CURSO .JS

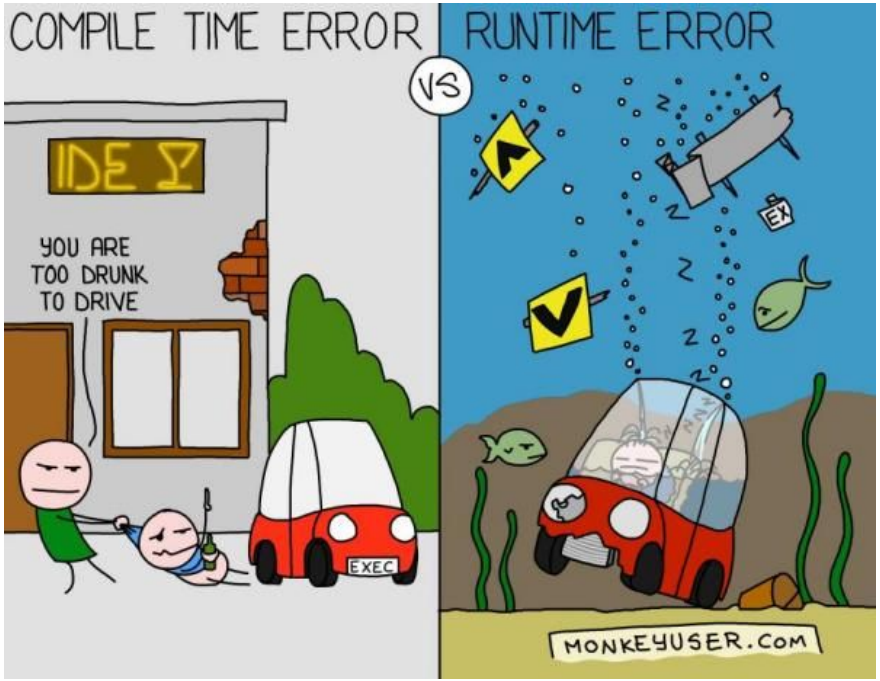
FUNDAMENTOS BÁSICOS

Autor: Jon Vadillo
www.jonvadillo.com

Lenguaje de programación interpretado y ~~de tipado estático~~ dinámicamente tipado del lado del cliente.

¿Qué es JavaScript?

- Los programas escritos en JS son llamados **scripts**.
- Se escriben en **texto plano** y no requieren compilación.
- Se puede escribir **directamente en la página HTML** o en archivos de **extensión .js**.
- Puede ejecutarse **tanto en navegadores como en servidores,** los cuales incluyen un “motor” de JS.
- El motor de JS analiza, “compila” y ejecuta el código aplicando optimizaciones.


COMPILE TIME ERROR
VS
RUNTIME ERROR
IDE
YOU ARE TOO DRUNK TO DRIVE
EXEC
EX
MONKEYUSER.COM

¿Dónde se utiliza JavaScript?

- Navegador
- Servidor
- Aplicaciones de escritorio
- Móvil
- Otros
  - IoT, robots, scripts,...

JS en el navegador

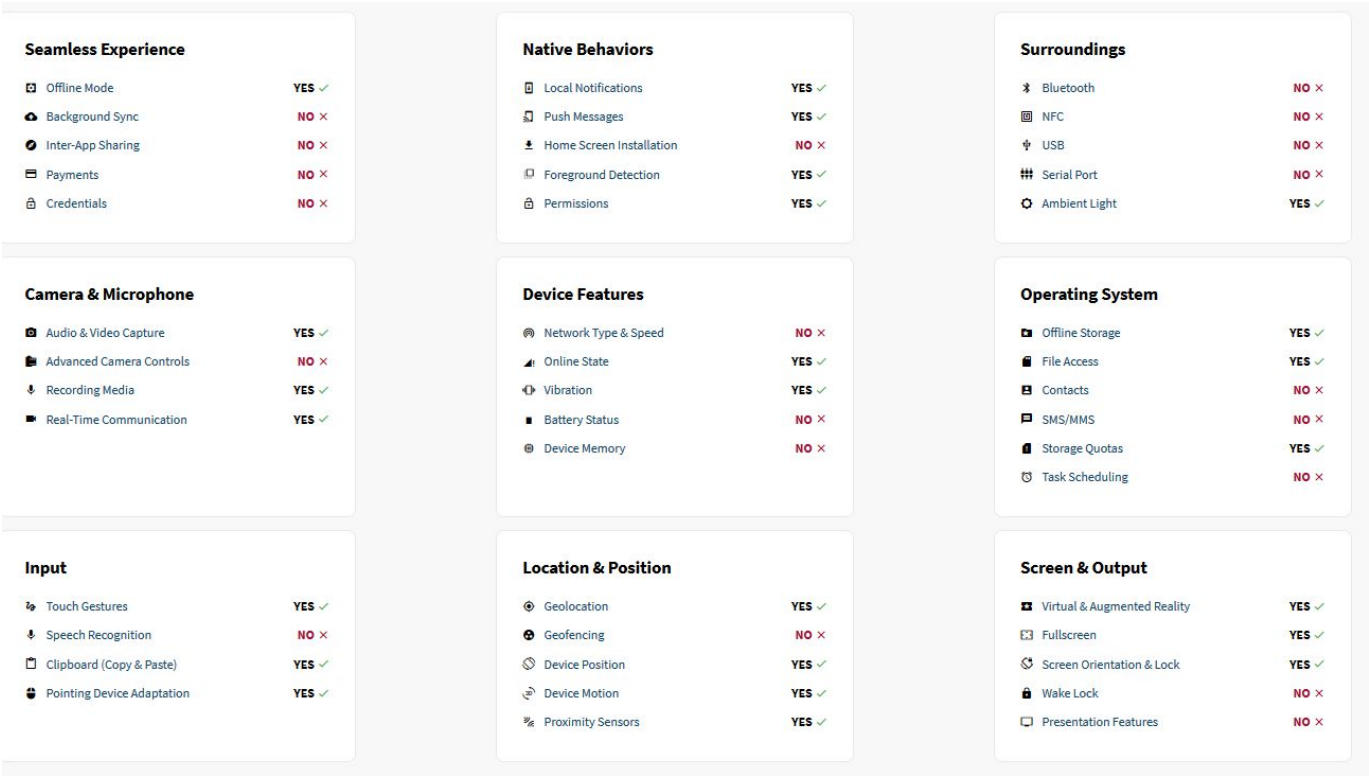
- Añadiendo funcionalidad

```
<script>
    alert( 'Hola, Mundo!' );
</script>
```

- Single Page Apps (SPA)

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

<div id="app"></div>
```

JS en el navegador



https://whatwebcando.today

Navegador/Móvil

- Pueden **reemplazar** aplicaciones móviles
- 2 alternativas
  - **Progressive Web Apps**
  - **React Native**

Progressive Web Apps (PWA)

- El usuario visita la URL
- Guarda la aplicación en el escritorio
- Son rápidas, ahorran espacio, funciona offline, notificaciones, ...

Source: web.dev

React Native

- Learn once, write anywhere.
- Written in JavaScript, rendered with native code

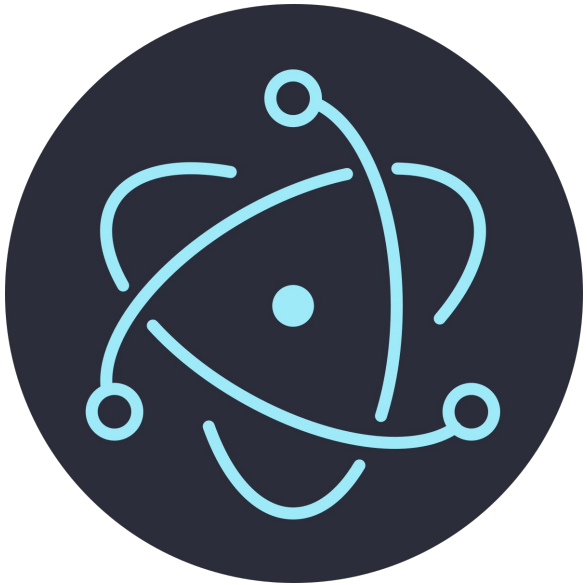
Servidor

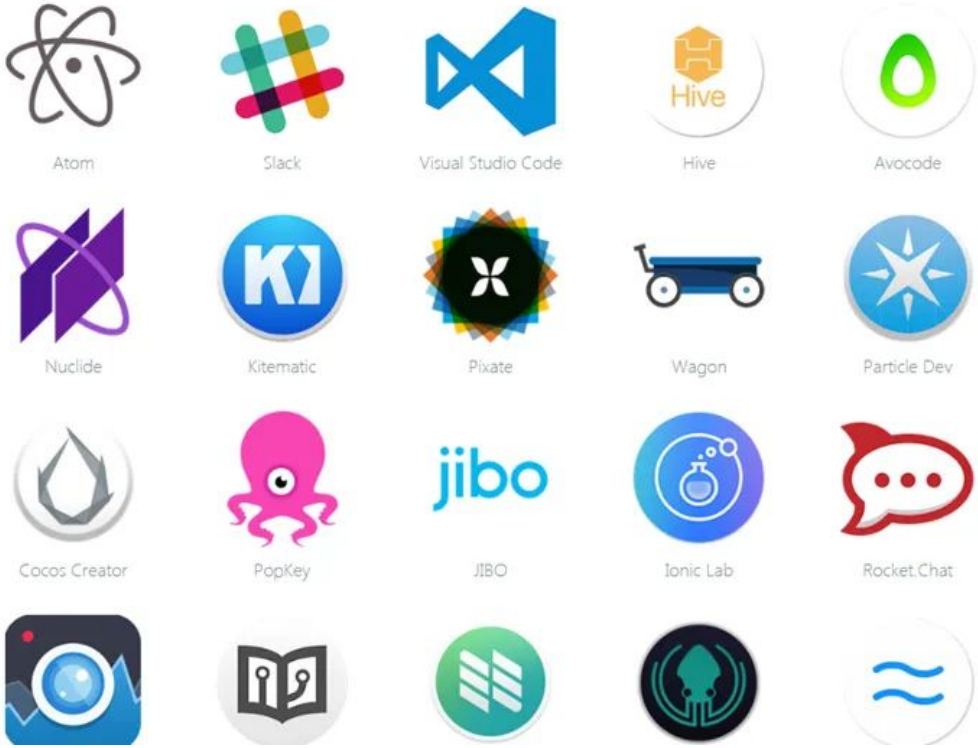
- Servidor
- Basado en eventos asíncronos
- Basado en el motor V8 de Google

```
const http = require('http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hola Mundo');
});

server.listen(port, hostname, () => {
  console.log(`El servidor se está ejecutando en http://${hostname}:${port}/`);
});
```

Aplicaciones de Escritorio

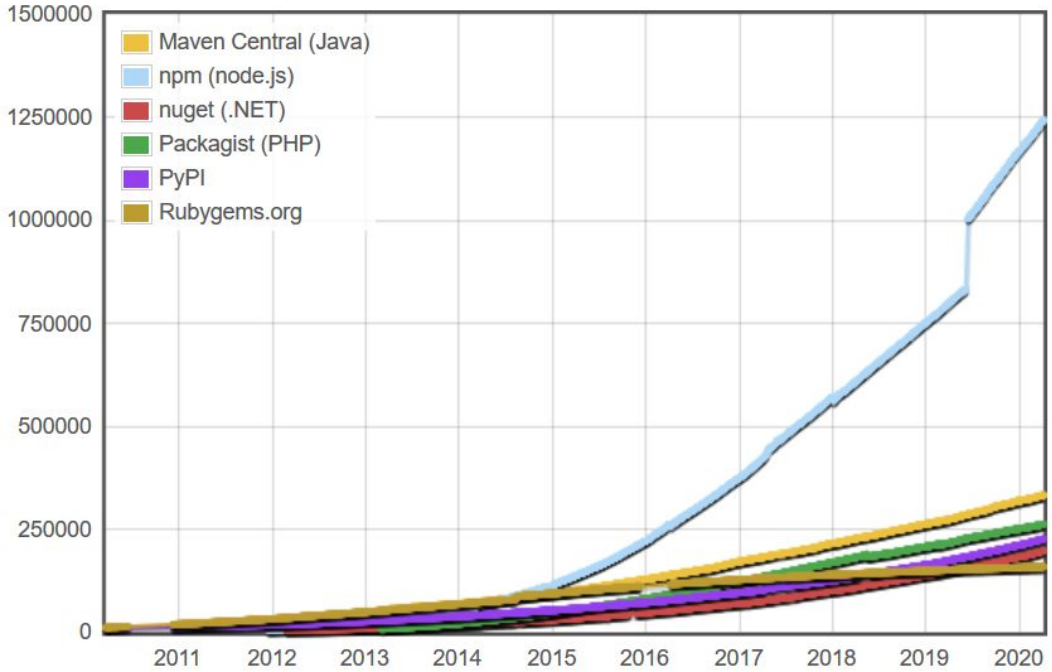
- Framework Electron
- Aplicaciones de escritorio creadas utilizando HTML, CSS y JS.
- Multiplataforma: Mac, Windows y Linux
- Open Source





https://www.electronjs.org/apps

¿Se utiliza tanto JavaScript?



http://www.modulecounts.com/

JavaScript en la Web

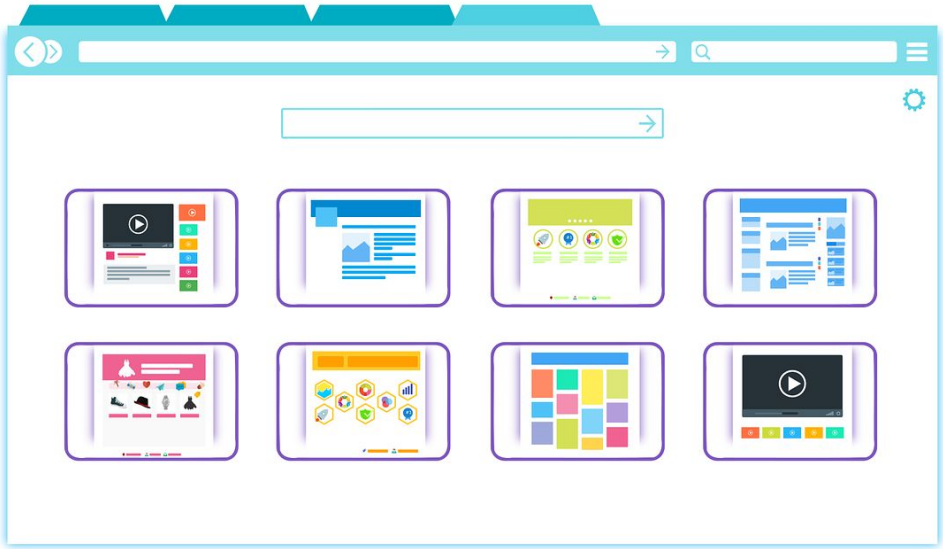


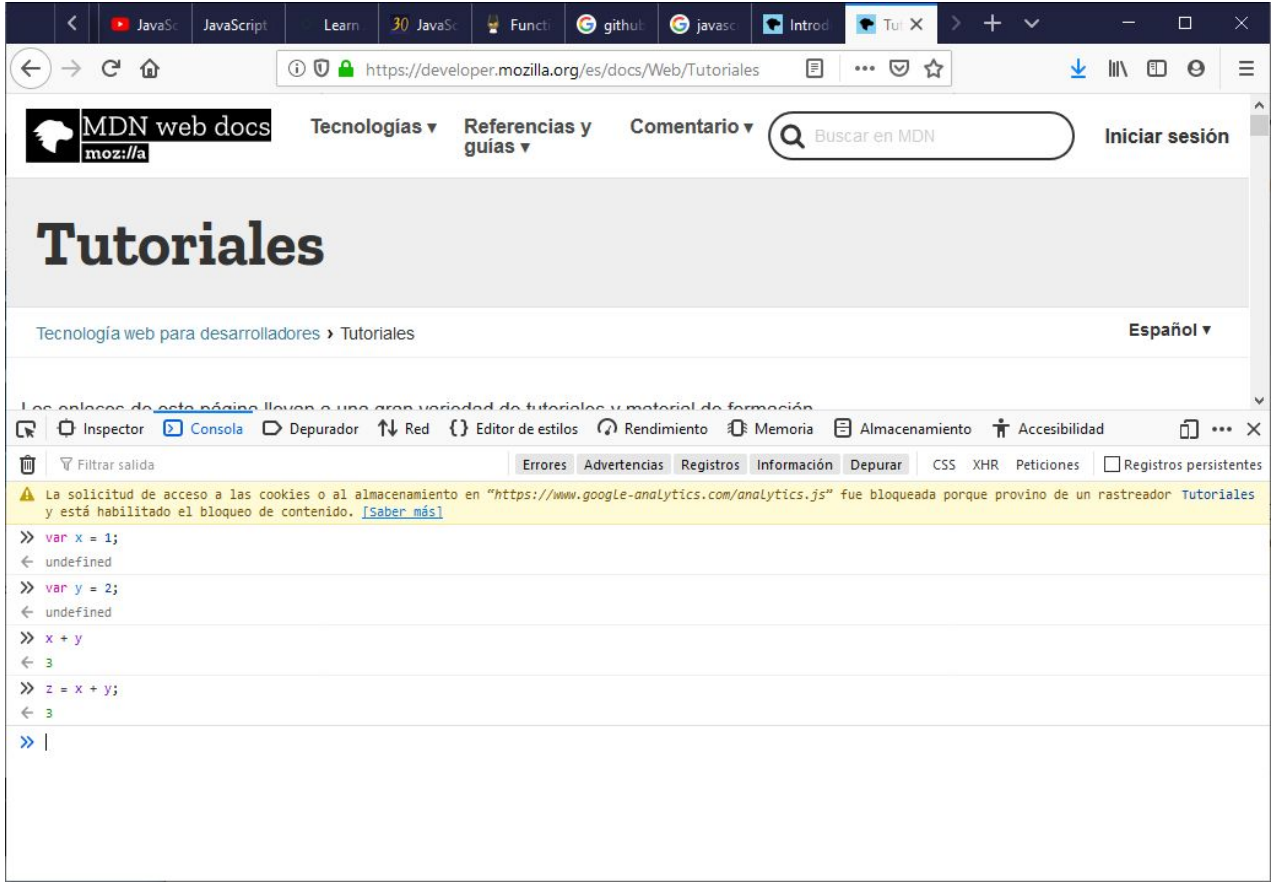
image: needpix.com

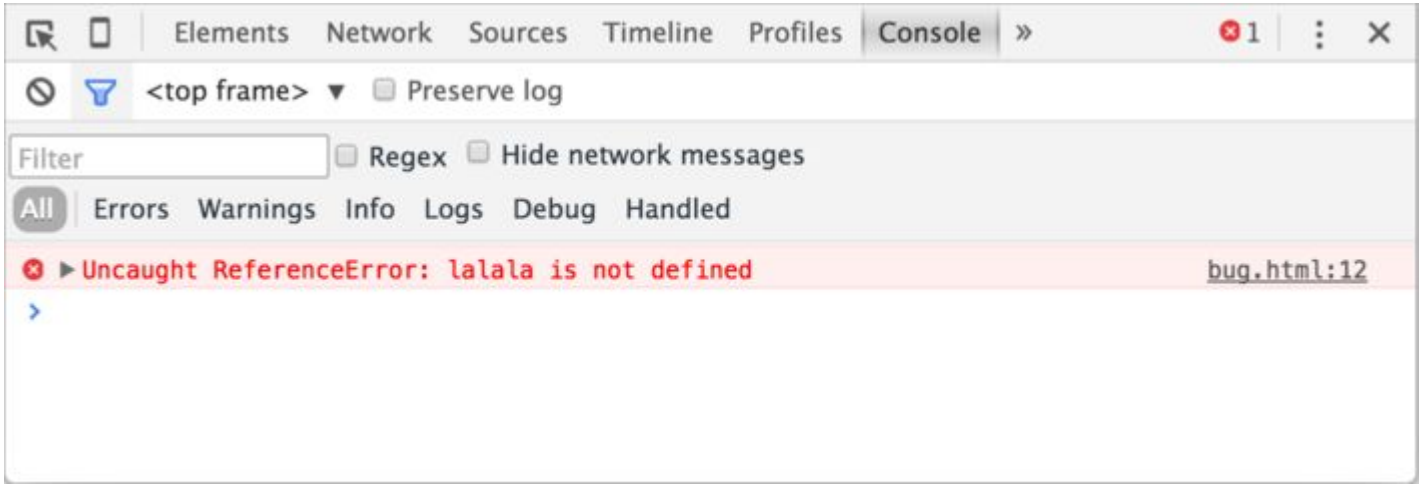
¿Qué puede hacer JavaScript?

- **Modificar los elementos HTML** (añadir un botón, cambiar un estilo, ocultar un formulario, etc.).
- **Interactuar** con el usuario: reaccionar ante un click en un botón, al escribir un texto, etc.
- **Enviar y recibir datos** de un servidor.
- Utilizar el **almacenamiento local** del cliente (*local storage*).
- Utilizar la cámara, GPS, ...

Entorno de desarrollo

- Editor
  - Editor de texto simple: (Sublime Text, Visual Studio Code, Atom)
  - IDEs (WebStorm, NetBeans)
- Navegador (Firefox, Chrome, Safari)
- Consola de desarrollo


https://developer.mozilla.org/es/docs/Web/Tutoriales
MDN web docs
moz://a
Tecnologías ▾
Referencias y guías ▾
Comentario ▾
Buscar en MDN
Iniciar sesión
Tutoriales
Tecnología web para desarrolladores › Tutoriales
Español ▾
Inspector
Consola
Depurador
Red
Editor de estilos
Rendimiento
Memoria
Almacenamiento
Accesibilidad
Filtrar salida
Errores
Advertencias
Registros
Información
Depurar
CSS
XHR
Peticiones
Registros persistentes
La solicitud de acceso a las cookies o al almacenamiento en "https://www.google-analytics.com/analytics.js" fue bloqueada porque provino de un rastreador Tutoriales y está habilitado el bloqueo de contenido. [Saber más]
var x = 1;
undefined
var y = 2;
undefined
x + y
3
z = x + y;
3


Elements
Network
Sources
Timeline
Profiles
Console
»
1
<top frame>
Preserve log
Filter
Regex
Hide network messages
All
Errors
Warnings
Info
Logs
Debug
Handled
Uncaught ReferenceError: lalala is not defined
bug.html:12

¡Hola mundo!

```
<html lang="es">
<head>
   <title>Hello World</title>
</head>
<body>
   <p>Incluye el script donde queramos dentro del body.</p>

   <script>
       alert( 'Hola, Mundo!' );
   </script>
</body>
</html>
```

¡Hola mundo!

```
<html lang="es">
<head>
   <title>Hello World</title>
</head>
<body>
   <p>Incluye el script donde queramos dentro del body.</p>

   <script>
       alert( 'Hola, Mundo!' );
   </script>
</body>
</html>
```

¿Cómo incluir JavaScript?

- Dentro de un elemento HTML (no recomendado)

```
<button type="button" onclick="myFunction()">HAZ CLICK</button>
```

- En el documento entre etiquetas <script> ... </script> (no recomendado)

```
<script>
    alert( 'Hola, Mundo!' );
</script>
```

- En un archivo .js externo

Script externos

- Separar el código JS en un fichero externo y referenciarlo.
- Es posible incluir **archivos locales** (mediante la ruta absoluta o relativa) o de **otros servidores indicando la URL**.

```
<script src="/js/script1.js"></script>
<script src="https://cdnjs.cloudflare.com/vue.js"></script>
```

- La ventaja de incluir un script de forma externa es que el navegador se descargará el fichero y lo guardará en la memoria caché. De esta forma al reutilizarlo en otras páginas no hará falta descargarlo de nuevo.

Hands on!

01. Crea una página que contenga un script con el siguiente código:

```
alert("Soy el script número 1");
```

02. Crea un archivo de extensión `.js` e inclúyelo en la página anterior. Su contenido será el siguiente:

```
alert("Soy el script externo");
```

Sintaxis

- Las sentencias finalizan con **punto y coma**.
- No obstante, los navegadores son capaces de intuir cuándo deben insertar un punto y coma y hacerlo **de forma automática** (se conoce como “automatic semicolon insertion”).

```
var x = 5;
var y = 6
var resultado = x + y
alert(resultado)
```

Comentarios

```
// Comentario de una línea

/**
* Comentario de varias líneas
* de extensión.
*/
```

Declaraciones

- **var** Declara una variable en el ámbito actual (global o local).
- **let** Declara una variable **en el ámbito de bloque.**
- **const** Declara una constante de **sólo lectura** en el ámbito de bloque.

```
const PI = 3.14;

if (PI > y) {
   let resultado = x;
   alert(resultado);
}
```

Variables

- Las variables se utilizan para almacenar información.
- El identificador (nombre de la variable) tiene que empezar con una **letra, un guión bajo (_) o un símbolo de dólar ($)**.
- Los valores subsiguientes pueden ser también números.
- JavaScript diferencia entre mayúsculas y minúsculas.

```
var nombre = "Mikel";
var $edad = 25;
var notaMedia = 7.5;
var Soltero = true;
```

Tipos de datos

- No es necesario especificar el tipo de dato de forma explícita.
- Es un lenguaje dinámicamente tipado, una variable puede cambiar de tipo.
- La función typeof permite conocer el tipo de datos que contiene una variable:
  - typeof x
  - typeof(x).

```
// no dará error
let mensaje = "hello";
mensaje = 123456;
```

Tipos de datos

- **number**: cualquier tipo de número (enteros o de punto flotante).
- **string**: cadenas de texto de uno o varios caracteres.
- **boolean**: true/false.
- **null**: para tipos de datos desconocidos. Representa “nada” o “vacío”.
- **undefined**: para valores sin asignar.
- **object**: estructuras complejas.

```
// Números
let numero = 52;
numero = -3.14;

// Strings
let nombre = 'Nora' //Comilla simple
let saludo = "Hola Nora" //Doble
saludo = `Hola ${nombre}` //Back-tick

let telefono = null

let email;
alert(email); // muestra "undefined"
```

Hands on!

03. Crea una página que muestre una única frase que contenga las siguientes variables:

- Dos strings con identificador: nombre y apellidos
- Un número con identificado: edad

Muestra la frase en pantalla mediante un alert.

Operadores

```
// sumar, restar, multiplicar y dividir números
let resultado = 5 + 5; // 10
resultado = 5 - 2; // 3
resultado = 3 * 3; // 9
resultado = 14 / 2; // 7

let numero = 2;
numero  *= 3 + 5;
numero ( n ); // 16

i++; // equivalente de i = i + 1
i--; // equivalente de i = i - 1;
```

Operadores

```
// concatenar dos cadenas
var frase = "Primera frase. " + "Segunda frase.";
// concatenar con +=
var frase = "I come first. ";
frase += "I come second.";
// construir cadenas con variables
var nombre = "Nora";
var frase = "Hola, me llamo " + nombre + ", y tu?";
// añadir variables a strings
var saludo = "Hola, me llamo ";
saludo += nombre;
```

Comparadores

```
1 < 10; // = true
1 > 10; // = false
2 <= 2; // = true
2 >= 2; // = true

// Solo comprueba si los valores son iguales, no los tipos
"5" == 5; // = true
// Comprobar valor y tipo:
"5" === 5; // = false
```

Comparadores

```
alert( 'Z' > 'A' ); // true
alert( 'Globo' > 'Gle' ); // true
alert( 'Bee' > 'Be' ); // true
alert( 'a' > 'A' ); // true

alert( '2' > 1 ); // true, string '2' se convierte a número 2
alert( '01' == 1 ); // true, string '01' se convierte a número 1

alert( true == 1 ); // true
alert( false == 0 ); // true
```

Not

```
alert( !true ); // false
alert( !0 ); // true
```

Condicionales

- Evalúa la condición entre paréntesis.
- Si el resultado es **true**, se ejecuta el código del bloque.
- Es posible encadenar condiciones **else if** o definir un **else** al final.

```
let contador = 1;
if (contador === 3){
    // se ejecuta si contador es 3
} else if (contador === 4){
    // se ejecuta si contador es 4
} else {
    // se ejecuta si contador no es 3 ni 4
}
```

Hands on!

04. Crea una página que evalúe si el valor de una variables es mayor, igual o menor que 18. En caso de ser igual o mayor mostrará el mensaje: “Ya puedes votar”, en caso de ser menor, mostrará cuántos años le quedan para votar. Por ejemplo, en caso de ser 15, mostrará el mensaje: “Todavía te quedan 3 años para votar”.

Abreviación

```
let result = condition ? value1 : value2;
```

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

Operador lógico OR

```
alert( true || true );   // true
alert( false || true );  // true
alert( true || false );  // true
alert( false || false ); // false
```

Operador lógico OR

```
let hora = 9;

if (hora < 10 || hora > 18) {
  alert( 'La tienda está cerrada.' );
}

hora = 12;
let esDomingo = true;

if (hora < 10 || hora > 18 || esDomingo) {
  alert( 'La tienda está cerrada.' ); // es domigo
}
```

Operador lógico AND

```
alert( true && true );   // true
alert( false && true );  // false
alert( true && false );  // false
alert( false && false ); // false
```

Operador lógico AND

```
if (nombre == 'Amaia' && edad == 30) {
  alert( 'Se llama Amaia y tiene 30 años' );
}
```

Bucle while

- Utilizamos bucles cuando necesitamos repetir acciones.
- Mientras se cumpla la condición, el código del bloque se seguirá ejecutando.

```
let i = 0;
while (i < 3) { // muestra 0, 1, 2
  alert( i );
  i++;
}
```

Bucle do...while

- La comprobación de la condición se realiza después de la primera ejecución.
- Asegura ejecutar el código del bloque al menos 1 vez.

```
let i = 0;
do {
  alert( i );
  i++;
} while (i < 3);
```

Hands on!

05. Crea un programa que itere por un bucle while tantas veces como el número almacenado en una variable.

06. Crea un programa que añada puntos (‘.’) al final de un string hasta que la longitud de ese string sea 30. Si la longitud es mayor que 30 no hará nada. Finalmente lo mostrará por pantalla.

Ejemplo: si el string es “casa”, el resultado será: “casa.....”

Nota: puedes utilizar la función prompt() para solicitar la frase al usuario:

```
let frase = prompt('Introduce tu frase:');
```

Bucle for

- Está compuesto por 3 partes:
  - Estado inicial
  - Condición a evaluar
  - Cambio tras cada iteración
- Puede declarar una variable en el estado inicial.

```
for (begin; condition; step) {
    // ... loop body ...
}
```

```
// muestra 0, luego 1, luego 2
for (let i = 0; i < 3; i++) {
    alert(i);
}
```

Finalizar el bucle: break

```
let suma = 0;

while (true) {

  suma += 2

  if (suma > 10) break;

}
```

```
let suma = 0;

for (let i = 0; i < 20; i++) {

    suma += 2

    if (suma > 10) break;
}
```

Saltar una iteración: continue

```
let suma = 0;

while (suma < 10) {

  suma += 1

  if (suma == 5) continue;

  alert(suma)

}
```

```
let suma = 0;

for (let i = 0; i < 10; i++) {

    if (i == 5) continue;

     alert(suma)

}
```

Hands on!

07. Crea un programa que muestre el resultado de la suma de todos los números del 20 al 50.

08. Crea un programa que itere por los números del 1 al 20 y muestre por pantalla únicamente los números pares.

Nota: utiliza la función `console.log()` para mostrar los números en la consola del navegador.

switch

- Reemplaza el uso de múltiples IFs de forma más descriptiva.
- Contiene varias cláusulas **case** que se van evaluando en orden. Si alguna coincide, se ejecuta el bloque de código hasta llegar al break.
- Exista una última opcional, llamada **default**, que se ejecutará si ningún case se ha ejecutado.

```
switch(x) {
    case 'value1':  // if (x === 'value1')
      ...
      [break]

    case 'value2':  // if (x === 'value2')
      ...
      [break]

    default:
      ...
      [break]
  }
```

switch

- Compara también el tipo del valor.
- Los valores a comparar pueden ser expresiones (p.ej. 1+1)

```
switch (numero) {
  case 1:
    alert( 'Muy pequeño' );
    break;
  case 2:
    alert( '¡Exacto!' );
    break;
  case 3:
    alert( 'Muy grande' );
    break;
  default:
    alert( "No tengo ni idea" );
}
```

Funciones

- Las funciones contienen un bloque de código y permiten ejecutarlo tantas veces como necesitemos, sin utilizar bucles.
- Evitan tener que repetir el mismo código en sitios distintos.
- Las funciones por lo general representan acciones, por lo tanto el nombre de la función será un verbo y describirá lo que hace.

```
function nombre_func(parametros) {
    ...código...
}
```

```
function saludar(nombre) {
    alert( 'Hola '+nombre );
}

saludar('Markel');
saludar('Nora');
```

Nombres de las funciones

- Las funciones por lo general **representan acciones**, por lo tanto el nombre de la función será un verbo y describirá lo que hace de forma breve:
  - showMessage(..) → muestra un mensaje
  - getName(..) → devuelve el nombre
  - calcSum(..) → calcula la suma y devuelve el resultado
  - createForm(..) → crea un formulario
  - checkPermission(..) → comprueba los permisos y devuelve true/false

Hands on!

09. Crea una función que reciba dos strings y muestre en un alert la concatenación de ambos..

10. Crea la función esPar que reciba un número y saque una alerta diciendo si el número es par o impar.

1 función → 1 tarea

- Las funciones deberían ser **simples y hacer una única cosa**.
- Cuando una función sea compleja, debería **dividirse en varias funciones más simples**.
- Ejemplo:
  - calcularNumerosPrimos(desde, hasta): una función que devuelva un listado de números primos dentro de un rango, debería separarse en dos funciones:
    - esPrimo(numero) → función que devuelve true/false si un número es primo.
    - calcularNumerosPrimos(desde, hasta) → recorre todos los números en el rango desde-hasta y utiliza la función esPrimo para decidir si incluirlos al array de números primos.

Variables locales

- Las variables definidas dentro de la función, solo están disponibles en esa función.
- Una función puede acceder y modificar una variable definida fuera.

```
let nombreExterior = 'Unai';
function mostrarNombre() {
    let nombreInterior = "Amaia"; // variable local
    alert(nombreInterior);
    alert(nombreExterior); // Puede acceder
  }

  mostrarNombre();
  alert( nombreInterior ); // Error! Es variable local
```

Valor por defecto

```
function mostrarMensaje(nombre, texto = "no hay texto") {
    alert( nombre + ": " + texto );
}

mostrarMensaje("Aitor"); // Aitor: no hay texto
```

Hands on!

11. Crea una función llamada *saludar* que reciba un nombre y muestre mediante una alerta un saludo. En caso de no recibir especificar ningún parámetro, utilizará como nombre la palabra anónimo. Aquí tienes algunos ejemplos:

- saludar(“Aritz”) → Hola, Aritz
- saludar(“Izaro”) → Hola, Izaro
- saludar() → Hola, anónimo

Devolver un valor

- Se utiliza la palabra **return** para devolver un valor.
- Únicamente se puede devolver **un valor**.

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

let result = sumar(1, 2);
alert( result ); // 3
```

Hands on!

12. Escribe una función *sumarConInteres* que sume dos cantidades y aplique un interés al resultado. Para cualquier suma menor o igual a 10, tendrá que sumarle un interés de 1. Para cada cantidad mayor que 10, el interés a sumar será 2. Por ejemplo, la llamada *sumarConInteres* (5, 3) devolverá 9 y la llamada a *sumarConInteres(7,9)* devolverá 18.

Arrow Functions

```
let func = function(arg1, arg2, ...argN) {
  return expression;
};
```

```
let func = (arg1, arg2, ...argN) => expression
```

Arrow Functions

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

```
  let sumar = (a, b) => a + b;

  alert( sumar(1, 2) ); // 3
```

Arrow Functions

```
let doble = n => n * 2;
// igual que: let doble = function(n) { return n * 2 }

alert( doble(3) ); // 6
```

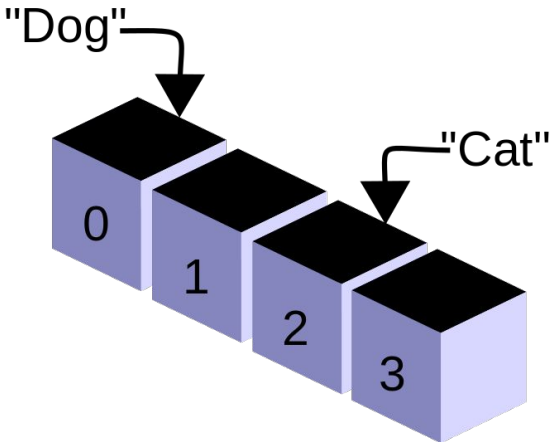
```
let saludar = () => alert("Hola!");

saludar();
```

Multiline Arrow Functions

```
let sum = (a, b) => {  // las llaves son necesarias
    let result = a + b;
    // si hay llaves, hay que especificar "return"
    return result;
};

alert( sum(1, 2) ); // 3
```

Arrays (arreglos o matrices)

- Permiten **almacenar de manera ordenada una lista de elementos** de datos en una única variable.
- Contienen múltiples valores almacenados en una lista, por lo que es posible realizar operaciones como: acceder a cada valor, recorrerlos en bucle, añadir o eliminar elementos de la lista, etc.



Source: Wikimedia

Declaración

```
// Declarar array vacío
let arr = new Array();
let arr = [];

// Inicializar array con valores
let estudiantes = ["Amaia", "Markel", "Nora"];
let valores = [14, true, "Hola"];
```

Acceso a elementos

```
let estudiantes = ["Amaia", "Markel", "Nora"];

alert( estudiantes[0] ); // Amaia
alert( estudiantes[1] ); // Markel
alert( estudiantes[2] ); // Nora
```

Modificar o añadir elementos

```
let estudiantes = ["Amaia", "Markel", "Nora"];
estudiantes[1] = 'Ane'; // ahora ["Amaia", "Ane", "Nora"]

// Añadir un elemento:
estudiantes[3] = 'Unai';
// ahora ["Amaia", "Ane", "Nora", "Unai"]
```

Operaciones comunes

```
// Añadir elemento al final
estudiantes.push('Lorea'); // devuelve la longitud

// Eliminar el último elemento
estudiantes.pop(); // devuelve el elemento eliminado

// Eliminar el primer elemento
estudiantes.shift(); // devuelve el elemento eliminado

// Añadir elementos al comienzo
estudiantes.unshift("Lorea");
```

Recorrer un array (for)

```
let estudiantes = ["Amaia", "Markel", "Nora"];

for (let i = 0; i < estudiantes.length; i++) {
  alert( estudiantes[i] );
}
```

Recorrer un array (forEach)

```
let estudiantes = ["Amaia", "Markel", "Nora"];

estudiantes.forEach(elemento => {
    console.log(elemento)
});

estudiantes.forEach((elemento, indice) => {
    console.log(indice);
    console.log(elemento);
});
```

Recorrer elementos de un array

```
let estudiantes = ["Amaia", "Markel", "Nora"];

// iterar sobre los elementos
for (let estudiante of estudiantes) {
  alert( estudiante );
}
```

Hands on!

13. Crea dos arrays, uno con 4 tipos de *animales* y otro con 3 nombres de *colores*.

- Muestra por pantalla el segundo elemento de cada array.
- Obtén el número de elementos de cada array y muéstralo por pantalla.
- Añade un elemento al final del array *animales* utilizando una función.
- Añade un elemento al principio del array *colores* utilizando una función.
- Recorre el array *animales* mostrando por pantalla cada uno de sus elementos.

Array multidimensional

```
let arr = [
    [1, 2, 3],
    [101, 102, 103],
    [201, 202, 203]
  ];

  alert( arr[1][1] ); // 102
```

Objetos

- Un objeto es una **colección de propiedades** (y una propiedad es una asociación entre un nombre (o clave) y un valor).
- Se puede definir mediante **pares clave-valor** entre llaves { }

```
let estudiante = { // el objeto
  nombre: "Aitor", // la clave "nombre" almacena el valor "Aitor"
  edad: 35         // la clave "edad" almacena el valor 35
};
```

Crear un objeto

```
let estudiante = new Object(); // usa el constructor de Object
let estudiante = {};  // se conoce como "object literal"
```

```
let estudiante = { // el objeto
  nombre: "Aitor", // la clave "nombre" almacena el valor "Aitor"
  edad: 35         // la clave "edad" almacena el valor 35
};
```

Acceder a las propiedades

```
let persona = {
    nombre: "Aitor",
    edad: 35,
    isEstudiante: true
};

// acceder a las propiedades
alert( persona.nombre ); // Aitor
// Alternativa:
alert( persona["edad"] ); // 35
```

Añadir/Eliminar una propiedad

```
let persona = {
    nombre: "Aitor",
    edad: 35,
    isEstudiante: true
};
// Añadir la propiedad 'email'
persona.email = "aitor@mail.com";

// Eliminar la propiedad 'edad'
delete persona.edad
```

Hands on!

14. Sigue las siguientes instrucciones:

- Crea un objeto llamado *aplicacion* con la propiedad nombre (el valor será "*Web App Industrial*".
- Añade las propiedades llamadas puerto (con el valor *8000*) y hostname (con el valor "*localhost*")
- Muestra el valor de cada propiedad por pantalla.
- Elimina la propiedad *hostname*.

Comprobar que la propiedad existe

```
let persona = { nombre: "Aitor", edad: 35 };

alert( "nombre" in persona ); // true, si que existe
alert( "apellido" in persona ); // false, no existe persona.apellido
```

Iterar por las propiedades de un objeto

```
let persona = {
    nombre: "Aitor",
    edad: 35,
    isEstudiante: true
};

for (let clave in persona) {
    // claves
    alert( clave );  // nombre, edad, isEstudiante
    // valores de las claves
    alert( persona[clave] ); // Aitor, 35, true
}
```

Hands on!

- **15.** Modifica el código del ejemplo anterior para que muestre el valor de todas las propiedades del objeto *aplicacion* mediante un bucle *for*.
- **16.** Crea un objeto llamado *cuentaCorriente* que contenga las siguientes propiedades:
    - *saldoInicial*: con valor de *1500*
    - *compras*: con valor de *-300*
    - *alquiler*: con valor de -*800*

  Crea una función que reciba el objeto y utilizando un bucle calcule la suma de los valores de las propiedades del objeto. En el ejemplo devolverá *400 (resultado de 1500-300-800)*.

Hands on!

- **17.** Crea un objeto llamado *inversiones* que contenga las siguientes propiedades:
    - *fondos*: con valor de *300*
    - *oro*: con valor de *400*

  Crea una función que reciba el objeto y un número entero por el cual tendrá que multiplicar todos los valores del objeto. Por ejemplo, si el número es *2*, *fondos* tendrá un valor de *600* y *oro* tendrá un valor de *800*.

Métodos en objetos

```
persona = {
  nombre: "Kepa",
  edad: 35,

  saludar: function() {
    alert("Hola");
  }
};

persona.saludar(); // Hola
```

Métodos en objetos (forma abreviada)

```
persona = {
  nombre: "Kepa",
  edad: 35,

  saludar() {
    alert("Hola");
  }
};

persona.saludar(); // Hola
```

Uso de ‘this’

```
persona = {
  nombre: "Kepa",
  edad: 35,

  saludar() {
    alert("Hola " + this.nombre);
  }
};

persona.saludar(); // Hola
```

Clases

- Permiten **almacenar de manera ordenada una lista de elementos** de datos en una única variable.
- El constructor inicializa el objeto creado, pudiendo declarar nuevas variables.

```
class Persona {
  edad = 20;
  constructor(nombre) {
    this.nombre = nombre;
  }
  saludar() {
    alert("Hola " + this.nombre);
  }
}
// Uso:
let persona = new Persona("Amaia");
persona.saludar();
```

Sources

- Mozilla MDN: https://developer.mozilla.org/es/
- Modern JavaScript: https://javascript.info/