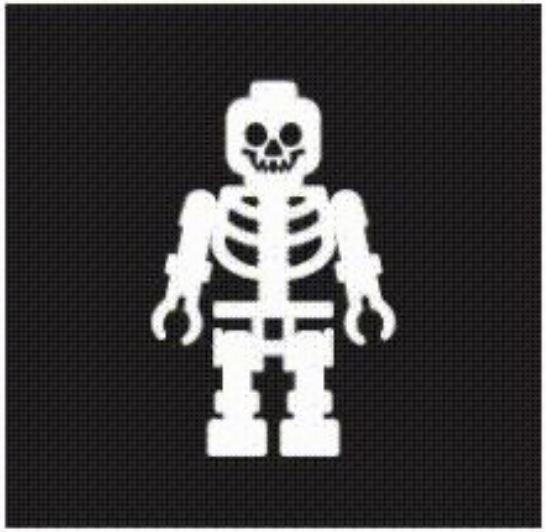
JavaScript:
Fundamentos y sintaxis moderna

Alicia Vázquez
@aliciaFPInf

Índice

Lenguajes que se ejecutan en el Navegador (cliente)

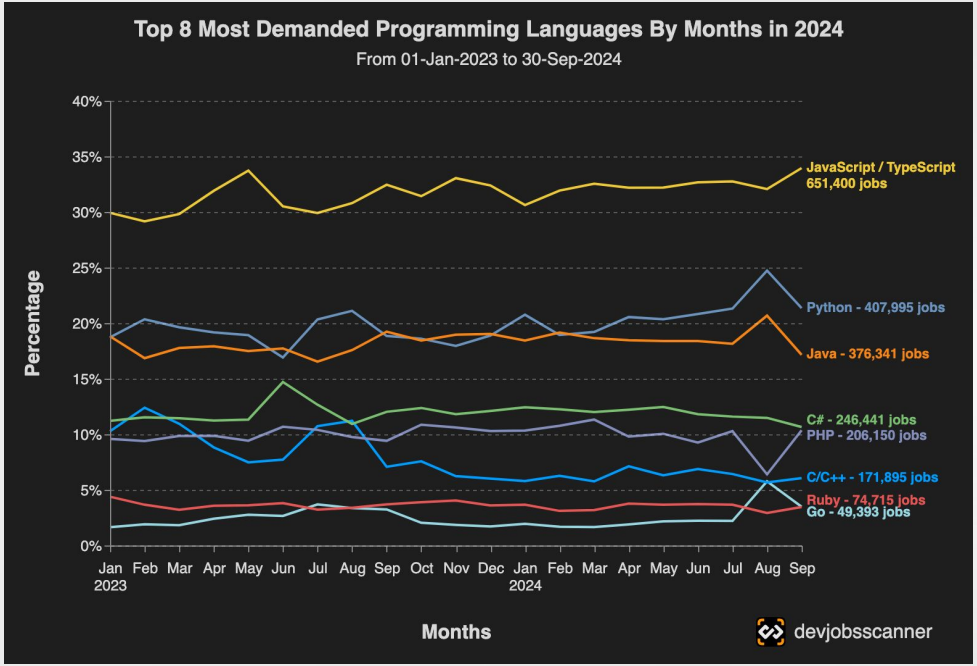
HTML
structure

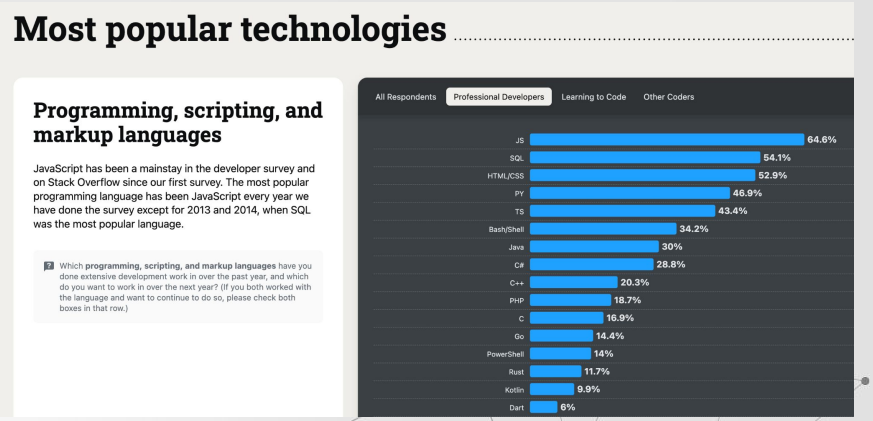


CSS
presentation/appearance

JavaScript
dynamism/action

Datos: Devjobsscanner y StackOverflow





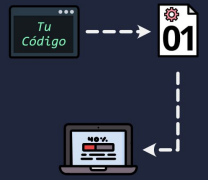
Datos: Tiobe

Nov 2024	Nov 2023	Change	Programming Language
1	1		Python
2	3	↑	C++
3	4	↑	Java
4	2	↓	C
5	5		C#
6	**6**		**JavaScript**
7	13	⇈	Go
8	12	⇈	Fortran
9	8	↓	Visual Basic

TIPOS DE LENGUAJE

COMPILADO

Convierte el código a binarios que lee el sistema operativo.

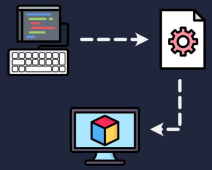


INTERPRETADO

Requieren de un programa que lea la instrucción del código en tiempo real, y la ejecute.

INTERMEDIO

Se compila el código fuente a un lenguaje intermedio y este último se ejecuta en una máquina virtual



Aprende a programar en cualquier lenguaje (primer curso gratis) en:

👉 ed.team/programacion

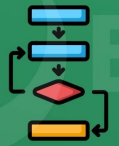
EDteam

¿QUÉ SON LOS PARADIGMAS DE PROGRAMACIÓN?

Los paradigmas son los diferentes estilo de usar la programación para resolver un problema.

PROGRAMACIÓN ESTRUCTURADA

Programación secuencial con la que todos aprendemos a programar. Usa ciclos y condicionales.



PROGRAMACIÓN ORIENTADA A OBJETOS

Divide los componentes del programa en objetos que tienen datos y comportamiento y se comunican entre sí.

PROGRAMACIÓN REACTIVA

Observa flujos de datos asincrónicos y reacciona frente a sus cambios.

PROGRAMACIÓN FUNCIONAL

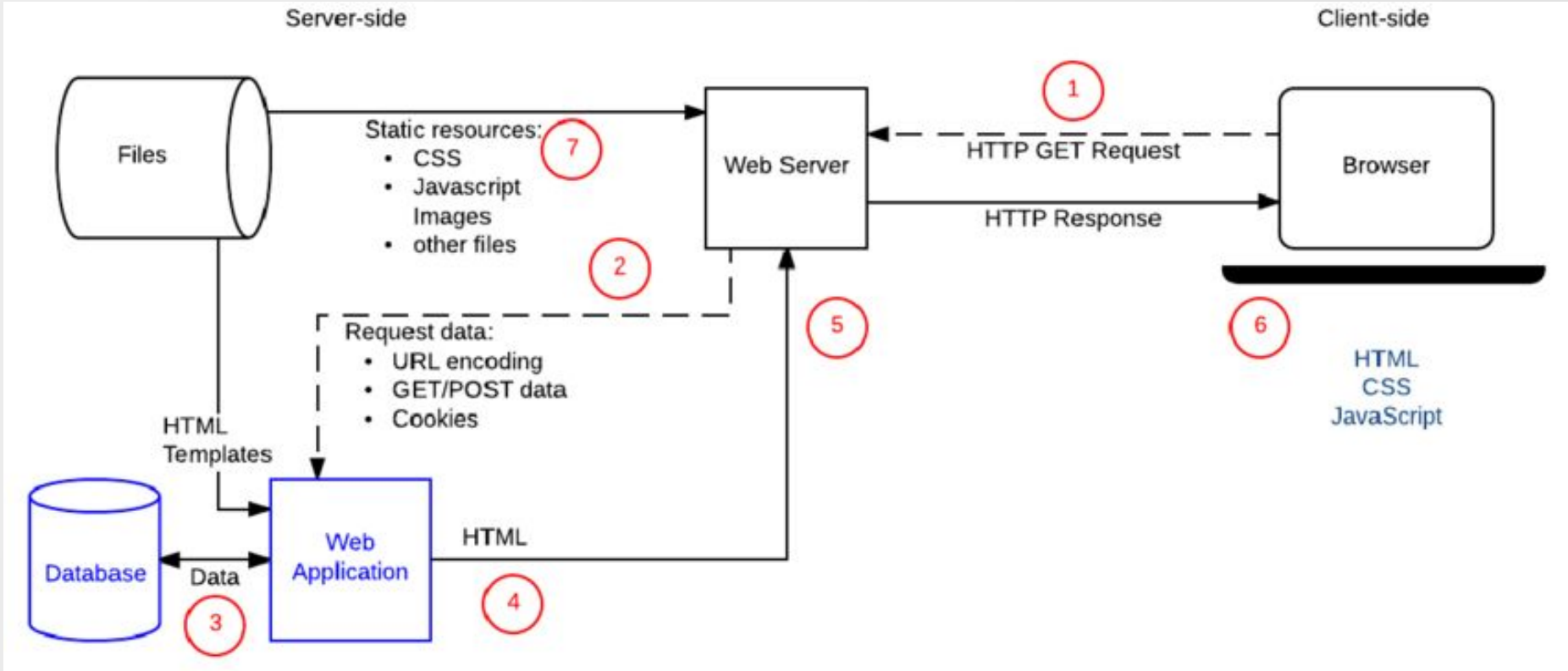
Divide el programa en tareas pequeñas que son ejecutadas por funciones.

Aprende a programar en cualquier lenguaje (primer curso gratis) en:

👉 ed.team/cursos/paradigmas

EDteam

Aplicaciones distribuidas: Patrón CLIENTE - SERVIDOR

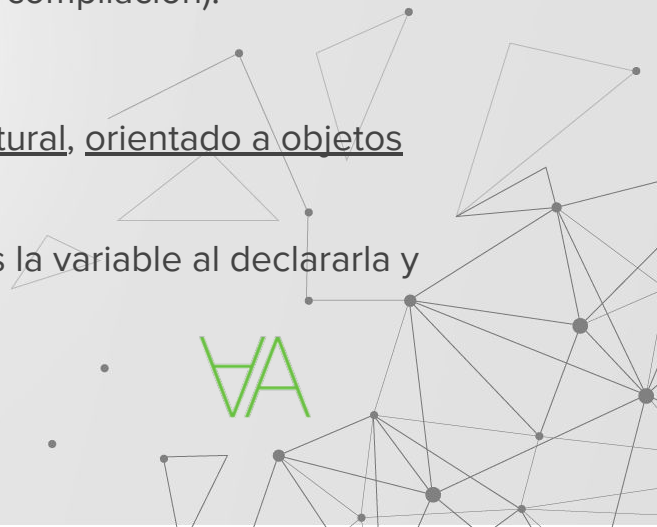


Javascript

Los tres lenguajes nativos de la web son HTML (contenido y su estructura), CSS (diseño del contenido y su estructura) y JavaScript que fue creado para "*dar vida a la web*".

JavaScript es el único lenguaje de programación que funciona en los navegadores de forma nativa es un

- Lenguaje **interpretado** de alto nivel (no tiene necesidad de compilación).
- **Multiplataforma**, no depende de un sistema operativo.
- **Multiparadigma** aunque se define como imperativo, estructural, orientado a objetos (pero basado en prototipos) y orientado a eventos
- **Débilmente tipado**, no hace falta especificar de qué tipo es la variable al declararla y además puede variar su tipo.



Javascript

Se utiliza principalmente en el lado del cliente, implementado como parte de un navegador web permitiendo crear interacción con el usuario y páginas web dinámicas.

- Cambiar el contenido de la página
- Cambiar los atributos de un elemento
- Cambiar la apariencia de algo
- Validar datos de formularios
- etc.

Actualmente es posible ejecutar JavaScript en el propio servidor (**NodeJS**) o en cualquier dispositivo que tenga un intérprete de JS. Javascript es también un lenguaje de back-end.

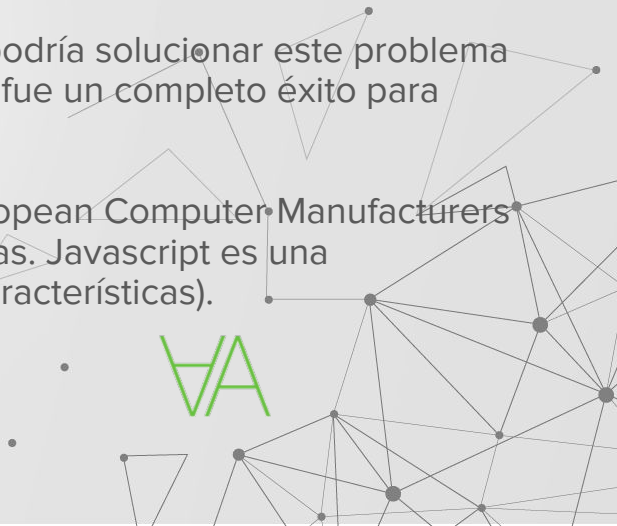
Breve História

A principios de los **años 90**, la mayoría de usuarios que se conectaban a Internet lo hacían con módems a una velocidad máxima de 28.8 kbps. En esa época, empezaban a desarrollarse las primeras aplicaciones web y por tanto, las páginas web comenzaban a incluir formularios complejos.

Con unas aplicaciones web cada vez más complejas y una velocidad de navegación tan lenta, surgió la necesidad de un lenguaje de programación que se ejecutará en el navegador del usuario. De esta forma, si el usuario no rellenaba correctamente un formulario, no se le hacía esperar mucho tiempo hasta que el servidor volviera a mostrar el formulario indicando los errores existentes.

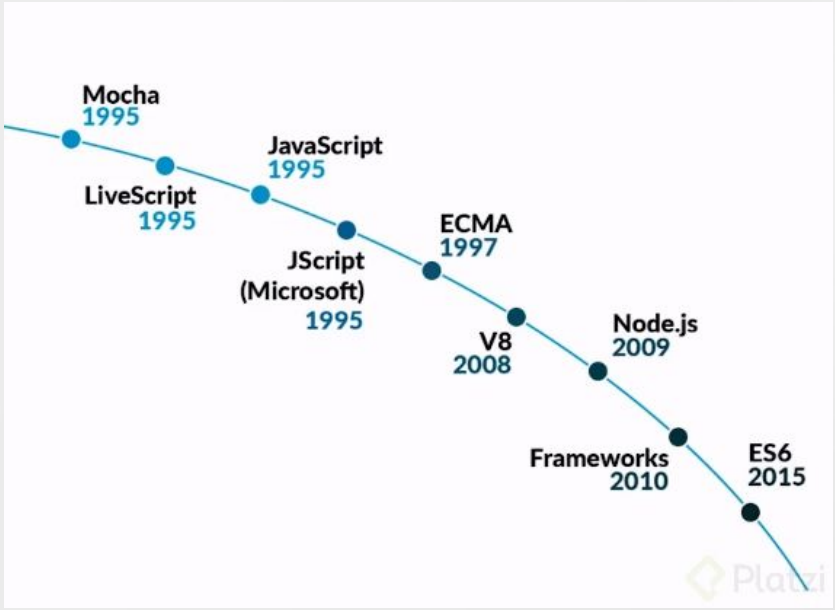
Brendan Eich, un programador que trabajaba en Netscape, pensó que podría solucionar este problema y lanzó en 1995 su lenguaje LiveScript. La primera versión de JavaScript fue un completo éxito para Netscape.

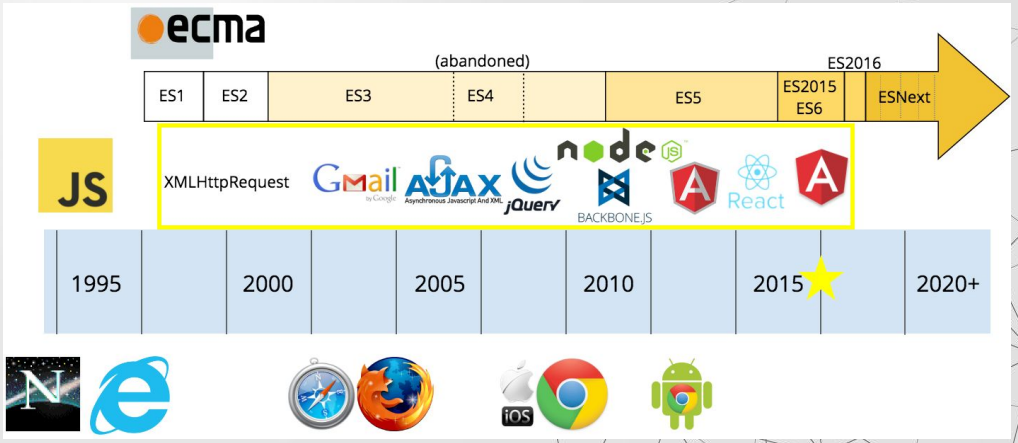
En 1997 se envió la especificación JavaScript 1.1 al organismo **ECMA** European Computer Manufacturers Association) y así estandarizar el lenguaje y evitar batallas entre empresas. Javascript es una implementación del lenguaje **ECMAScript** (el estándar que define sus características).

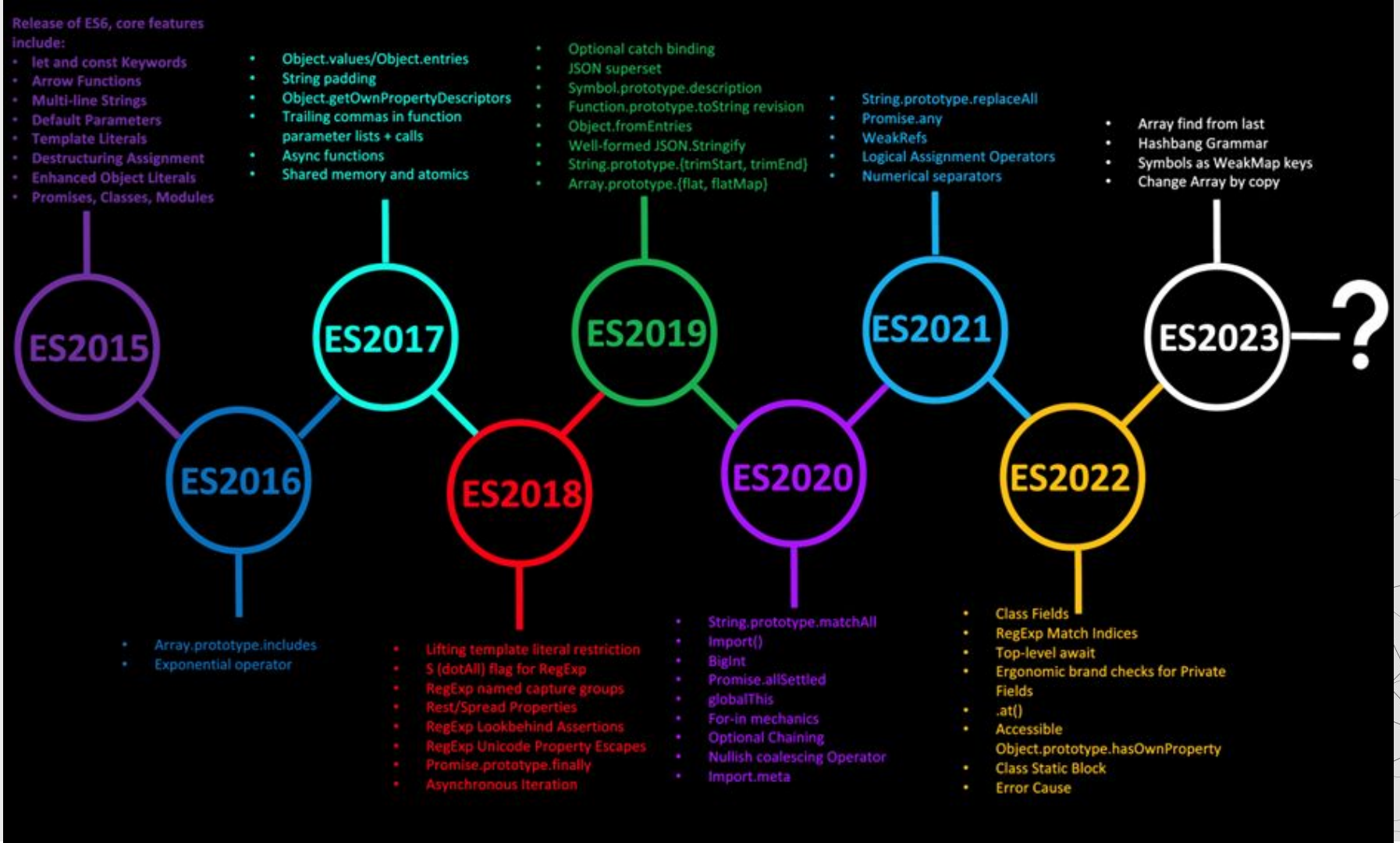


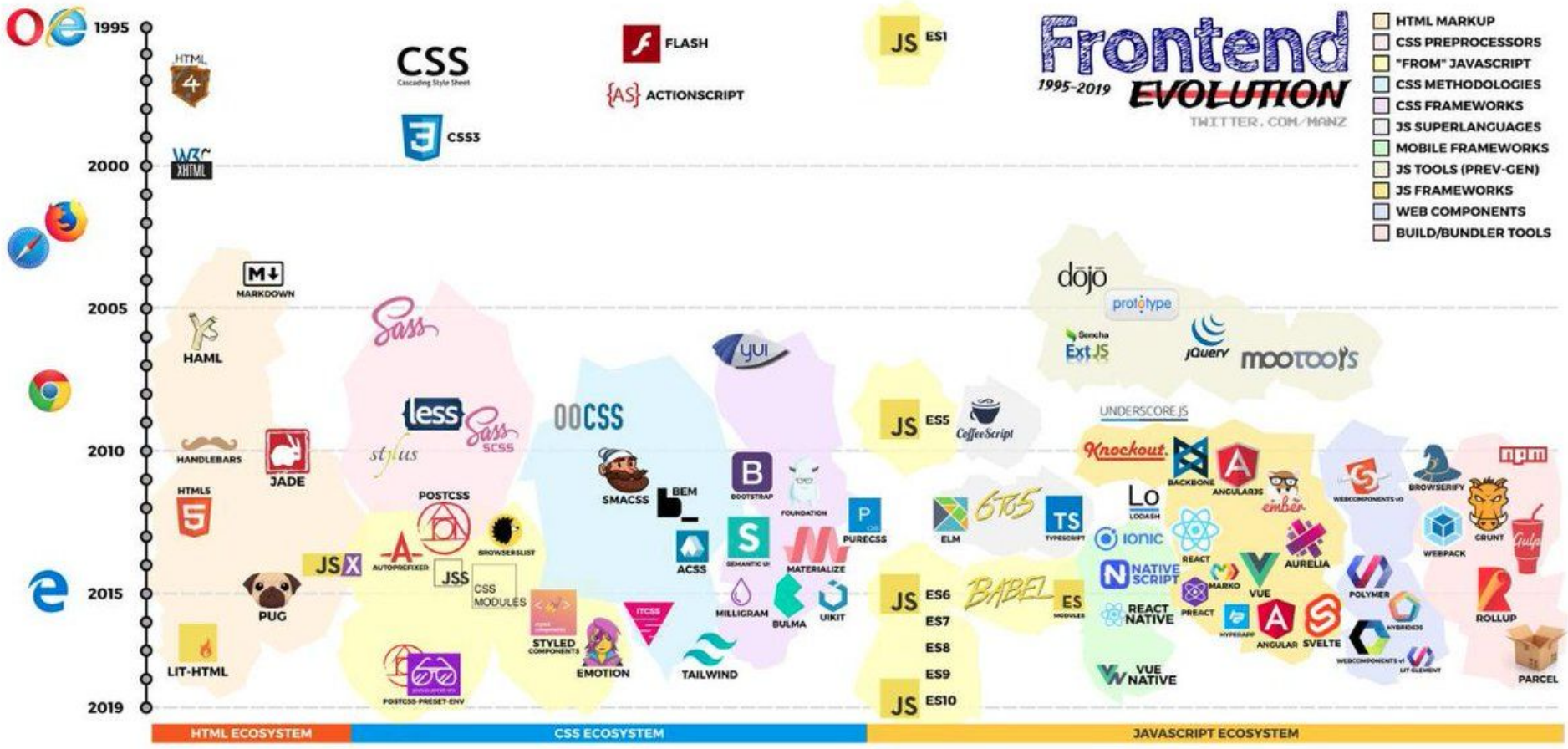
Breve história

Todos los navegadores a partir de 2012 soportan al menos la versión ES5.1 completamente. En **2015** se lanzó la 6ª versión, inicialmente llamada **ES6** y posteriormente renombrada como **ES2015**, que introdujo importantes mejoras en el lenguaje (*let*, *const* y la función flecha), la que supuso una inflexión en el desarrollo de JS. Desde entonces van saliendo nuevas versiones cada año que introducen cambios pequeños. La última es la **ECMAScript 2023**.






Release of ES6, core features include:
let and const Keywords
Arrow Functions
Multi-line Strings
Default Parameters
Template Literals
Destructuring Assignment
Enhanced Object Literals
Promises, Classes, Modules
ES2015
Array.prototype.includes
Exponential operator
ES2016
Object.values/Object.entries
String padding
Object.getOwnPropertyDescriptors
Trailing commas in function parameter lists + calls
Async functions
Shared memory and atomics
ES2017
Lifting template literal restriction
S (dotAll) flag for RegExp
RegExp named capture groups
Rest/Spread Properties
RegExp Lookbehind Assertions
RegExp Unicode Property Escapes
Promise.prototype.finally
Asynchronous Iteration
ES2018
Optional catch binding
JSON superset
Symbol.prototype.description
Function.prototype.toString revision
Object.fromEntries
Well-formed JSON.Stringify
String.prototype.{trimStart, trimEnd}
Array.prototype.{flat, flatMap}
ES2019
String.prototype.matchAll
Import()
BigInt
Promise.allSettled
globalThis
For-in mechanics
Optional Chaining
Nullish coalescing Operator
Import.meta
ES2020
String.prototype.replaceAll
Promise.any
WeakRefs
Logical Assignment Operators
Numerical separators
ES2021
Class Fields
RegExp Match Indices
Top-level await
Ergonomic brand checks for Private Fields
.at()
Accessible Object.prototype.hasOwnProperty
Class Static Block
Error Cause
ES2022
Array find from last
Hashbang Grammar
Symbols as WeakMap keys
Change Array by copy
ES2023
?

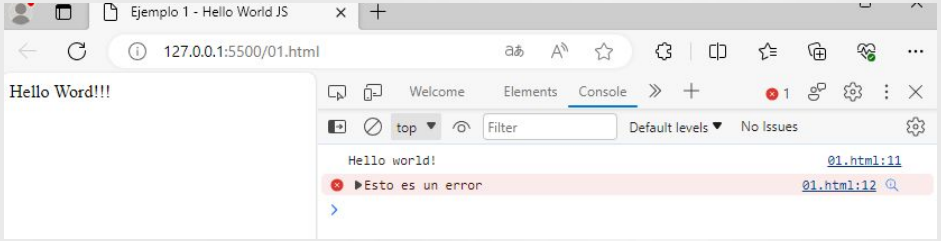

Frontend
1995-2019 EVOLUTION
TWITTER.COM/MANZ
HTML MARKUP
CSS PREPROCESSORS
"FROM" JAVASCRIPT
CSS METHODOLOGIES
CSS FRAMEWORKS
JS SUPERLANGUAGES
MOBILE FRAMEWORKS
JS TOOLS (PREV-GEN)
JS FRAMEWORKS
WEB COMPONENTS
BUILD/BUNDLER TOOLS
1995
2000
2005
2010
2015
2019
HTML ECOSYSTEM
CSS ECOSYSTEM
JAVASCRIPT ECOSYSTEM
CSS
CSS3
FLASH
ACTIONSCRIPT
JS ES1
JS ES5
JS ES6
ES7
ES8
ES9
JS ES10
HTML
XHTML
MARKDOWN
HAML
HANDLEBARS
JADE
PUG
LIT-HTML
Sass
less
stylus
SCSS
POSTCSS
AUTOPREFIXER
BROWSERSLIST
POSTCSS-PRESET-ENV
JSX
JSS
CSS MODULES
STYLED COMPONENTS
EMOTION
ITCSS
TAILWIND
OOCSS
SMACSS
BEM
ACSS
BOOTSTRAP
FOUNDATION
SEMANTIC UI
MATERIALIZE
PURECSS
MILLIGRAM
BULMA
UIKIT
yui
dojo
prototype
Sencha ExtJS
jQuery
mootools
CoffeeScript
ELM
6to5
TS
TYPESCRIPT
BABEL
ES MODULES
UNDERSCORE.JS
Knockout.
BACKBONE
ANGULARJS
ember
LODASH
ionic
REACT
NATIVE SCRIPT
REACT NATIVE
MARKO
PREACT
VUE
HYPERAPP
ANGULAR
SVELTE
AURELIA
VUE NATIVE
WEBCOMPONENTS v0
WEBCOMPONENTS v1
HYBRIDES
LIT-ELEMENT
POLYMER
BROWSERIFY
WEBPACK
GRUNT
Gulp
ROLLUP
PARCEL
npm

Javascript

La consola del navegador

Es la herramienta que más nos va a ayudar a la hora de depurar nuestro código. Abrimos las herramientas para el desarrollador (en Chrome y Firefox pulsando la tecla F12) y vamos a la pestaña **Consola**:

Allí vemos mensajes del navegador como errores y advertencias que genera el código y todos los mensajes que pongamos en el código para ayudarnos a depurarlo (usando los comandos **console.log** y **console.error**).



Además en ella podemos escribir instrucciones Javascript que se ejecutarán mostrando su resultado. También la usaremos para mostrar el valor de nuestras variables y para probar código que, una vez que funcione correctamente, lo copiaremos a nuestro programa.

Depurar:

Siempre depuraremos los programas desde aquí (ponemos puntos de interrupción, vemos el valor de las variables, ...). Ver: Debugging en el navegador

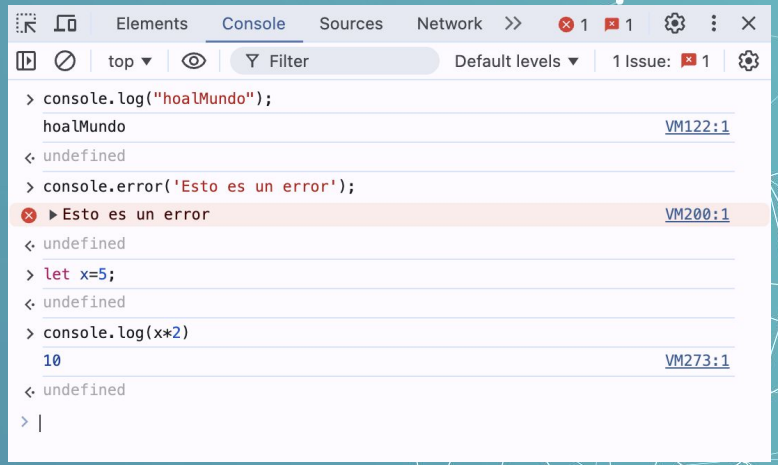
Probando la consola

1. Abre las herramientas de desarrollador en tu navegador (F12 o clic derecho > Inspeccionar).
2. Ve a la pestaña **'Consola'**.
3. Escribe código directamente en la consola para probar y depurar:

console.log('Hola, mundo!');

console.error('Esto es un error');

*let x = 5; console.log(x * 2);*



Ejercicio 1: Identifica los errores

Corrige los errores en el siguiente código usando la consola:

console.log('Hola, mundo!);

let nombre = ‘Manolo’;

console.log('Bienvenida ' + noombre);

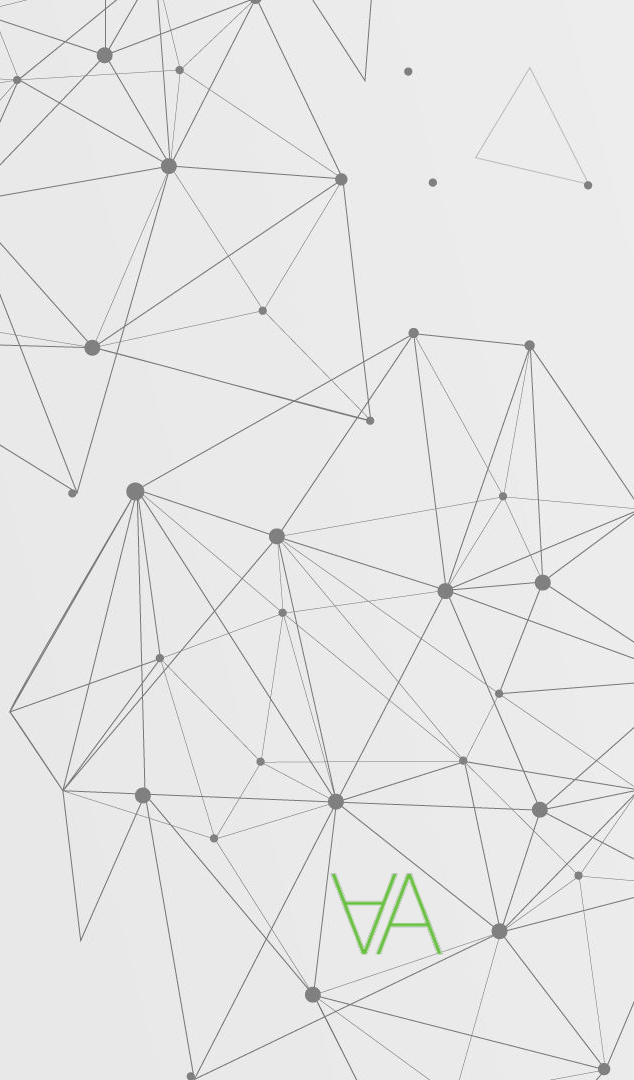
console.log(saludo);

Pistas:

- Observa los mensajes de error en la consola.
- Corrige cada línea y pruébala.

01

Incluir JS en HMTL



Incluir JavaScript en nuestra web

HTML nos ofrece la etiqueta **<script></script>** que será el que nos abra un espacio en el que incluir nuestro código JS.

De la misma manera que hemos hecho con CSS, podemos escribir directamente el código JavaScript en nuestro archivo HTML o bien hacerlo en un fichero ***.js** que luego importamos a nuestra página.

¿Cuál crees que es la mejor opción? ¿Por qué?

defer:

- El script se carga en paralelo al HTML, pero su ejecución se retrasa hasta que todo el HTML ha sido procesado.
- Ideal para scripts que interactúan con el DOM.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">

    <script src="scripts/01-variables.js" defer></script>

    <link rel="stylesheet" text="text/css"
href="css/style.css">
    <title>Aprendiendo JavaScript Moderno</title>
</head>
<body>
    <h1>Aprendiendo JavaScript Moderno</h1>
    <div id="contenedor"></div>
</body>
</html>
```

02

Ventanas modales

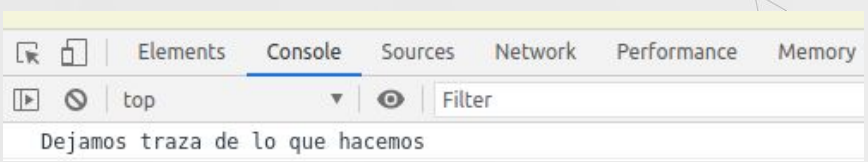
Ventanas modales: alert, confirm, prompt

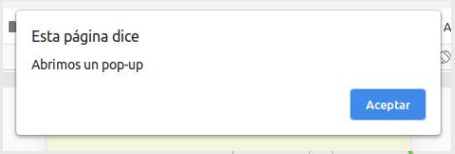
alert(“”); muestra un mensaje con un botón de Aceptar.

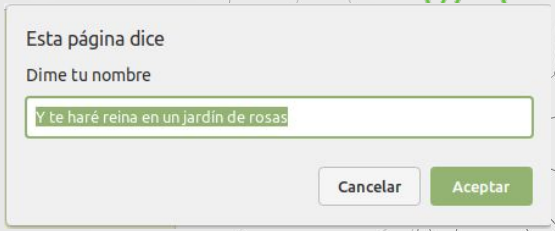
confirm(“”); muestra una ventana con dos botones (Aceptar y Cancelar). Devolverá true si se Acepta, false si se Cancela.

prompt(“”); muestra una ventana con dos botones (Aceptar y Cancelar) y un campo de texto para que el usuario escriba. Devolverá el valor del texto introducido en caso de Aceptar y nada en caso contrario.

```
    <script>
        console.log( "Dejamos traza de lo que
hacemos");
        alert( "Abrimos un modal" );
    </script>
```







Ejercicio 2: Modales

Crea una página que contenga un título “Hola”:

1. Muestre un alert con el mensaje: '¡Bienvenido a mi página!'.
2. Use confirm para preguntar: '¿Te gustaría continuar?'.
3. Si el usuario acepta, pide su nombre con prompt y muestra:
    a. console.log('¡Hola, ' + nombre + '!');

03

Variables

Variables (let, const)

Variables

Las variables son un contenedor de información en que él vamos almacenando información. Deben ser **identificadores únicos** y seguir las siguientes normas:

- Solo pueden contener letras, números, _ y $
- Deben empezar por una letra o por _
- JS es case sensitive (diferencia entre mayúsculas y minúsculas).

Algunas de las prácticas que deberíamos seguir respecto a las variables son:

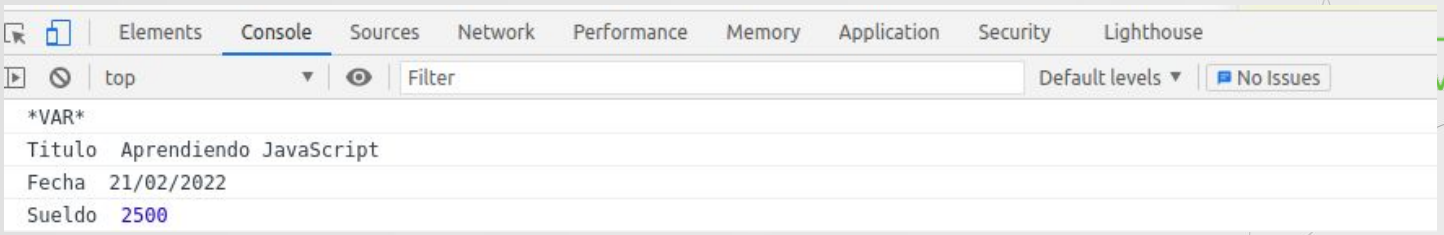
- Elegir un buen nombre es fundamental. Evitar abreviaturas o nombres sin significado (a, b, c, ...)
- Evitar en lo posible variables globales
- Declarar todas las variables al principio
- Inicializar las variables al declararlas
- Usar para nombrarlas la notación camelCase

Variables

En el caso de JS, podemos asignar valores de diferentes tipos a una misma variable y no pasa nada ➜

JS es un **lenguaje no tipado**.

```
// EcmaScript 2021 / ES12 /ES.NEXT
console.log("*VAR*");
var variable="Aprendiendo JavaScript";
console.log("Titulo ", variable);
variable= "21/02/2022";
console.log("Fecha ", variable);
variable= 2500;
console.log("Sueldo ", variable);
```



LET, CONST

Dentro de las variables existen dos tipos:

- **LET (Variables con valor cambiante)**
  - **Ámbito de bloque**: Solo es accesible dentro del bloque `{}` donde se declara.
  - **Reasignable**: Permite actualizar el valor almacenado, pero **no redeclararla** en el mismo ámbito.
  - **Seguridad**: Lanza error si intentas usarla antes de definirla.

- **CONST (Constantes y referencias estables)**
  - **Ámbito de bloque**: Igual que `let`, limitada a sus llaves `{}`.
  - **No reasignable**: El identificador no puede apuntar a un nuevo valor.
  - **Mutabilidad de objetos**: Si es un objeto o array, la "caja" es fija pero el contenido (propiedades o índices) **puede cambiar**.

04

Operadores

Operadores aritméticos

Se aplican a los tipos numéricos y a algunos Strings.

- Suma (+)
- Resta (-)
- Multiplicación (*)
- División (/)
- Módulo (%)
- Incremento de 1 (++)
- Decremento de 1 (- -)

```
let a = 100;
a = ((a + 10 - 15 ) * 10 / 2 ) % 1;

let cad1 = "Hola";
let cad2 = "¿qué tal?";
let cad = cad1 + cad2;
```

Operadores básicos: Asignación

Para asignar un valor usaremos el operador del igual =. Al gual que entros lenguaje podemos usar operaciones básicos y abreviar.

```
let a = 100;
a += 10; // a=a+10;  110
a -= 15; // a=a-15;  95
a *= 10; // a=a*10;  950
a /= 2; // a=a/2;   475
a %= 2; //a=a%2;  1 (el módulo es 1, a es impar)
```

Operadores lógicos

Son operadores que nos permiten comparar el valor de dos o más variables, siendo el resultado un booleano que nos permitirá tomar decisiones. Son operadores que se usan en los condicionales de las estructuras de control y los bucles.

- **igual en valor (==)**
- **igual en valor y en tipo (===)**
- diferente en valor (**!=**)
- diferente en valor o en tipus (**!==**)
- mayor (**>**)
- menor (**<**)
- mayor o igual (**>=**)
- menor o igual (**<=**)
- conjunción lógica (**&&**)
- disjunción lógica (ll)
- negación lógica (**!**)

NOTA

JavaScript proporciona dos operaciones distintas para **comparar** la igualdad de dos elementos:

=== - Igualdad estricta (o "triple igual" o "identidad"). El operador igualdad estricta compara la igualdad de dos valores. Ninguno de estos valores se convierte de manera implícita antes de ser comparado. Si los valores tienen tipos diferentes son considerados diferentes.

== -Igualdad débil o relajada ("doble igual"). El operador igualdad débil compara la igualdad de dos valores después de convertir ambos valores a un tipo de datos común. Tras la conversión , la comparación que se lleva a cabo funciona exactamente como ===.

05

Tipos de datos

TIPOS DE DATOS

```
//Number
let numero = 14;
let decimal = 3.14;
```

```
//String comillas simples o dobles
let saludo = "Hello";
let mundo = 'World';
```

```
//Booleano
let mayorEdad = true;
let menorEdad = Boolean(0);
```

NUMBER

Valor numérico (enteros, decimales, etc.)

STRING

Valor de texto (cadenas, caracteres, etc.)

BOOLEAN

Valor booleano (true o false)

```
//Null, vacio
let vacio = null;
```

```
//Undefined
let noDefinido;
```

typeof(nombreVariable)
Nos permite saber de qué tipo de dato es la variable.

NULL

Es como un objeto vacío que puede ser asignado a una variable.

UNDEFINED

Valor sin definir, variable sin inicializar.

FUNCTION*

Función guardada en una variable.

OBJECT*

Estructura compleja compuesta con más variables.

NUMBER

Sólo hay 1 tipo de números, no diferencia entre enteros y decimales. El tipo de dato para cualquier número es **`number`**. El carácter para la coma decimal es el **. (punto)**

Algunos **métodos** útiles de los números son:

- **toFixed(num)**: redondea el número a los decimales indicados. `23.2376.toFixed(2) devuelve 23.24`
- **toLocaleString()**: devuelve el número convertido al formato local. `23.76.toLocaleString() devuelve '23,76' (convierte el punto decimal en coma)`
- **isNaN(valor)**: nos dice si el valor pasado es un número (false) o no (true)
- **isFinite(valor)**: devuelve true si el valor es finito (no es Infinity ni -Infinity).
- **parseInt(valor)**: convierte el valor pasado a un número entero. Siempre que empiece por un número la conversión se podrá hacer.

```
parseInt(3.65)       // Devuelve 3
parseInt('3.65')     // Devuelve 3

parseInt('3 manzanas')     // Devuelve 3
```

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

Ejercicios de Numbers

1. Haz una función que de dos números pasados por parámetros indica (saca por consola) cúal de ellos es mayor.
2. Haz una función que calcule (sin usar *) la multiplicación de dos números.
3. Utilizando la función anterior haz una función que calcule la multiplicación de 3 números.
4. Crea una función que permita calcular la nota media de 3 notas para que devuelva la media con 1 decimal.
5. Encuentra entre los primeros 10.000 números naturales los números que completan la siguiente propiedad: La suma del cubo de cada una de sus cifras y que dé el mismo número:

   → por ejemplo 153: 1*1*1+5*5*5+3*3*3= $1^3+5^3+3^3=153$.

6. (optativo) Crea una función que devuelva el cubo (numero^3) de un número pero que compruebe si el parámetro pasado es un número entero. Si no es un entero o no es un número mostrará un alert indicando cuál es el problema y devolverá false.

STRING

Las cadenas de texto van entre comillas simples o dobles, es indiferente. Para forzar la conversión a cadena se usa la función **String(valor)**

Algunos **métodos** y **propiedades** de las cadenas son:

.length: devuelve la longitud de una cadena.
.charAt(posición): devuelve el carácter en esa posición
.indexOf(carácter): devuelve el número de la posición o -1 si no lo encuentra
.lastIndexOf(carácter): devuelve la posición del último carácter.
.substring(desde, hasta): devuelve una subcadena
.substr(desde, num caracteres):devuelve una subcadena
.replace(busco, reemplaza): Devuelve el string con el replace hecho.
.toLowerCase(): devuelve string en minúsculas.
.toUpperCase(): devuelve string en mayúsculas.
.localeCompare(cadena): devuelve -1 si la cadena a que se aplica el método es anterior alfabéticamente a 'cadena', 1 si es posterior y 0 si ambas son iguales. Tiene en cuenta caracteres locales como acentos ñ, ç, etc.

STRINGS

.trim(cadena): quita el exceso de espacios
.startsWith(cadena): devuelve true si el string empieza con la cadena.
.endsWith(cadena): devuelve true si el string acaba con la cadena.
.includes(cadena): devuelve true si el string contiene con la cadena.
.repeat(veces): devuelve una cadena con el string repetido N veces..
.split(separador): devuelve un array de cadenas/caracteres según separador.

Nota: Todos estos métodos NO modifican el String original, es decir que el contenido de la variable no se ve alterado, solo retorna lo que le pedimos.

Un String es una cadena de caracteres que podemos concatenar (**+**).

```
let nombre = "Alicia";
let apellidos = "Vazquez";
let profesion = "Profesora de informática"
let nya = nombre + " " + apellidos + "/n" + profesion;
console.log(nya);
```

```
let plantilla = `Template:
Hola soy ${nombre} ${apellidos}
y soy ${profesion}`;
console.log(plantilla);
```

Ejercicios de Strings

Haz una **función** para cada uno de los problemas propuestos.

1. Comprueba si un string contiene el substring 'As' al principio. Si lo contiene, devuelve el string original. Sino, devuelve el string con 'As' concatenado al principio de este.

```
obtenStringEmpiezaConAs('ASTerisco') → Asterisco

obtenStringEmpiezaConAs('TEroide') → Asteroide

obtenStringEmpiezaConAs('') → ''
```

2. Introduce un string y un número, comprueba que es menor que la longitud del string en una función. La función debe devolver el string sin el carácter en esa posición.

```
eliminaCarracterPosicion('murcielago',3) → 'mucielago'

eliminaCarracterPosicion('rinoceronte',12) → 'rinoceronte'
```

3. Dado un string intercambia la posición del primer y último carácter del string. Tienes que comprobar que el string tiene longitud mayor que 0.

BOOLEAN

Los valores booleanos son **true** y **false**. Para convertir algo a booleano se puede usar **Boolean(valor)** aunque también puede hacerse con la doble negación (!!). Cualquier valor se evaluará a true excepto 0, NaN, null, undefined o una cadena vacía ('') que se evaluarán a false.

Los operadores lógicos son ! (negación), && (and), || (or). También tenemos 2 operadores de diferente: != y !== que se comportan como hemos dicho antes.

```
'3' == 3 --> true
3 == 3.0 --> true
0 == false --> true
'' == false --> true
' ' == false --> true
[] == false --> true
null == false --> false
undefined == false --> false
undefined == null --> true
undefined === null --> false
```

OJO: Los operadores relacionales son >, >=, <, <=. Cuando se compara un número y una cadena ésta se convierte a número y no al revés (23 > '5' devuelve true, aunque '23' > '5' devuelve false)

DATE

El tipo de dato **Date** nos permite manipular fechas, pero hemos de conocerlo bien sino es fácil equivocarse.

```
// Obtenemos la fecha actual y la guardamos en la variable fecha
let fecha = new Date();

// Obtenemos la fecha 22 de Febrero de 2022, a las 22h20min22s
fecha = new Date("2022/02/22 22:20:22");

// Creamos una fecha pasando cada uno de sus componentes numéricos
// y, m son obligatorios, el resto no.
//new Date(y, m, d, h, min, s, ms);
fecha=new Date(2022, 03, 15);
```

DATE

Una vez hemos creado una fecha y tenemos el objeto , podemos obtener todas sus variables a través de sus métodos. Los **getters** son:

- `getDay()` Devuelve el día de la semana (0 Domingo, 6 Sábado).
- `getFullYear()` Devuelve el año con 4 cifras.
- `getMonth()` Devuelve la representación interna del mes (0 Enero - 11 Diciembre).
- `getDate()` Devuelve el día del mes.
- `getHours()` Devuelve la hora (en 24h)
- `getMinutes()` Devuelve los minutos.
- `getSeconds()` Devuelve los segundos.
- `getMilliseconds()` Devuelve los milisegundos.

Y lo mismo con los **setters:**

- `setFullYear(year)`
- `setMonth(month)`
- `setDate(day)`
- `setHour(hour)`
- `setMinutes(min)`
- `setSeconds(sec)`
- `setMilliseconds(ms)`

DATE

El ***timestamp***, es un formato numérico utilizado para calcular una fecha en UNIX. Es una forma poco práctica y legible para humanos, pero muy eficiente en términos informáticos. Se trata de un número que representa la cantidad de segundos transcurridos desde la fecha 1/1/1970, a las 00:00:00.

Así pues, siendo números, resulta muy fácil trabajar y operar con ellos. Una fecha **A** y una fecha posterior **B**, si hacemos **B - A** nos devuelve el número de segundos transcurridos entre ambas fechas, con lo que se podría sacar la diferencia de tiempo.

```
A = new Date(2001, 05, 04);
B = new Date(2022, 03, 15);
let edadS = B.getTime() - A.getTime();
console.log(edadS);
```

06

Estructuras de control

Condicionales (if, else), bucles (for, while).

Estructura condicional: IF

Las primeras estructuras de control son el **If** y el **if/else** que nos permiten decirle al navegador que realice A o B, según la condición.

```
//IF
let nota = 7;
console.log("He realizado mi examen.");

// Condición (si nota es mayor o igual a 5)
if (nota >= 5) {
 console.log("¡Estoy aprobado!");
}
```

```
let nota = 7;
console.log("He realizado mi examen. Mi
resultado es el siguiente:");

// Condición
if (nota < 5) {
 // Acción A (nota es menor que 5)
 calificacion = "suspendido";
}
if (nota >= 5) {
 // Acción B (nota es mayor o igual que 5)
 calificacion = "aprobado";
}

console.log("Estoy", calificacion);
```

Ejercicio: Dividir vocales de consonantes

Programa la función `separarVocalesDeConsonantes()` que partiendo de una cadena de entrada, la trabaje y retorne una cadena que tenga todas las vocales y luego las constantes (en el mismo orden en el que aparecen).

Antes de ponerte a “picar”, escribe en el mismo fichero que vas a hacer la codificación cómo vas a resolver el problema (en lenguaje humano). Piensa cuando vas a saber si tu programa funciona o no, define bajo qué circunstancias vas a saber que todo está ok.

Escribe las llamadas que vas a hacer y los resultados que esperas.

Estructura condicional: IF/ELSE

```
let nota = 7;
console.log("He realizado mi examen. Mi resultado es el siguiente:");

// Condición
if (nota < 5) {
 // Acción A (nota es menor que 5)
 calificacion = "suspendido";
} else {
 // Acción B: Cualquier otro caso a A (nota es mayor o igual que 5)
 calificacion = "aprobado";
}
console.log("Estoy", calificacion);
```

```
let nota = 7;
console.log("He realizado mi examen.");

// Condición
if (nota < 5) {
 calificacion = "Insuficiente";
} else if (nota < 6) {
 calificación = "Suficiente";
} else if (nota < 8) {
 calificacion = "Bien";
} else if (nota <= 9) {
 calificacion = "Notable";
} else {
 calificacion = "Sobresaliente";
}

console.log("He obtenido un", calificacion);
```

Estructura condicional: Operador ternario

condicion ? cierto : falso

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

Con estructuras if else

```
let nota = 7;
console.log("He realizado mi examen.
Mi resultado es el siguiente:" );

if (nota < 5) {
  calificacion = "suspendido";
} else {
  calificacion = "aprobado";
}


  console.log("Estoy",
  calificacion);
```

Con el operador ternario

```
let nota = 7;
console.log("He realizado mi examen. Mi resultado es el
siguiente:");

// Operador ternario: (condición ? verdadero : falso)
let calificacion = nota < 5 ? "suspendido" : "aprobado";


  console.log("Estoy", calificación);
```

Estructura condicional: SWITCH

Si lo que necesitamos es evaluar posibles valores de una variable, entonces usaremos **switch().**

Pero la traducción no es directa y hay pequeños matices a tener en cuenta respecto al ejemplo de los if/else.

- Valores vs Rangos: Hay que poner todos los casos.
- **Break:** Hay que poner break al final de cada bloque para que no continúe evaluando:

```
var nota = 3;
console.log("He realizado mi examen y ");
// Nota: Este ejemplo NO es equivalente al ejemplo anterior (leer abajo)
switch (nota) {
 case 10:
   calificacion = "Sobresaliente";
   break;
 case 9:
 case 8:
// tanto para 9 como para 8 entra aquí, luego con break no sigue evaluando.
   calificacion = "Notable";
   break;
 case 7:
 case 6:
   calificacion = "Bien";
   break;
 case 5:
   calificacion = "Suficiente";
   break;
 case 4:
 case 3:
 case 2:
 case 1:
 case 0:
   calificacion = "Insuficiente";
   break;
 default:
   // Cualquier otro caso
   calificacion = "Nota errónea";
   break;
}
console.log("he obtenido un: ", calificacion);
```

Estructura condicional: SWITCH

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

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

BUCLES

Conceptos básicos

Condición: Existe una condición que se evalúa para saber si se debe repetir el bucle o finalizarlo. Generalmente, si la condición es verdadera, se repite. Si es falsa, se finaliza.

Iteración: Cada repetición de un bucle se denomina iteración.

Contador: Algunos bucles tienen una variable que cuenta el número de repeticiones que ha hecho, y finaliza al llegar al límite indicado. Dicha variable hay que inicializarla (crearla y darle un valor) antes de comenzar el bucle.

Incremento: Si hay un contador, en cada iteración deberemos incrementar o decrementar el contador.

Bucle infinito: Es un bucle que se queda eternamente repitiéndose puesto que la condición siempre se cumple al no cambiar de estado las variables que intervienen (ej. contador) y el programa se queda «colgado».

WHILE

While, es el bucle más simple que hay. Si se cumple la condición se ejecuta, en caso contrario saldrá del bucle y dejará su ejecución.

En el caso de while debe ser el programador el que incrementa el contador o se encargue de modificar el estado de la condición.

```
i = 0; // Inicialización de la variable contador

// Condición: Mientras la variable contador sea menor de 5
while (i < 5) {
 console.log("Valor de i:", i);

 i = i + 1; // Incrementamos el valor de i
}



i = 0;
bucle = true;

// Condición: Mientras sea cierto
while (bucle) {
   console.log("Valor de i:", i);
   i = i + 1; // Incrementamos el valor de i
   if (i >= 5) {
       bucle = false;
   }
}
```

DO…WHILE

Do…While, es el bucle que se ejecuta mínimo una vez, puesto que la condición está al final del bucle.

```
let i = 1; // Inicialización de la variable contador

// Condición: Mientras la variable contador sea menor de 5
do{
      console.log("Valor de i:", i);
      i = i + 1; // Incrementamos el valor de i
}while (i < 5)
```

FOR

For, es el bucle más utilizado en el mundo de la programación. En el caso de usar un contador es el más fácil de usar, puesto que en la apertura del **for**, inicializamos el contador, especificamos la condición y especificamos en incremento, todo a la vez.

También podemos tener más de un contador a la vez en el mismo bucle.

```
// for (inicialización; condición; incremento)
for (i = 0; i < 5; i++) {
   console.log("Valor de i:", i);
}

// doble contador
for (i = 0, j = 5; i < 5; i++, j--) {
   console.log("Valor de i y j:", i, j);
}
```

Ejercicios de condicionales

1. **Números primos.**

   Encuentra e imprime todos los números **primos** dentro del rango [inicio, fin]. Un número es primo si solo es divisible entre 1 y sí mismo, es decir que no es divisible por ningún número inferior a él entre dos.

   Nota: Busca información referente a **Math.sqr()**

2. **Número secreto**

   Genera un número secreto aleatorio entre 1 y 100 usando **Math.random().**

   Pide al usuario que adivine el número. Y tendrás que ir dándo pistas: indicando si el número del usuario es mayor, menor o bien ha acertado. Al final informa al usuario cuantos intentos ha hecho.