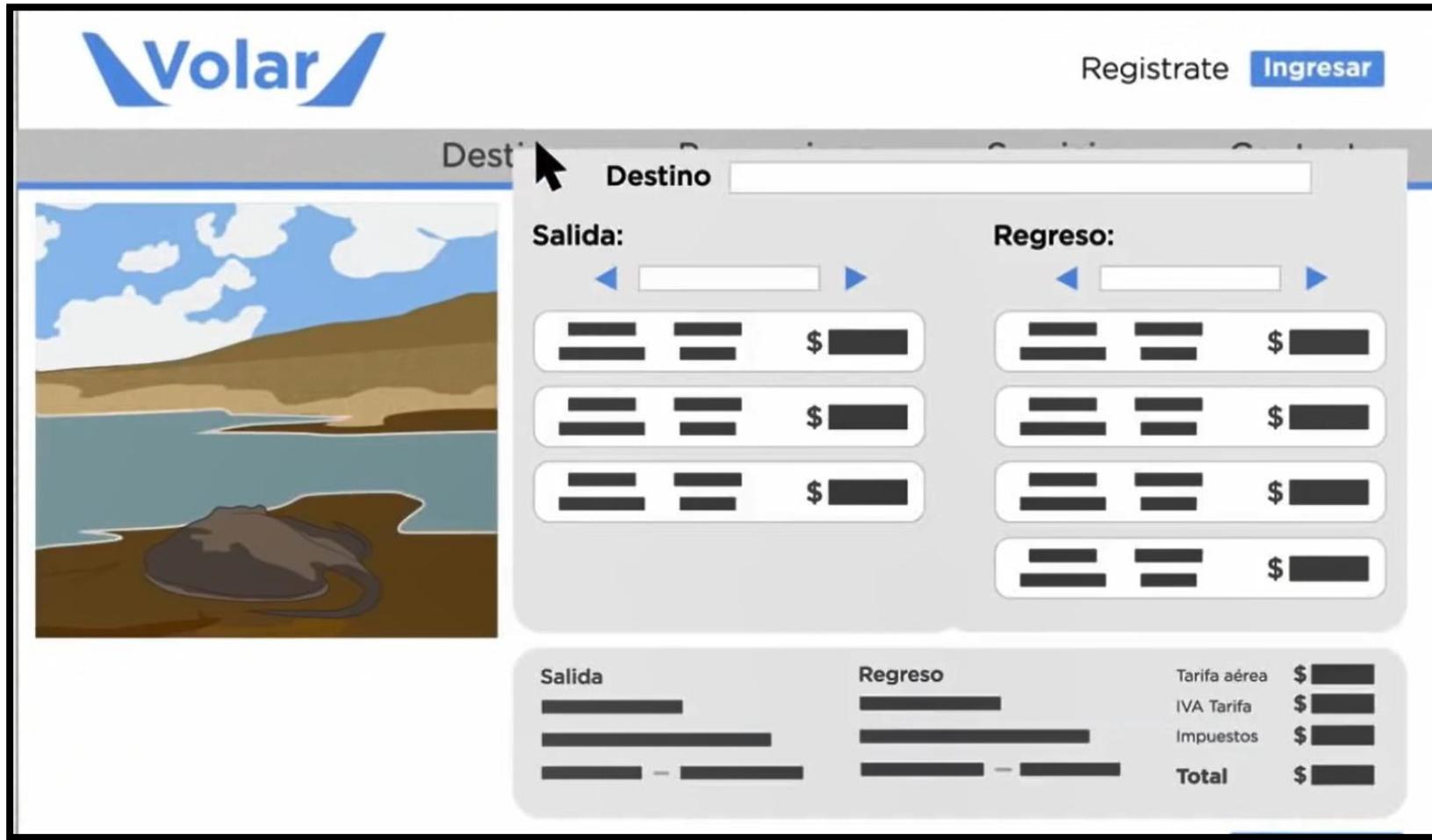


- El **back-end** es el encargado de procesar la información, almacenarla y recuperarla correctamente, dando un valor añadido a los datos recogidos por el front-end

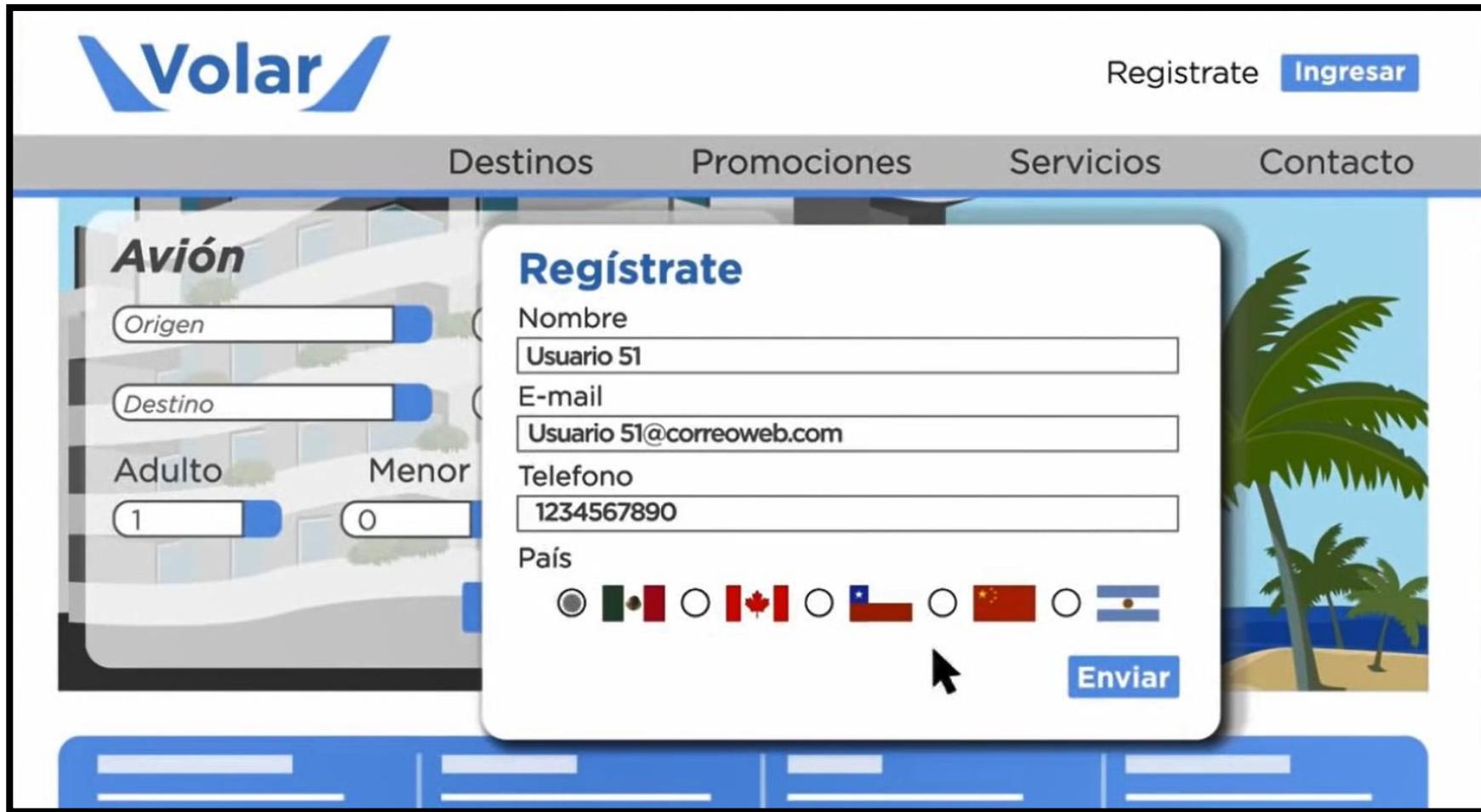
Front-end VS Back-end



Front-end VS Back-end



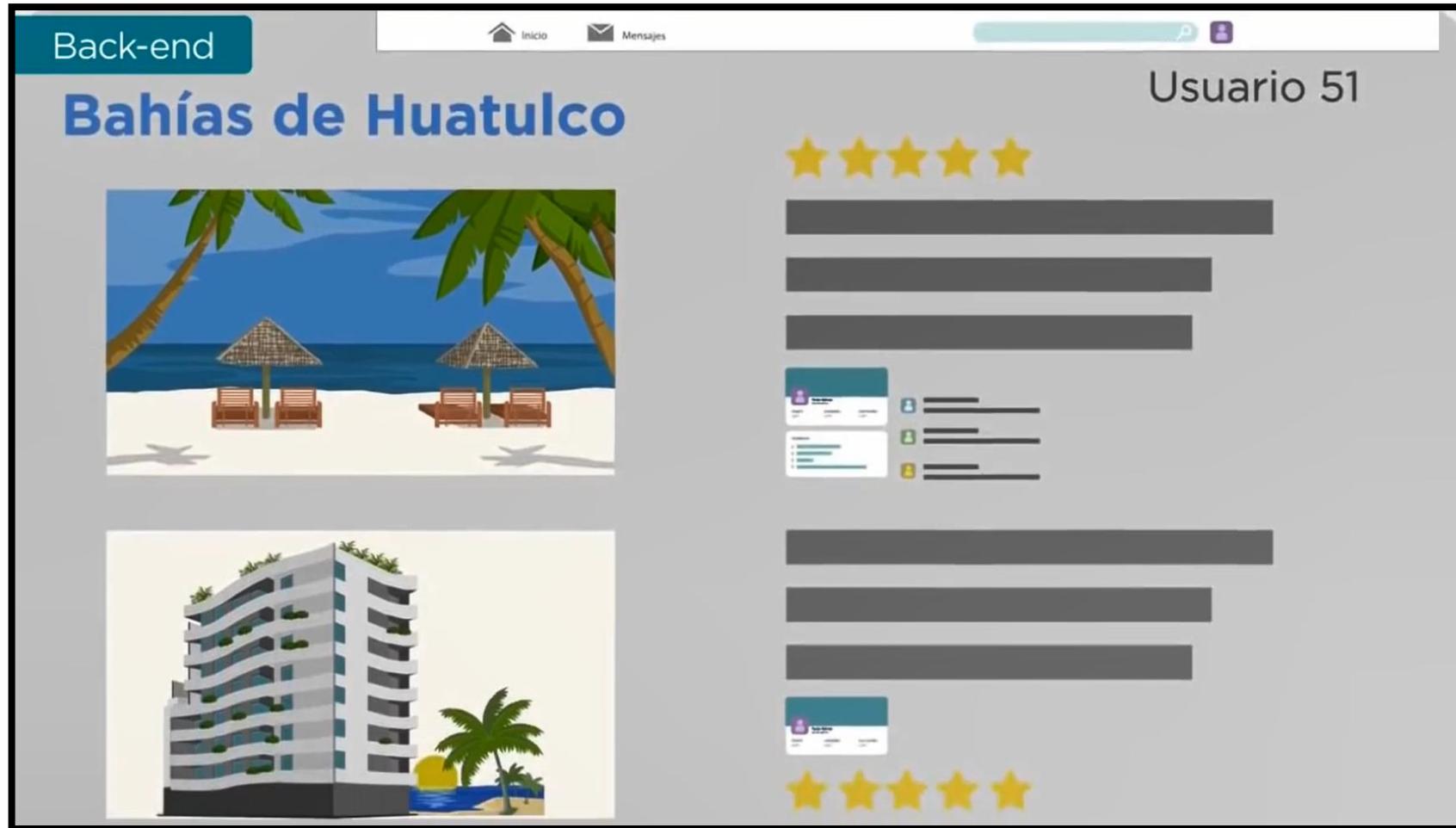
Front-end VS Back-end



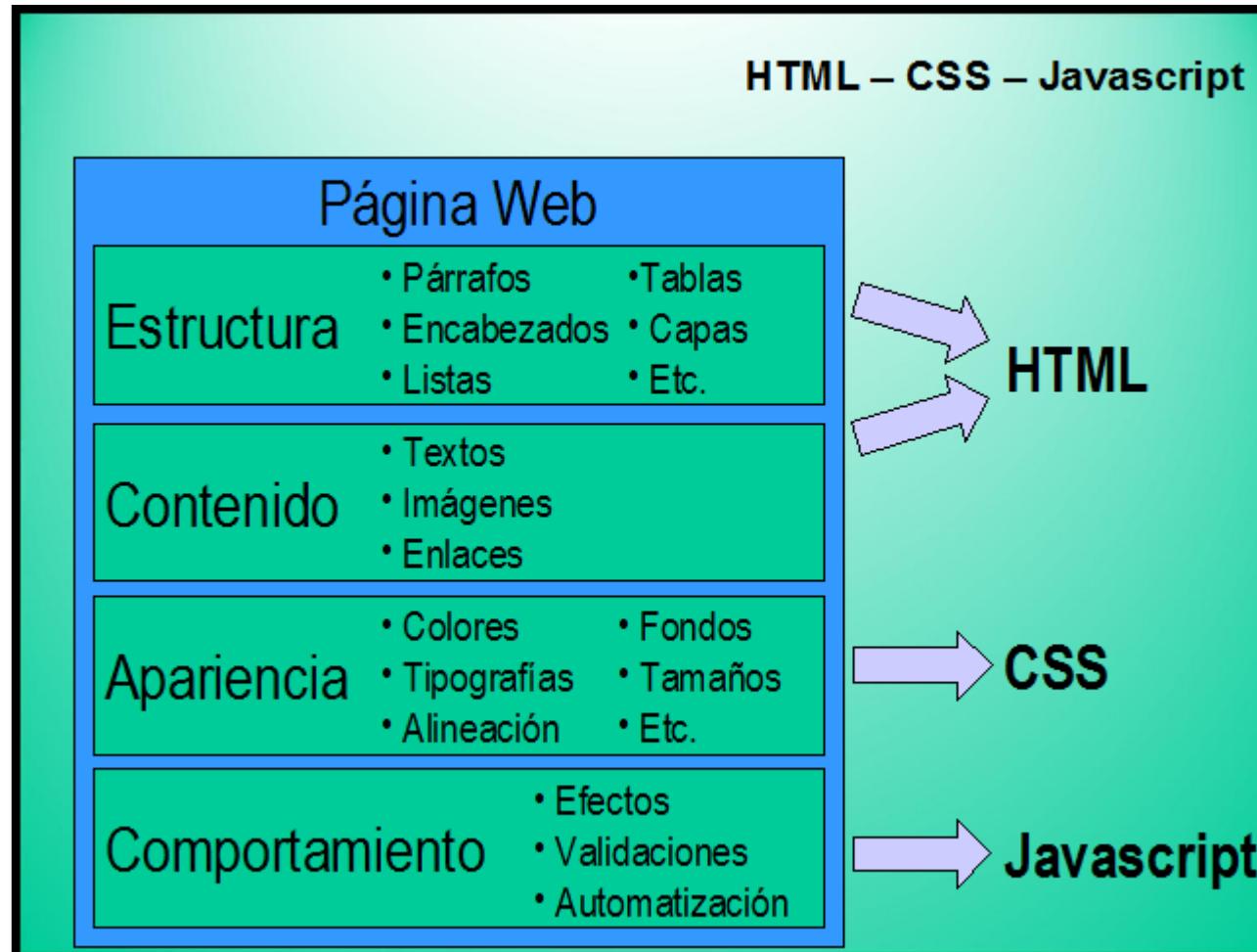
Front-end VS Back-end

1028	Secretaría de Contraloría y Desarrollo Administrativo	Gran usuario	Álvaro Obregón	Distrito Federal	Ciudad de México	1001	9	1001	22	10	7	Urbano	1
1048	Las Águilas	Gran usuario	Álvaro Obregón	Distrito Federal	Ciudad de México	1001	9	1001	22	10	8	Urbano	1
1049	Tlacopac	Pueblo	Álvaro Obregón	Distrito Federal	Ciudad de México	1001	9	1001	9	10	10	Urbano	1
1050	Ex-Hacienda de Guadalupe Chimalistac	Colonia	Álvaro Obregón	Distrito Federal	Ciudad de México	1001	9	1001	9	10	9	Urbano	1
1060	San Ángel Inn	Pueblo	Álvaro Obregón	Distrito Federal	Ciudad de México	1001	9	1001	28	10	9	Urbano	1
1060	Altavista	Colonia	Álvaro Obregón	Distrito Federal	Ciudad de México	1001	9	1001	9	10	12	Urbano	1
1070	Chimalistac	Unidad habitacional	Álvaro Obregón	Distrito Federal	Ciudad de México	1001	9	1001	31	10	13	Urbano	1
1080	Progreso Tizapan	Pueblo	Álvaro Obregón	Distrito Federal	Ciudad de México	1001	9	1001	28	10	14	Urbano	1
1089	Ermita Tizapan	Colonia	Álvaro Obregón	Distrito Federal	Ciudad de México	1001	9	1001	9	10	16	Urbano	1
1090	Tizapan	Colonia	Álvaro Obregón	Distrito Federal	Ciudad de México	1001	9	1001	9	10	18	Urbano	1
1090	La Otra Banda	Barrio	Álvaro Obregón	Distrito Federal	Ciudad de México	1001	9	1001	2	10	17	Urbano	1
1090	Loreto	Barrio	Álvaro Obregón	Distrito Federal	Ciudad de México	1001	9	1001	2	10	25	Urbano	1
1100	1	Unidad habitacional	Álvaro Obregón	Distrito Federal	Ciudad de México	1131	9	1131	31	10	26	Urbano	1
1100	1	Colonia	Álvaro Obregón	Distrito Federal	Ciudad de México	1131	9	1131	9	10	29	Urbano	1
1100	1	Colonia	Álvaro Obregón	Distrito Federal	Ciudad de México	1131	9	1131	9	10	31	Urbano	1
1100	1	Colonia	Álvaro Obregón	Distrito Federal	Ciudad de México	1131	9	1131	9	10	33	Urbano	1
1100	1	Colonia	Álvaro Obregón	Distrito Federal	Ciudad de México	1131	9	1131	9	10	37	Urbano	1
1100	1	Colonia	Álvaro Obregón	Distrito Federal	Ciudad de México	1131	9	1131	9	10	36	Urbano	1
1100	1	Colonia	Álvaro Obregón	Distrito Federal	Ciudad de México	1131	9	1131	9	10	39	Urbano	1
1100	1	Colonia	Álvaro Obregón	Distrito Federal	Ciudad de México	1131	9	1131	9	10	34	Urbano	1
1100	1	Colonia	Álvaro Obregón	Distrito Federal	Ciudad de México	1131	9	1131	22	10	41	Urbano	1
1100	1	Unidad habitacional	Álvaro Obregón	Distrito Federal	Ciudad de México	1131	9	1131	31	10	50	Urbano	1
1100	1	Colonia	Álvaro Obregón	Distrito Federal	Ciudad de México	1131	9	1131	9	10	42	Urbano	1
1100	1	Colonia	Álvaro Obregón	Distrito Federal	Ciudad de México	1131	9	1131	9	10	46	Urbano	1
1100	1	Colonia	Álvaro Obregón	Distrito Federal	Ciudad de México	1131	9	1131	9	10	47	Urbano	1
1100	1	Colonia	Álvaro Obregón	Distrito Federal	Ciudad de México	1131	9	1131	9	10	44	Urbano	1
1100	1	Colonia	Álvaro Obregón	Distrito Federal	Ciudad de México	1131	9	1131	9	10	43	Urbano	1
1100	1	Unidad habitacional	Álvaro Obregón	Distrito Federal	Ciudad de México	1131	9	1131	31	10	54	Urbano	1
1100	1	Colonia	Álvaro Obregón	Distrito Federal	Ciudad de México	1131	9	1131	9	10	52	Urbano	1
1100	1	Colonia	Álvaro Obregón	Distrito Federal	Ciudad de México	1131	9	1131	9	10	56	Urbano	1
1100	1	Colonia	Álvaro Obregón	Distrito Federal	Ciudad de México	1131	9	1131	9	10	53	Urbano	1
1100	1	Colonia	Álvaro Obregón	Distrito Federal	Ciudad de México	1131	9	1131	9	10	58	Urbano	1
1100	1	Colonia	Álvaro Obregón	Distrito Federal	Ciudad de México	1131	9	1131	9	10	59	Urbano	1
1100	1	Colonia	Álvaro Obregón	Distrito Federal	Ciudad de México	1131	9	1131	9	10	60	Urbano	1
1100	1	Colonia	Álvaro Obregón	Distrito Federal	Ciudad de México	1131	9	1131	9	10	64	Urbano	1
1100	1	Colonia	Álvaro Obregón	Distrito Federal	Ciudad de México	1131	9	1131	9	10	63	Urbano	1
1100	1	Colonia	Álvaro Obregón	Distrito Federal	Ciudad de México	1131	9	1131	22	10	65	Urbano	1
1115	Delegación Cuauhtémoc	Gran usuario	Álvaro Obregón	Distrito Federal	Ciudad de México	1131	9	1131	22	10	66	Urbano	1
1116	Maria G. E.	Colonia	Álvaro Obregón	Distrito Federal	Ciudad de México	1131	9	1131	9	10	68	Urbano	1
1116	La Victoria	Colonia	Álvaro Obregón	Distrito Federal	Ciudad de México	1131	9	1131	9	10	69	Urbano	1
1116	Bosque	Colonia	Álvaro Obregón	Distrito Federal	Ciudad de México	1131	9	1131	9	10	66	Urbano	1
1116	Isidro Fabián	Colonia	Álvaro Obregón	Distrito Federal	Ciudad de México	1131	9	1131	9	10	67	Urbano	1
1117	Santa Fe I	Unidad habitacional	Álvaro Obregón	Distrito Federal	Ciudad de México	1131	9	1131	31	10	75	Urbano	1

Front-end VS Back-end



- El front-end se divide en diversos lenguajes cada uno con sus funciones.



Renacimiento de JavaScript con la UX

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



- ❑ Vamos a usar **JavaScript** porque es el lenguaje de script más utilizado en el lado del cliente, y está soportado mayoritariamente por todas las plataformas.
- ❑ Brendan Eich que trabajaba en el navegador Netscape desarrollo un lenguaje de programación llamado **Mocha**, que posteriormente se denominó **LiveScript**.
- ❑ Coincidiendo con la popularidad del lenguaje Java acabo denominándose **JavaScript**.
- ❑ ¿Entonces qué es **ECMA Script**?

- Debido a que **Microsoft (JScript, VBScript)** y otros sacaron sus propias versiones, los autores **originales** desarrollaron un **estándar** para la **ECMA** que es el adoptado en la actualidad por todos los navegadores.
- De hecho **TypeScript** que ha sido desarrollado también por Microsoft es compatible con **JavaScript**, porque 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**

Sintaxis básica, variables y operadores



- La sintaxis de JavaScript en la parte básica 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 y minúsculas.
- Se pueden incluir comentarios con // y /* */
- Existen palabras reservadas del lenguaje.

- ❑ No es necesario establecer el tipo de datos en la declaración de las variables que se ajusta automáticamente.
- ❑ Los tipos de datos básicos existentes son:
 - ✓ Numérico, incluyendo los números decimales.
 - ✓ Lógico o booleano, true o false.
 - ✓ Tipo cadena o string, que almacena texto.
- ❑ El resto son de tipo objeto.

- El nombre de la variable solo puede contener letras, números y los símbolos “_” y “\$”, además de no empezar por número. Para declarar variables se usa la palabra **let**.
- Existe otra forma de declarar variables con la palabra **var**, pero es desaconsejado su uso porque su alcance es confuso.
- Lo veremos más adelante con los bucles y funciones

```
Let primera;
Let puntuacion,record,jugador;
Let correcta="ok";
Let _correcta2=8;
Let $correctisima=4.78;
Let correcta_de_mas=true;
Let 1incorrecta="blue";
Let puntuacion=0,record=5000,jugador="Juan";
```

- Si probamos el código anterior vemos que primera no tiene valor y muestra undefined.
- Existen operadores aritméticos: +,-,*,/,%, ++,--,+=,-=, ...
- De comparación: ==,!=,>,<,>=,<=, ===, !==
- Lógicos: !(NOT), &&(AND) y || (OR)

```
Let cantidad,precio,total;
cantidad=100;
precio=5;
total=cantidad*precio;
```

- 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’

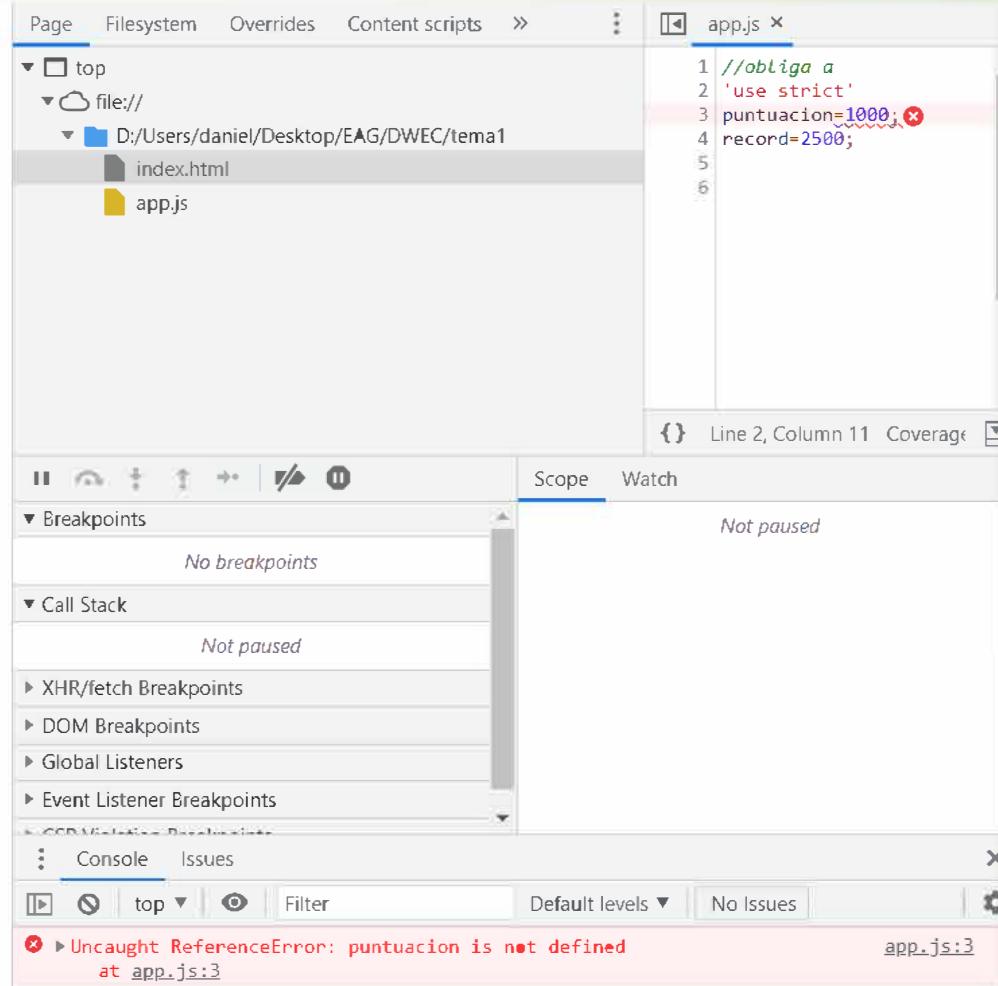
//si no ponemos let automaticamente es var
//el scope o ambito es global y no lo esta controlado
puntuacion=1000;
record=2500;



//obliga a establecer con let o var
'use strict'
puntuacion=1000;
record=2500;



'use strict'
let puntuacion=1000;
let record=2500;



The screenshot shows a browser developer tools debugger interface. On the left, the file structure is visible with 'app.js' selected. The code editor on the right shows the following code with line 3 underlined in red:

```
1 //obliga a
2 'use strict'
3 puntuacion=1000; ~~~~~ ✖
4 record=2500;
5
6
```

The status bar at the bottom indicates 'Line 2, Column 11 Coverage'.

The debugger sidebar on the left includes sections for Breakpoints, Call Stack, XHR/fetch Breakpoints, DOM Breakpoints, Global Listeners, and Event Listener Breakpoints. The 'Console' tab at the bottom is active, showing the error message:

```
Uncaught ReferenceError: puntuacion is not defined
    at app.js:3
```

The 'Issues' tab is also present in the bottom right corner.

Más ejemplos de operadores matemáticos

```
let num1=4, num2=5.4;  
let suma=num1+num2;  
let aumenta_cinco=num2+5;  
let quita=num1-3;  
let escrito="3";  
let resultado=num1+escrito;
```

```
Let base,exponente;  
base=2;  
exponente=5;  
Let potencia=base**exponente;  
Let resto=base%exponente;
```

- Otros operadores de asignación que incluyen operandos.

Operador	Utilización	Expresión equivalente
<code>+=</code>	<code>op1 += op2</code>	<code>op1 = op1 + op2</code>
<code>-=</code>	<code>op1 -= op2</code>	<code>op1 = op1 - op2</code>
<code>*=</code>	<code>op1 *= op2</code>	<code>op1 = op1 * op2</code>
<code>/=</code>	<code>op1 /= op2</code>	<code>op1 = op1 / op2</code>
<code>%=</code>	<code>op1 %= op2</code>	<code>op1 = op1 % op2</code>

¿Cuál es la diferencia entre == y ===?

- La principal diferencia es que === hace comparación de valores y de tipos, dando lugar a que si usamos == obtenemos lo siguiente:

```
0 == "0"      // true
132 == "132" // true
false == 0    // true
false == "0" // true
true == 1     // true
true == "1"   // true
"1,2" == [1,2] //true
```

- Tiene su lógica pero esconden un peligro

¿Cuál es la diferencia entre == y ===?

- Podría parecer que el motivo es perdida de eficiencia, pero realmente deriva a un problema de semántica muy grave:

```
null == undefined // true
false == "" // true
false == [] // true
false == [0] // true
false == [[[]]] // true
0 == "" // true
```

- Conclusión usar siempre === y !==, pero no os rialis de quienes no lo usan.

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

```
let A,B,C,D,E,F,G,H;  
A=5;  
B=3;  
(A>B)  
(A<B)  
(A>=B)  
(A<=B)  
(A==B)  
(A!=B)
```

```
4==4  
4>3  
x=5  
x<=7  
x<5  
edad>=18  
x==5  
jornada=="media"  
jornada!="completa"  
x%2==0
```

Operadores lógicos o booleanos

AND: sirve para **UNIR**

OR: sirve para **ELEGIR**

NOT: sirve para **EXCLUIR**

¿Cual es el valor de cada variable al ejecutar el código?

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

Tipo booleano en el mundo real

```
continuar=true;
!continuar

x>=1 && x<=10
x>45 && sexo==> "masculino"
x>45 || sexo==> "masculino"
letra==> "A" || letra==> "E" || letra==> "I" || letra==> "O" || letra==> "U"
continuar && credito>0

encontrado=false;
fin=true;
encontrado || fin
```

Tipo booleano en el mundo real

```
alert(nombre==="Miguel");
alert(frase==="Eureka" );
alert(!aprobado);
alert(!fin);
alert(mes>=1 && mes<=12);
alert(ahorros>precio_coche || loteria==true );
alert(bachillerato==true || prueba_acceso==true);
alert(isNaN(34));
alert(isNaN("dani"));
alert(!isNaN(10));
```

Utilización de valores constantes

```
//Si necesitamos guardar un valor y no va a cambiar  
//es mas eficiente que una variable  
const num1=4,num2=5.4;  
const rojo="#FF0000";  
const gravedad=9.8;  
//darle significado a números  
const pedido_recibido=0;  
const pedido_procesado=1;  
const pedido_enviado=2;  
const arriba="w";  
const abajo="s";  
...
```

String y Array



Operaciones básicas de string

```
Let texto="Un texto con 'comillas' dentro";
Let otro_texto='Otro texto tambien con "comillas" dentro';
Let numero_letras=text.length;

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

```
type(prueba1); //Devuelve el tipo
type(prueba2); //de una expresion o variable
type(prueba3);
```

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
```

Template o plantilla

- ❑ Si usamos la comilla backticks o acento grave permite escribir en varia 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 templateates mas sencillo
`Quince es ${a + b} y no ${2 * a + b}.`;

//Si queremos salto de linea
'Quince es ' + (a + b) + ' y\nno ' + (2 * a + b) + '.';
``Quince es ${a + b} y
no ${2 * a + b}.`;
```

console.log

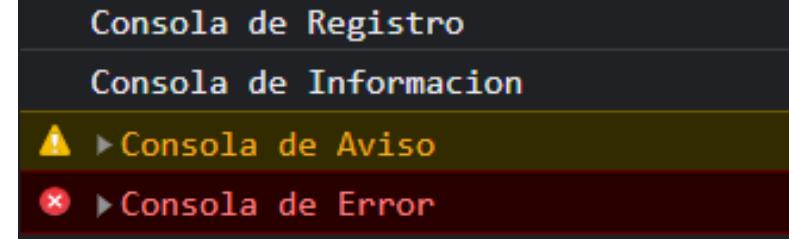
- Para escribir algo en la consola, existe la función `console.log()` / `console.table()` / etc...
- Los usuarios regulares no ven este output, ya que está en la consola. Para verlo, debemos abrir la consola de desarrolladores y presionar la tecla Esc y en otro tab: se abrirá la consola debajo.

```
let x = 1;  
console.log("x:", x);
```

console.log y otras variantes

```
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}.`);
```

```
console.log('Consola de Registro');
console.info('Consola de Informacion');
console.debug('Console de Depuracion');
console.warn('Consola de Aviso');
console.error('Consola de Error');
```



Falta un mensaje ¿por qué?

alert

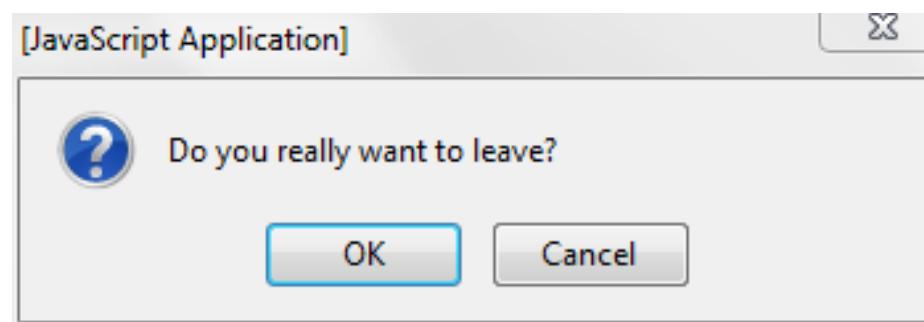
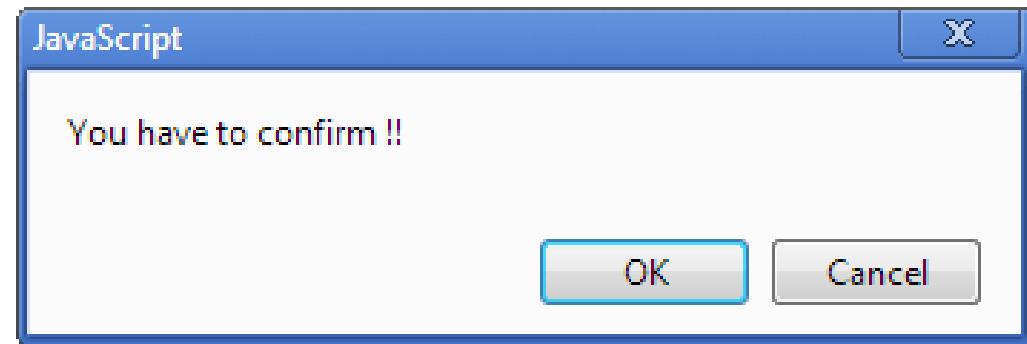
- Muestra una ventana con el mensaje que le hayamos indicado y es una forma básica de comunicación.



- Ya hemos visto alert para mostrar información, también existen mas métodos para pedir y mostrar información.
- La instrucción **prompt** además de mostrar un mensaje permite recoger información del usuario



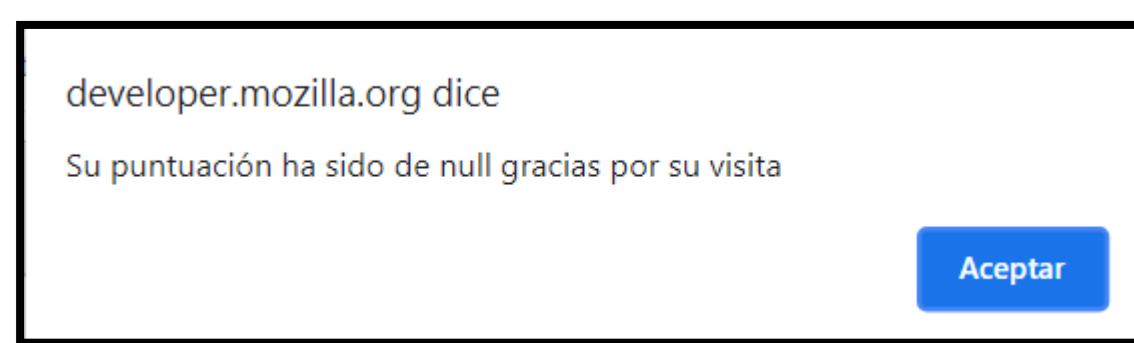
- La orden **confirm** es parecida a la anterior pero ofreciendo la respuesta si o no.



Entrada y salida de datos

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

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



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

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

Entrada y salida con JavaScript

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

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

Los templates mejoran el código

```
let mensaje=prompt("Escriba una frase:");
document.write(`<h1>${mensaje}</h1>`);
let nivel=prompt("¿Que nivel de encabezado desea?(1-6)");
document.write(`<h${nivel}>Bienvenido a mi pagina web</h${nivel}>`);
```

Esta línea no daría el resultado esperado
Sugerencia: probar parseInt

```
document.write(` ${nivel+1} `);
```

Depuración básica usando

```
alert(document);  
alert(Math)
```

Esta página dice

[object HTMLDocument]

Aceptar

Esta página dice

[object Math]

Aceptar

```
console.log(document)  
console.log(Math)  
//pasa lo mismo con array  
//o cualquier objeto
```

```
▼#document  
  <!DOCTYPE html>  
  <html>  
    ▶<head>...</head>  
    ▶<body>...</body>  
  </html>  
  
▼Math {abs: f, acos: f, acosh:  
  E: 2.718281828459045  
  LN2: 0.6931471805599453  
  LN10: 2.302585092994046  
  LOG2E: 1.4426950408889634  
  LOG10E: 0.4342944819032518  
  PI: 3.141592653589793  
  SQRT1_2: 0.7071067811865476  
  SQRT2: 1.4142135623730951  
  ▶ abs: f abs()  
  ▶ acos: f acos()  
  ▶ acosh: f acosh()  
  ▶ asin: f asin()  
  ▶ asinh: f asinh()  
  ▶ atan: f atan()  
  ▶ atan2: f atan2()
```

- 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 seria la siguiente

Posición	0	1	2	3
Valor	25	36	21	58

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

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

- Un array en JavaScript no tiene un tamaño predefinido, con lo que puede tener tantos valores como sea necesario.
- Al final del tema 1 y en el tema 2 veremos toda versatilidad de los arrays en Javascript.

Operaciones básicas de un Array

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

//un string tambien 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]="León";
coche[1]="Seat";
coche[2]=5;
coche[3]="V6";

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

Los templates funcionan muy bien con los arrays

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

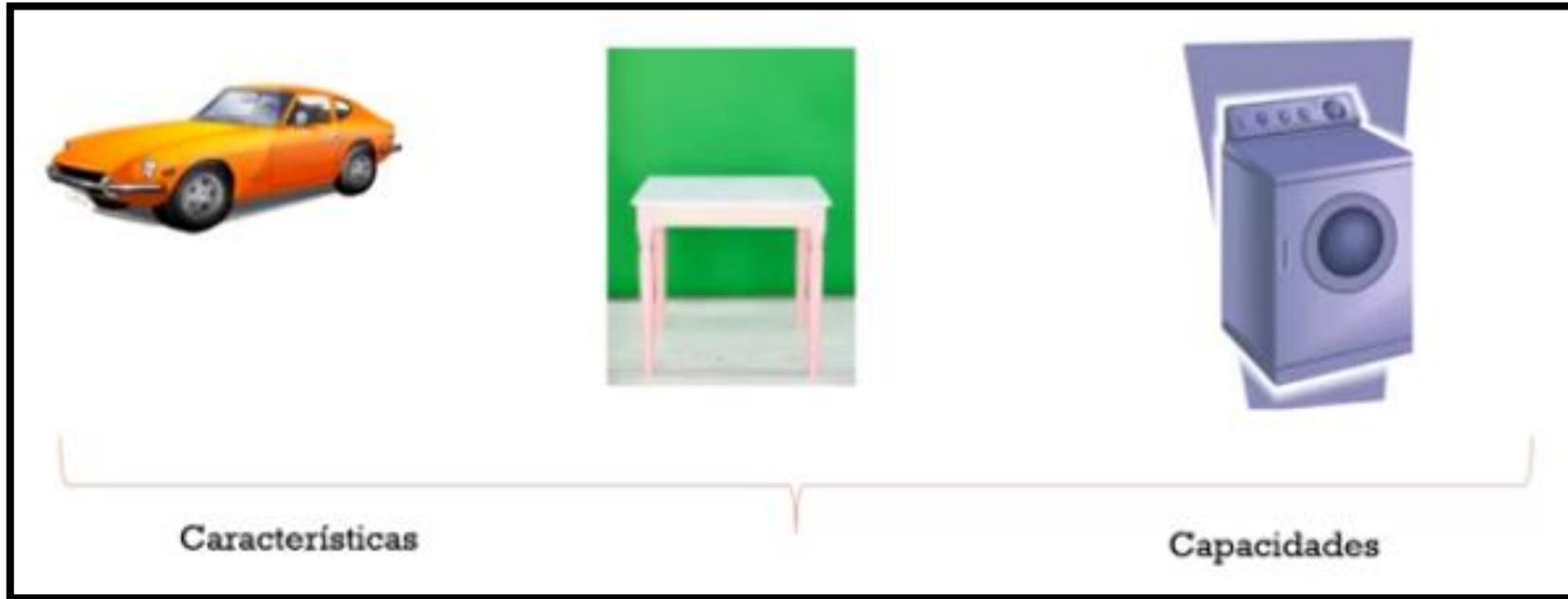
Dando lugar a crear un código mucho más claro y sencillo.

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 ordenes perteneciente al objeto **window** (se puede omitir).
- Por supuesto **document.write** que indica que el método **write** 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?



¿Qué es la POO?

- Propiedades (Cómo es):

- Tiene puertas
- Tiene ruedas
- Tiene ventanas
- Tiene motor
- Etc.



Objeto=Coche

- Métodos (¿Qué puede hacer?):

- Arrancar
- Acelerar
- Frenar
- Girar
- Etc.



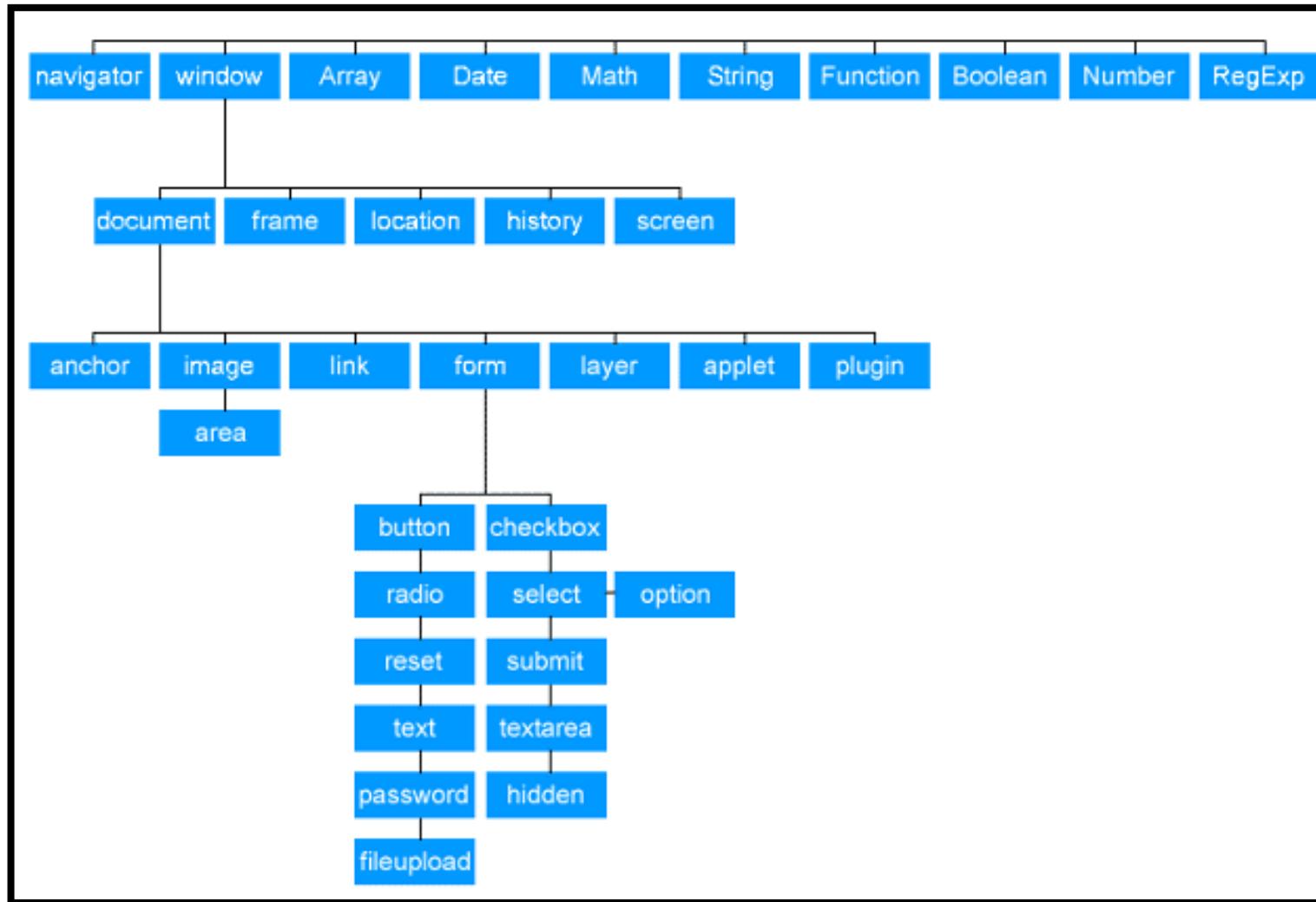
Instancia de coche.
Coche 1



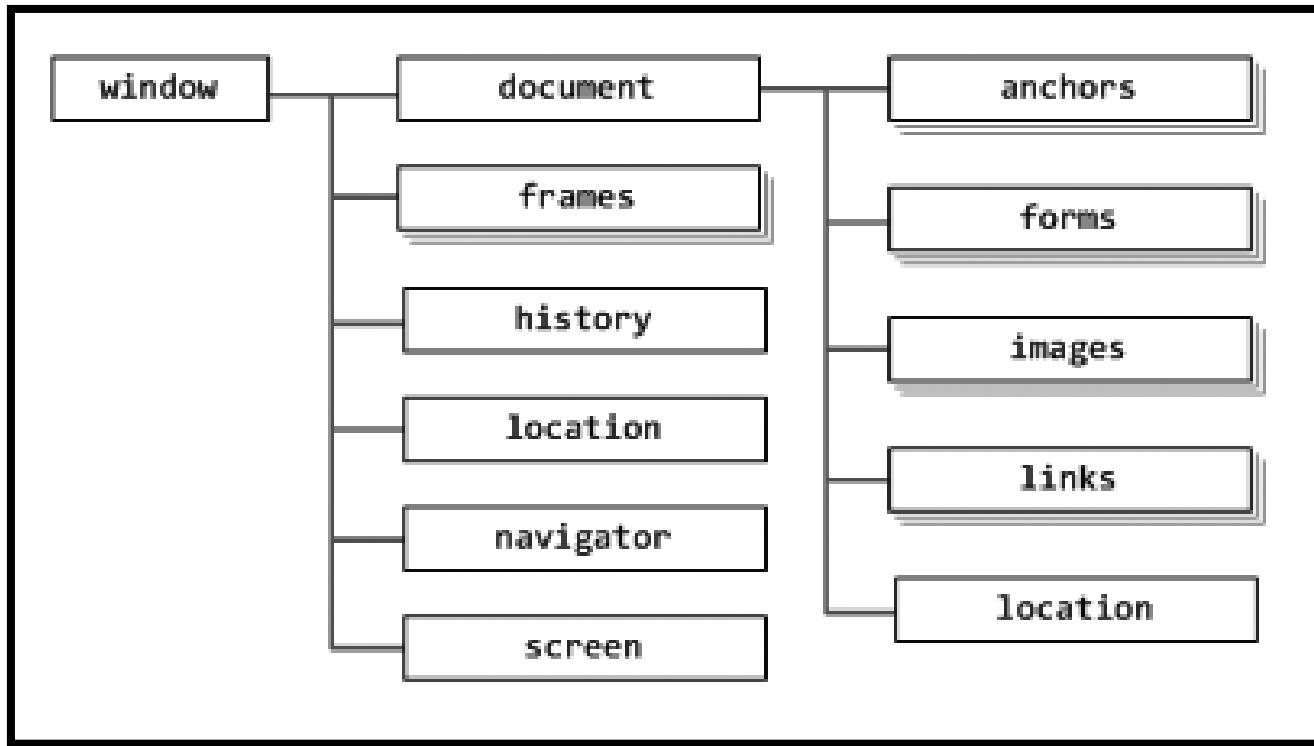
Instancia de coche.
Coche 2

- En la programación orientada a objetos los elementos de un programa van a funcionar conceptualmente igual.

Jerarquía de objetos JavaScript



window y document



Operador . (punto)

```
//Características
document.body
document.title
document.URL
document.links //array de enlaces
document.images //array de imágenes
document.cookie

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

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

Objeto string

```
let nombre="Daniel,Garcia";
//Caracteristicas
nombre.length;
//Capacidades
"Nombre: " + nombre
nombre.toUpperCase(); //Convierte a mayusculas
nombre.toLowerCase(); //Convierte a minusculas
nombre.indexOf("n"); //Posicion de La Letra n
nombre.split(","); //Separa string por ,
```

Más métodos clase String

```
'Widget con id'.includes('Widget'); // true
'Hola'.includes('Adiós'); // false

let str = "stringify";
// esto es lo mismo para substring
str.substring(2, 6); // "ring"
str.substring(6, 2); // "ring"

'aabbcc'.replaceAll('b', '.'); // 'aa..cc'
"xxx".replaceAll('_', '_'); //x_x_x

str=" I have outer spaces      ";
cleanStr=str.trim(); // "I have outer spaces"

(8).toString(2)
(25).toString(2)
(235).toString(2)
//Conversion a binario
"1000"
"11001"
"11101011"
```

Clase array

```
let colores=["Verde","Azul","Rojo"];
//Características
colores.length;
//Capacidades
colores.indexOf("Rojo");
colores.reverse();
colores.sort();
colores.push("Amarillo"); //Añade al final
colores.unshift("Amarillo"); //Añade al comienzo
colores.pop(); //Saca del final
colores.unshift() //Saca del comienzo
```

Controlar tamaño del array

```
let jsLibraries = ['react', 'redux', 'vue', 'D3', 'Chart']
jsLibraries.length = jsLibraries.length - 1 //Borra el ultimo
jsLibraries.length = 0 // Lo vacia
jsLibraries.length = jsLibraries.length - 3 // ["react", "redux"]
//Deja hueco
delete jsLibraries[2] // ['react', 'redux', vacío, 'D3', 'Chart']
splice(pos, cantidad) //borra cantidad desde posicion
jsLibraries.splice(0, 1) // ['redux', 'vue', 'D3', 'Chart']
jsLibraries.splice(1, 3) //["react", "Chart"]
//Devuelve los que borra 'redux', vacío, 'D3'
//----
//permite añadir a la vez que borra
let fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2, 1, "Lemon", "Kiwi");
```

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 únicamente vamos a usar sus métodos
//y no reasignamos a esas variables 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
```

Objetos predefinidos JavaScript

- Date.
- window.
- history.
- navigator.
- screen.
- cookie.
- etc...

Más información en:

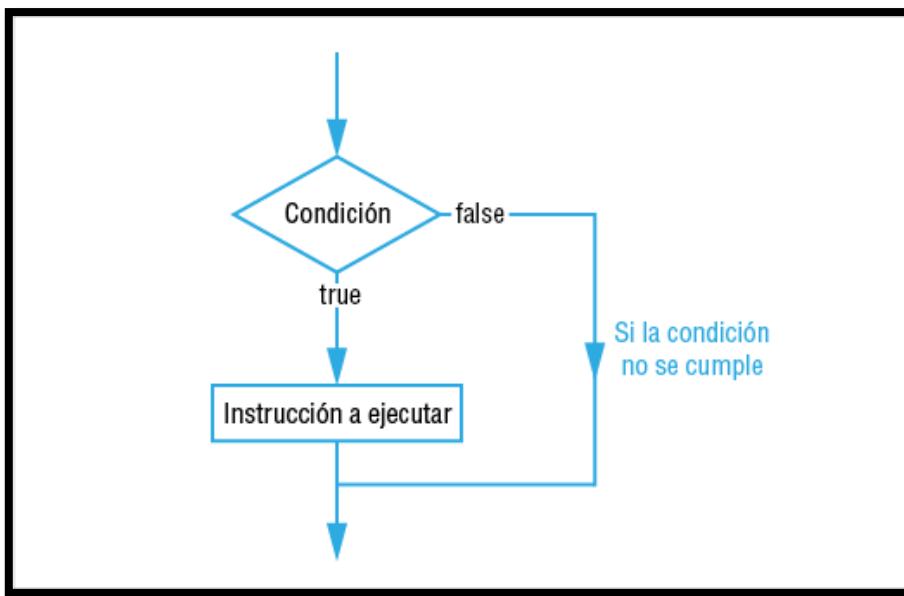
<https://www.arkaitzgarro.com/javascript/capitulo-14.html>

if - elseif - else y switch



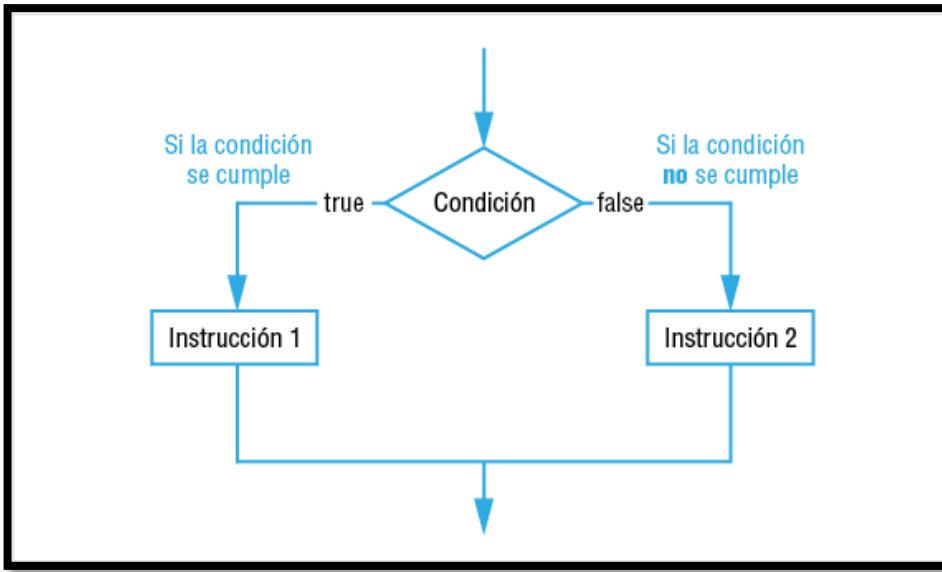
- Las estructuras condicionales realizan unas acciones u otras dependiendo del **estado de las variables**.
- Vamos a ver las estructuras **if-elseif-else** y **switch-case**
- La principal diferencia es que la estructura **if-else** comprueba si una **condición de cualquier tipo** es true o false.
- Mientras que la estructura **switch** solo permite definir **condiciones de igualdad** con varios valores concretos.

Esquemas if-else



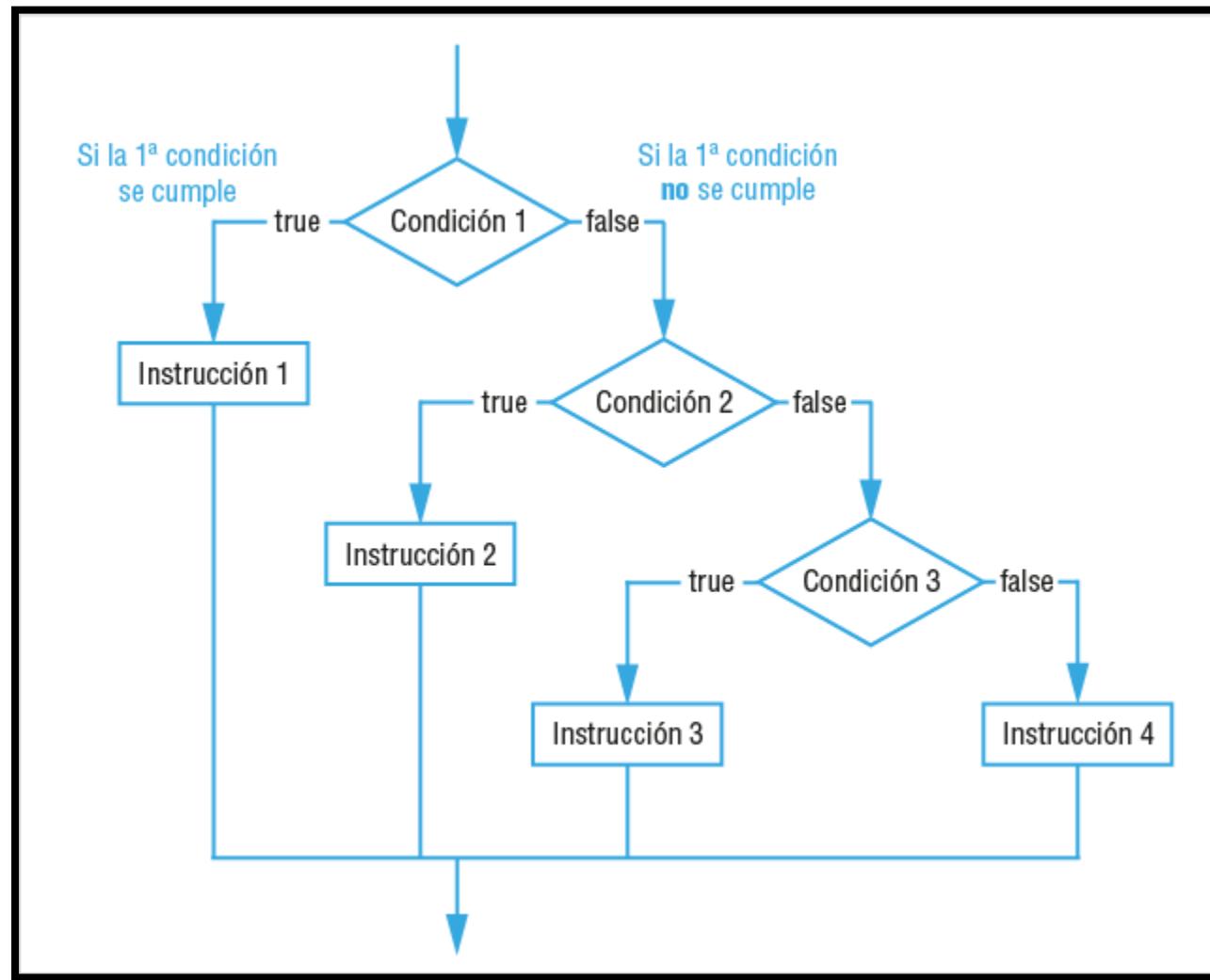
```
let miEdad=....  
if(miEdad>30){  
    console.log("Ya eres una persona adulta");  
}
```

Esquemas if-else



```
let semáforo=...  
if(semáforo==="verde")  
{  
    console.log("Puede pasar");  
}else{  
    console.log("Detengase");  
}
```

Esquemas if-else



- Una forma de hacer una comprobación más completa seria la siguiente.

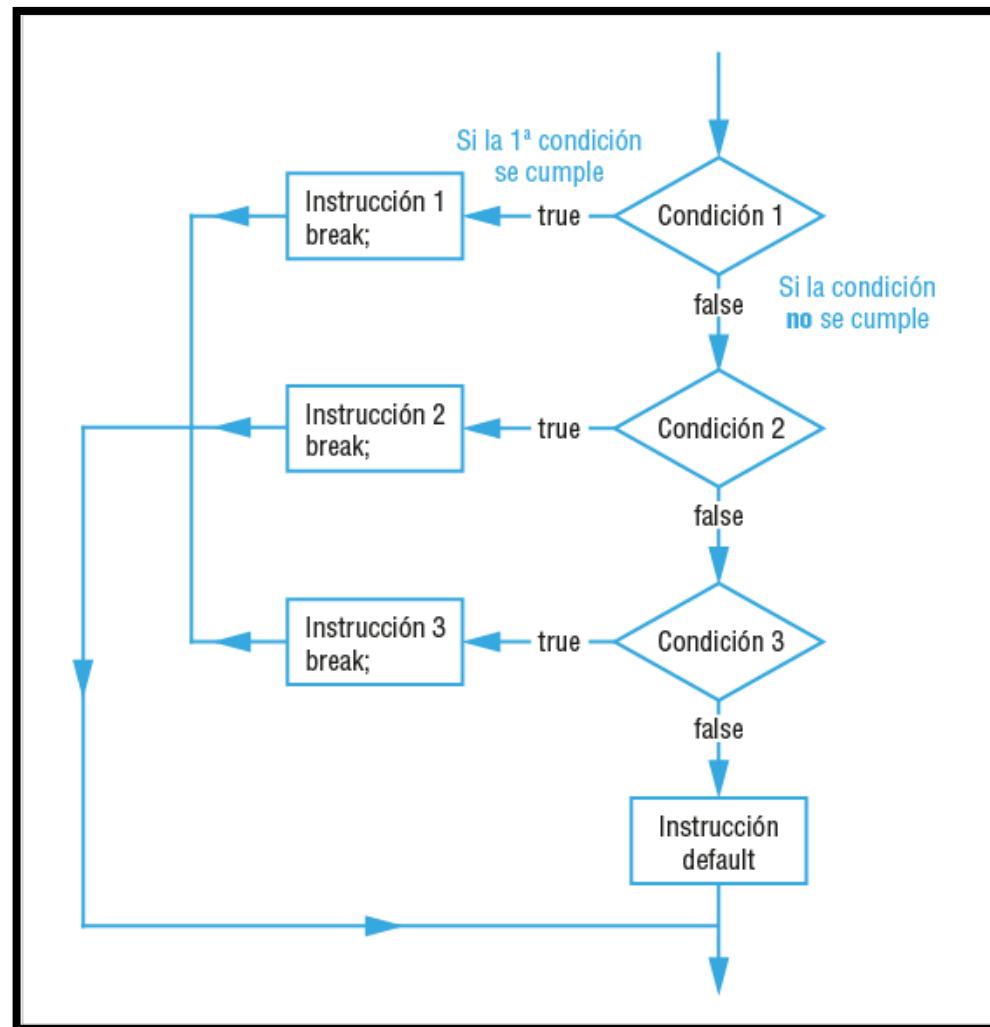
```
if(nota>=5)
{
    if(nota<7)
    {
        console.log("Aprobado</h2>");
    }else{
        console.log("Notable</h1>");
    }
}else{
    console.log("Suspensos</h3>");
}
```

- De manera equivalente se puede usar la forma if-elseif

```
if(nota>=7)
{
    console.log("Notable</h1>");
}else if(nota>=5){
    console.log("Aprobado</h2>");
}else{
    console.log("Suspensos</h3>");
}
```

```
let resolucion = ...
if(resolucion<400)
{
    console.log("Resolucion movil");
}else if(resolucion<800){
    console.log("Resolucion tabler");
}else if(resolucion<1280){
    console.log("Resolucion tablet");
}else{
    console.log("Resolucion monitor");
}
```

Esquema switch



Esquemas switch

```
let dia=prompt("Introduce el dia de la semana");
switch(dia) {
    case 1:
        alert("Lunes");
    break;
    case 2:
        alert("Martes");
    break;
    ...
    default:
        alert("No es un dia de la semana");
}
```

Comparación if y switch

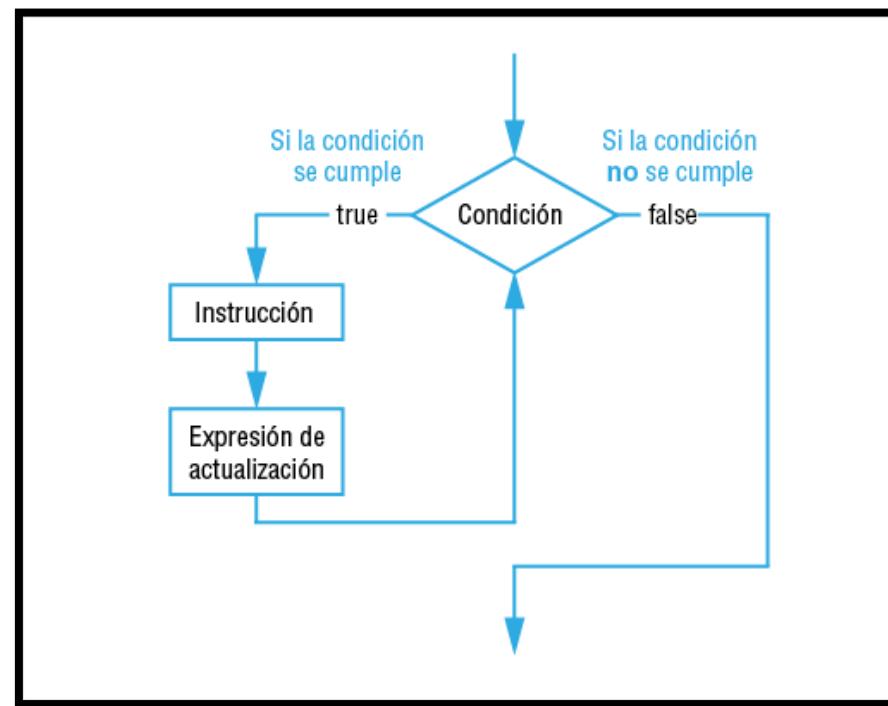
```
let materia=...
if(materia=="matematicas")
{
    console.log("Va de numeros");
}else if(materia=="lengua"){
    console.log("Va de letras");
}else if(materia=="ingles"){
    console.log("Va de hablar");
}else if(materia=="ciencias"){
    console.log("Va de investigar");
}else{
    console.log("No sé de que va");
}
```

```
let materia=...
switch(materia)
{
    case "matematicas":
        console.log("Va de numeros");
        break;
    case "lengua":
        console.log("Va de letras");
        break;
    case "ingles":
        console.log("Va de hablar");
        break;
    case "ciencias":
        console.log("Va de investigar");
        break;
    default:
        console.log("No sé de que va");
}
```

while, do while y for



- Con el bucle for, podemos repetir una tarea un número fijo de veces.
- El bucle while permite crear bucles que se ejecutan cero o más veces de manera indefinida/ilimitada.



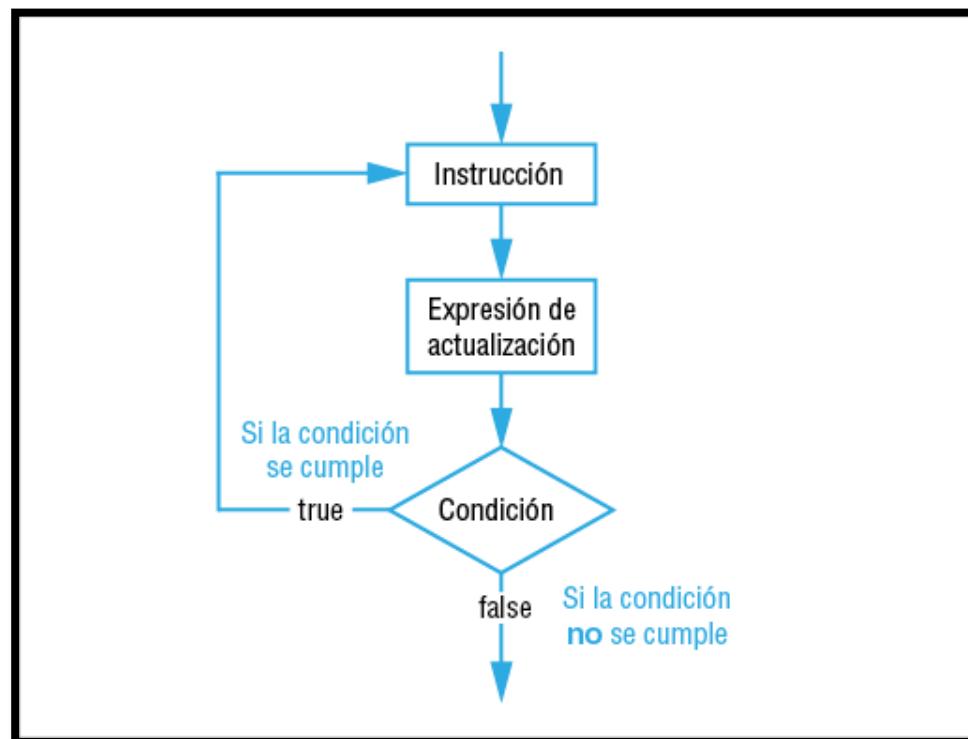
Ejemplos while

```
let i=0;
while(i<10)
{
    console.log("<br>El numero es " + i);
    i++;
}
```

```
let edad=prompt("Introduce tu edad");
while(edad<0 || isNaN(edad))
{
    edad=prompt("Introduce tu edad");
}
```

```
let secreta="eureka";
let intento="";
while(intento!==secreta)
{
    intento=prompt("¿Que palabra es?");
```

- El bucle do-while es una variante del bucle while que se ejecuta siempre al menos una vez.
- Permite escribir de manera más clara en ciertas situaciones



Ejemplos do-while

```
let i=1;
let respuesta;
do
{
    respuesta=confirm("¿Desea salir? Intento"+ i);
    i++;
}while (respuesta!==true);
```

```
let color;
do
{
    color=prompt("indica un color distinto de blanco");
}while(color==="white"|| color==="#FFF" || color==="#FFFFFF" );
```

- ❑ El bucle for permite ejecutar un bloque de código un número fijo y conocido de veces.
- ❑ Ese numero fijo de veces se establece mediante una condición de <= ó =>.

```
for (expresión inicial; condición; incremento)
{
    // Instrucciones a ejecutar dentro del bucle.
}
```

Bucle for

```
for(let veces=1;veces<=10;veces++){
    console.log(`El nombre numero ${veces} es ${nombre}`);
}
```

```
for(let i=1;i<=6;i++){
    document.write(`<h${i}>Cabecera ${i}</${i}h>`);
}
```

Array con for y sus variantes

```
let lenguajes=["HTML","CSS","JavaScript"];
//Solo para arrays
for(let i=0;i<lenguajes.length;i++)
{
    document.write(`<input type='button' value='${lenguajes[i]}'>`);
}

//funciona en array y objetos, no en colecciones
for(let posicion in lenguajes)
{
    document.write(`<input type='button' value='${lenguajes[posicion]}'>`);
}

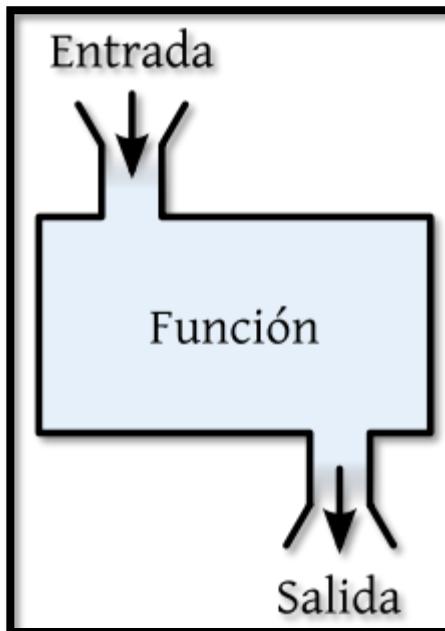
//funciona en arrays y colecciones, no en objetos
for(let elemento of lenguajes)
{
    document.write(`<input type='button' value='${elemento}'>`);
}
```

Modularización de código.

Las 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 en JavaScript

```
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);
```

Diseño modular de funciones

No incluir entrada salida en funciones

```
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
```

Diseño modular de funciones

```
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;  
}  
  
let paises=["España","Francia","Portugal","Italia"];  
let tipo= ...  
document.write(ejemploModular(paises,tipo));
```

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
```

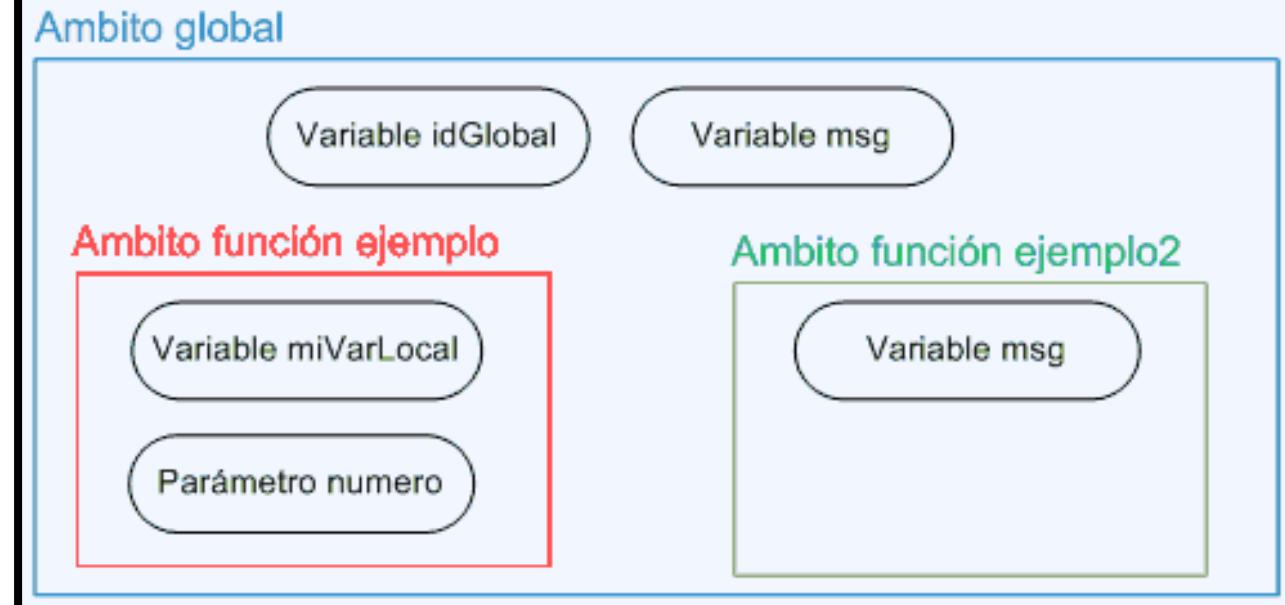
- El ámbito de una variable, es la zona del programa donde existe la variable y se puede operar con ella.
- **Variables locales:** Son variables como el contador de un bucle for o declaradas dentro de una función. No se pueden acceder desde fuera de su bloque.
- **Variables globales:** Son variables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

Ámbito variables

```
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

```
//Funcion anonima
function (mensaje){
    console.log("¡¡"+mensaje+"!!")
}
//Se pierde si no se guarda o usa en algo
-----
//Las funciones son un tipo de dato
//por lo tanto se pueden guardar en const o let
const exclamar=function (mensaje){
    console.log("¡¡"+mensaje+"!!")
}

//equivale a
function exclamar(mensaje){
    console.log("¡¡"+mensaje+"!!")
}

exclamar("Hola mundo");
```

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+" ! !")
});
```

Callback

```
//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.



- En JavaScript existe un tipo de datos especial para modelar valores compuestos con propiedades
- Se les llaman comúnmente **objetos**
- Se basa en usar llaves y el sistema clave-valor



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

Manejo básico de un JSON

```
//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 JSON

```
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 variables  
  fruta:cantidad,  
}
```

Más formas de crear objetos JSON

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

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
```

Podemos almacenar funciones como valores de propiedad

```
const john = {  
    name: "John",  
    sayHi: function() {  
        console.log("Hi buddy!");  
    }  
};  
  
john.sayHi(); // Hi buddy!
```

En los siguientes temas profundizaremos en la programación funcional

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 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 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
```

- ❑ ¿Entonces que 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 lenguaje de programación

XML

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <endereco>
3   <cep>31270901</cep>
4   <city>Belo Horizonte</city>
5   <neighborhood>Pampulha</neighborhood>
6   <service>correios</service>
7   <state>MG</state>
8   <street>Av. Presidente Antônio Carlos, 6627</street>
9 </endereco>
```

vs.

JSON

```
1 {
2   "endereco": {
3     "cep": "31270901",
4     "city": "Belo Horizonte",
5     "neighborhood": "Pampulha",
6     "service": "correios",
7     "state": "MG",
8     "street": "Av. Presidente Antônio Carlos, 6627"
9   }
10 }
```

- ❑ No deja de ser un **String** que contiene información estructura 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"  
    }  
]
```

- 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 intercambiar **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

- ✓ <http://www.rtve.es/api/noticias.json>
- ✓ <https://pokeapi.co/>
- ✓ <https://api.chucknorris.io/>

- ✓ <https://www.mocky.io/>
- ✓ <https://jsonplaceholder.typicode.com/>
- ✓ <https://randomuser.me/>
- ✓ <https://fakestoreapi.com/>

- ❑ 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 queharemos es usar ficheros locales que tiene los datos en formato JSON.



Otras variaciones del console para depurar código rápidamente

```
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>