

JavaScript desde cero

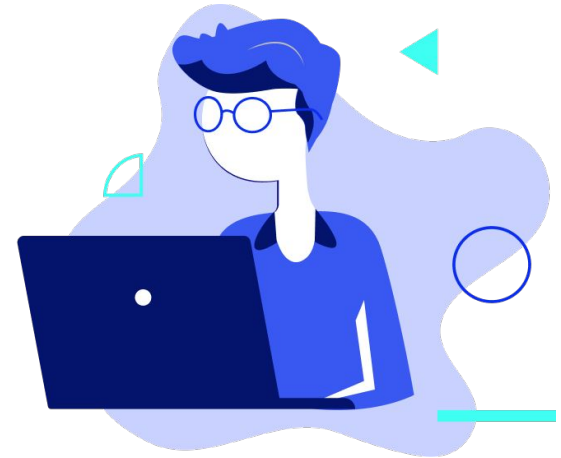
Módulo 1

Introducción

Aprender a programar

Un programa es un **conjunto de instrucciones** que ejecuta un procesador de computadora. Todo programa tendrá un conjunto finito de instrucciones, que **se van ejecutando 1 a 1 en cadena, hasta finalizar la ejecución.**

Para empezar a trabajar con un programa es importante entender que **es un plan** que se va a llevar a cabo.



Pseudocódigo

- El *pseudocódigo* **omite detalles** que quizás se vayan a trabajar, en una siguiente etapa, con el lenguaje de programación elegido. La razón de esta omisión es que esos detalles no son esenciales para comprender en sí **de qué se trata el programa y cuál es su fin**.
- Es útil para planificar correctamente **aquello que se llevará a cabo en el lenguaje de programación que se decida implementar**.
- **Se basa en convenciones de cualquier lenguaje de programación**, pero lo hace entendible para las personas. Además, es **independientemente de cualquier lenguaje específico de programación**.



Veamos un ejemplo en el siguiente slide.

Pseudocódigo

Inicio

// Declarar las variables para almacenar números a sumar

Entero num1, num2

// Solicitar el ingreso del primer número

Mostrar "Ingresa el primer número:"**Leer** num1

// Solicitar el ingreso del segundo número

Mostrar "Ingresa el segundo número:"**Leer** num2

// Calcular la suma de ambos números

Entero resultado

resultado = num1 + num2

// Mostrar el resultado

Mostrar "El resultado de la suma es:", resultado**Fin**

Aplicación: ¿Cómo reconocerla?

Una aplicación es un **programa específico que resuelve un problema concreto.**

Por ejemplo: aplicaciones contables, de gestión de RR.HH, aplicaciones de liquidación de sueldos.

Una de las características principales de las aplicaciones es la **interacción directa con el usuario.**

Un sistema está formado por un **conjunto de programas, involucra también herramientas de hardware** (partes físicas: monitores, teclados, impresoras, etc.), **redes de comunicación, bases de datos, servidores.**




Tipos de aplicaciones

Aplicaciones de escritorio

Son aquellas que típicamente corren en un **sistema Windows** y pueden ser abiertas yendo a la lista de programas instalados en el sistema operativo. **Trabajan con ventanas, tienen un menú** en la parte superior (con opciones tales como: *archivos, herramientas, configuración, y otros*). Estas aplicaciones permiten ingresar datos, obtener reportes de datos, etc. Existe mucha **interacción con el teclado y el mouse** de la computadora.

El botón **secundario del mouse** suele generar el conocido menú contextual, muy útil ya que representa un atajo para la ejecución de una funcionalidad específica.

Podemos decir que las aplicaciones de escritorio son las “**aplicaciones tradicionales**” de **interfaz gráfica**.



Aplicaciones de consola

Son las que utilizan una ventana de **Línea de comandos** o **Terminal**, como **salida**.

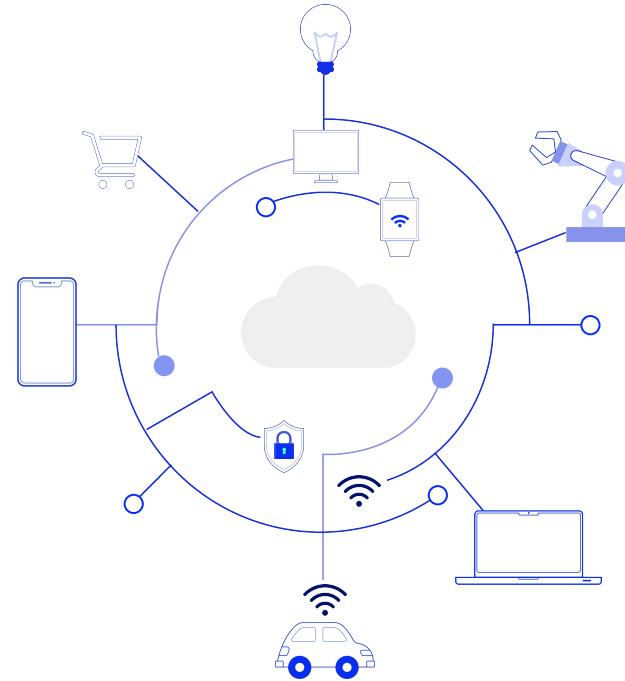
Quizás, esta definición no ayude demasiado a entender de qué se trata, pero básicamente **Visual Basic.Net**, **C#**, **Java**, y **Node.js**, utilizan este tipo de consola para poder **programar** y **crear aplicaciones de escritorio**.



Aplicaciones web

Se accede a ellas desde un *web browser* (**Edge**, **Firefox**, **Chrome**, y otros), a través de alguna dirección **web** o **url**.

Los lenguajes de programación propios de la web han invadido diferentes espacios, y no necesariamente se utilizan para crear páginas o sitios web. Las empresas pueden requerir aplicaciones web para manejar cuestiones internas porque brindan la posibilidad de **acceder desde cualquier dispositivo mientras haya conexión a internet**.



Aplicaciones Mobile

Se trata nada más y nada menos de las famosas **apps**. Funcionan sobre dispositivos *mobile* (*tablets*, celulares, etc.).

Corren en sistemas operativos móviles como **Android**, **iOS** y **iPadOS**.

Resuelven de forma más simple gestiones y operaciones del usuario.

Por ejemplo: las apps de **Mercado Libre**, **Instagram**, **Uber**, **Rappi**, **Cabify** seguramente son las que más utilizas en tu teléfono o dispositivo móvil.



Aplicaciones de *software*

Además de las diferentes interfaces informáticas que existen actualmente, hay varios tipos de aplicaciones de *software* que se pueden crear.

Estas pueden segmentarse en los siguientes grupos:

Aplicaciones compiladas

Se concentran específicamente en aplicaciones distribuidas como archivos ejecutables.

Ejemplos: apps instalables, apps móviles, videojuegos.

Aplicaciones interpretadas

Las *aplicaciones web* son un buen ejemplo de aplicaciones interpretadas.

Necesitamos un intérprete intermedio (el navegador web), para que estas se ejecuten.

Aplicaciones transpiladas

Apps creadas con un *framework* o librería intermedia, que se deben traducir a un lenguaje intermedio para que luego sean ejecutadas como una aplicación compilada o interpretada.

Ejemplos: React, Angular, Vue, PHP, .NET, Electron.

Lenguajes de programación

Se trata de un lenguaje formal con **reglas estrictas de escritura**, que permite comunicarle a una computadora qué es lo que debe hacer con absoluto detalle.

Todo lenguaje de programación se conforma por un conjunto de símbolos, signos de puntuación, operadores, valores, palabras clave e identificadores que permiten escribir las instrucciones a ejecutar.

A través de los lenguajes de programación es posible **crear programas**.

Existen docenas de lenguajes de programación, muchos son similares entre sí, también tienen algunas diferencias.

Lo más importante es comprender cómo JavaScript (y todo lo que deriva de este lenguaje de programación) es, hoy, el centro del universo tecnológico. Es impensado, casi en todos los ámbitos, que un programador, aunque conozca y maneje otro lenguaje, no sepa JavaScript. Vamos a contarte por qué.

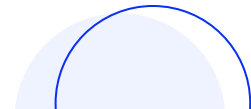
JavaScript como lenguaje de programación

JavaScript es el centro de todo porque es un lenguaje de programación **interpretado por el propio navegador** (*Chrome, Firefox, Opera, IE, y otros*), sin necesidad de nada más en absoluto.

La web domina el mundo de la tecnología, desde la creación de interfaces para fábricas de autos, cajeros automáticos, o simplemente aplicaciones para lograr que los empleados puedan desde cualquier lugar donde hay conexión a Internet resolver cualquier problema laboral o trabajar sin necesidad de moverse. **Esto reduce costos y mejora el rendimiento.**

Entre las tantas cosas que podemos hacer con JavaScript se pueden mencionar:

- **Abrir cuadros de diálogo.**
- **Mostrar mensajes.**
- **Validar datos en un formulario.**
- **Hacer una galería de imágenes.**



Implementar JS en línea

Los documentos HTML cuentan con una **etiqueta** `<script>`, que permite definir código JS.

Si bien se puede implementar en pruebas de código rápidas, desde 1999 **se recomienda crear y escribir código JS en archivos dedicados, y nunca en un documento HTML**. Esto hace más fácil el mantenimiento del código JS, utilizado en múltiples documentos HTML.

Vemos un ejemplo de la implementación de JS desde el archivo HTML en el siguiente slide.



```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>JS desde Cero</title>
</head>
<body>
  <h1>Título de primer nivel</h1>
  <p>Esto es un párrafo de texto, poco extenso.</p>
  <script>
    // Dentro de estas etiquetas podríamos escribir código JS
  </script>
</body>
</html>
```

Implementar JS de manera externa

Para crear un archivo JavaScript, antes de comenzar a escribir la lógica de las aplicaciones, se debe **contar con un documento HTML**.

Los archivos JS se referencian dentro del encabezado **<head>** de los documentos HTML.

El atributo **defer**, que acompaña al archivo JS, solo es necesario incluirlo cuando interactuemos desde JS con elementos HTML.

Siempre **conviene crear subcarpetas dedicadas, para mantener ordenado los diferentes tipos de archivos** que conforman a un proyecto.

Veamos un ejemplo en el siguiente slide.

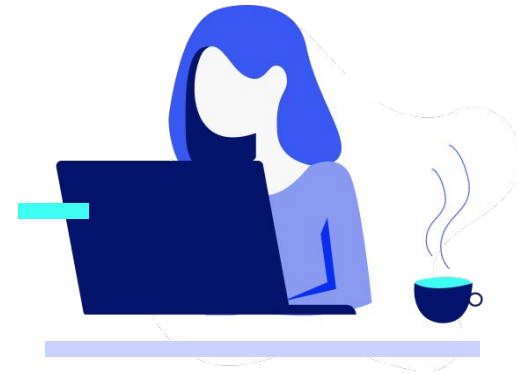



```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>JS desde Cero</title>
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <script defer src="js/main.js"></script>
</head>
<body>
  <h1>Título de primer nivel</h1>
  <p>Esto es un párrafo de texto, poco extenso.</p>
</body>
</html>
```

El atributo **defer** se terminó de soportar, en todos los navegadores web, en el año 2014, con el objetivo de **permitir declarar los archivos JS dentro del apartado <head> de un documento HTML.**

Si bien, todavía es posible declarar un archivo JS, al final de un documento HTML, **es importante referenciarlo dentro del apartado <head> cuando trabajamos con sitios web que requieren un manejo de SEO efectivo.**

De hacerlo erróneamente, perjudicaremos el posicionamiento de estos sitios web en los resultados de búsqueda de: Google, Bing, y otros tantos buscadores conocidos.



Realizar comentarios en el código

Al igual que casi todos los lenguajes de programación, **JS permite dejar comentarios de texto intercalados con el código.**

Estos comentarios se pueden crear como de **una sola línea //**.

Si se debe escribir un contenido más extenso, es posible recurrir a la combinación de caracteres que permite escribir **múltiples líneas de comentarios /* */**.

```
index.js

// Comentario de una sola línea en archivos JS

/*
Si debemos escribir más de una línea de comentario en
los archivos JS, podemos recurrir a esta combinación
de caracteres.
*/
```



Los comentarios sirven para **hacer anotaciones** que permitan entender, en una lectura rápida, **qué es lo que realiza determinado algoritmo.**

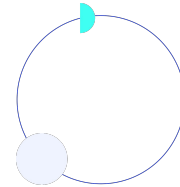
Estos caracteres dedicados, también son útiles para **comentar código que no deseamos que se ejecute.**

El código del ejemplo de la derecha, no se podrá ejecutar porque fue comentado.

```
index.js
// let variableA = "Soy una variable"

/*
  const userName = "EducaciónIT"

  console.log("Usuario:", userName)
*/
```



Herramientas integradas de JS

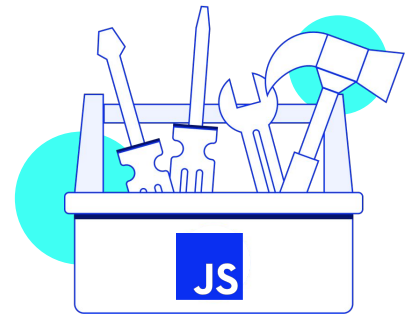
Herramientas integradas de JS

Para programar aplicaciones, JS ofrece una serie de herramientas integradas que ayudan a analizar y trabajar de manera más eficiente.

Las herramientas se dividen en **dos categorías**:

- **cuadros de diálogo.**
- **herramientas de consola.**

Su uso principal es poder analizar y depurar nuestras aplicaciones.



Primer script: cuadro de diálogo `alert()`

Este cuadro de diálogo, alerta al usuario sobre diferentes situaciones. Aunque, actualmente, los cuadros de diálogo fueron reemplazados, en su gran mayoría, por **ventanas modales más vistosas**.

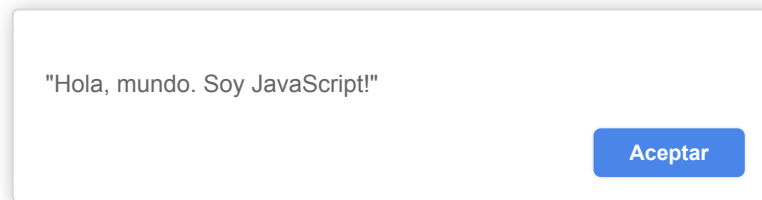
La realidad es que nos ayudarán, a nosotros desarrolladores, a aprender la **sintaxis de JS** y poder depurar y analizar el comportamiento de los algoritmos que escribiremos a continuación.

En tu **codigo.js** escribe lo siguiente:

A screenshot of a code editor window titled 'index.js'. The code inside is `alert("Hola, mundo. Soy JavaScript!");`. The editor has a dark background and a light blue border on the right side.

```
index.js
alert("Hola, mundo. Soy JavaScript!");
```

El resultado será el siguiente:



Uso del punto y coma

JavaScript es un lenguaje de programación que evoluciona todos los años, ininterrumpidamente, desde el año 2015. En la versión conocida como **EcmaScript v6**, lanzada ese mismo año, transformó a **opcional el uso del punto y coma “;” para finalizar cada sentencia JS.**

Está en nosotros usarlo o no. En algunos casos dará claridad en la lectura de la sintaxis, delimitando dónde comienza y dónde termina. Pero si ya programamos en otros lenguajes de programación donde no se utiliza el punto y coma, podremos obviarlo para así sentirnos más cómodos.

```
index.js
//Punto y coma, no!
alert("Hola, mundo. Soy JavaScript!")

//Punto y coma, sí!
alert("Hola, mundo. Soy JavaScript!");
```



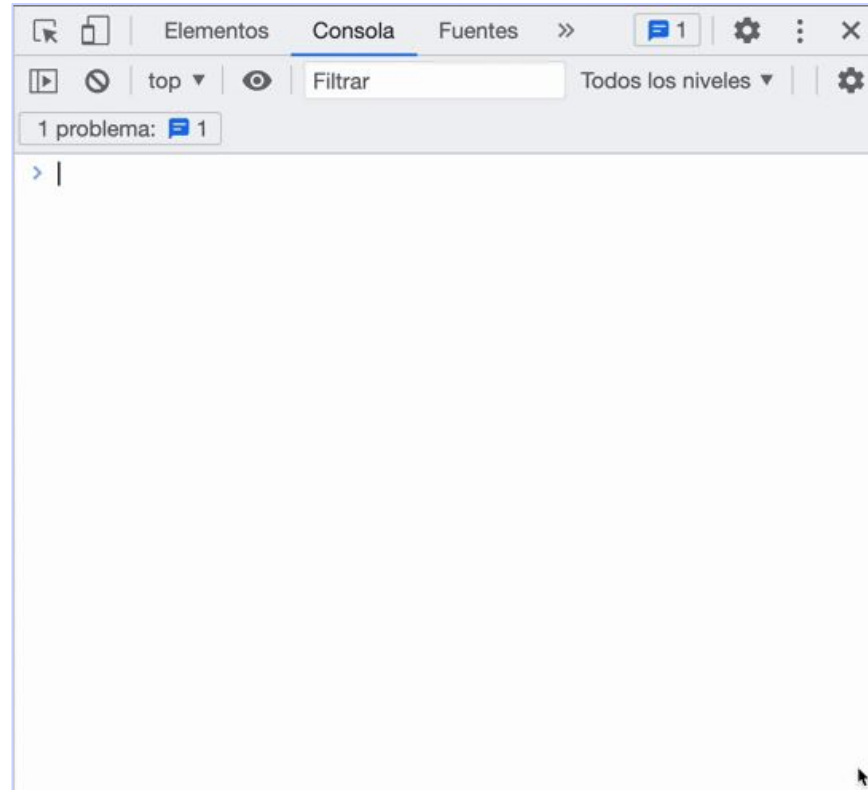
El objeto JS: *console*

JavaScript console es un objeto, nativo de este lenguaje, que cuenta con una serie de métodos incorporados que permiten sacar provecho al momento de analizar el código o comportamiento de estas aplicaciones.

La mayoría de sus métodos arrojan resultados en la **pestaña *Console*** de las ***Herramientas para el Desarrollador (DevTools)***.

En las próximas *slides*, veremos estos métodos:

- `console.log()`.
- `console.warn()`.
- `console.error()`.
- `console.table()`.



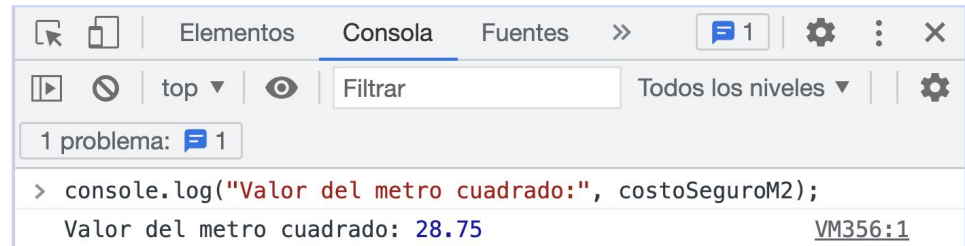
`console.log()`

El método `console.log()` permite **visualizar un mensaje definido**, en la Consola JS.

El texto puede provenir de un texto definido manualmente por nosotros, o desde el valor contenido en una variable o constante.

```
Console.log()

console.log("Este texto está predeterminado", miVariable);
```



`console.warn()`

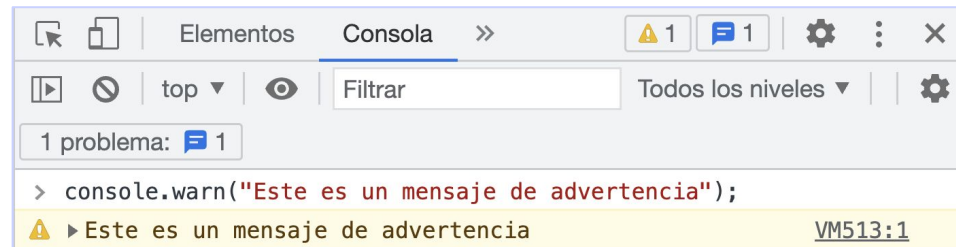
El método `console.warn()` **visualiza también un mensaje** en la Consola JS con un **énfasis de color** refiriendo a un tipo de **mensaje de advertencia**.

A su vez, incorpora el **ícono de alerta**, para que le prestemos más atención al mismo al identificarlo en la consola JS.



```
Console.warn()

console.warn("Este es un mensaje de advertencia");
```



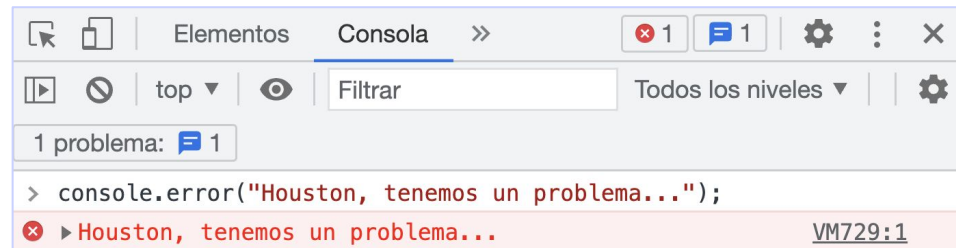
`console.error()`

El método `console.error()` visualiza también un mensaje en la Consola JS, pero con un **énfasis de color** refiriendo a un tipo de mensaje de error.



```
Console.error()

console.error("Houston, tenemos un problema...");
```

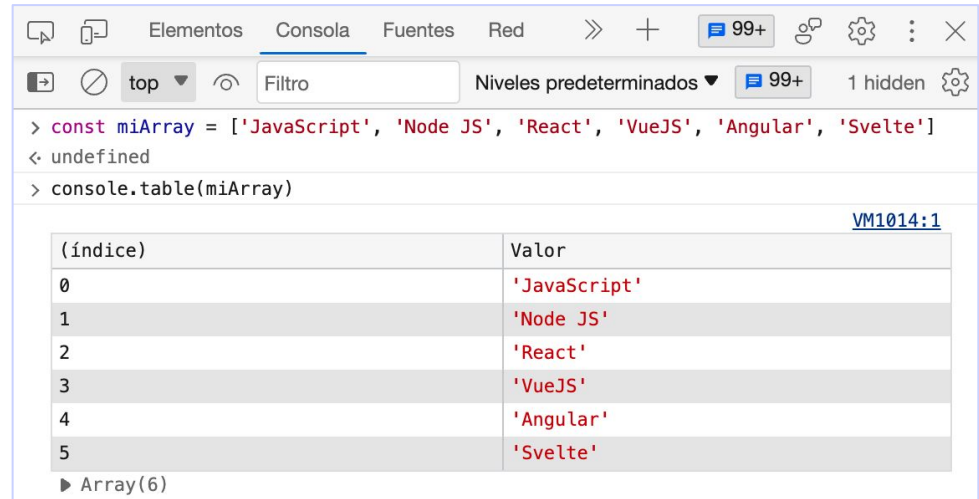


`console.table()`

El método `console.table()` es de gran utilidad para **representar en pantalla datos tabulares**.

Por ejemplo, cuando se trabaja con arrays de elementos, *arrays* de objetos, u objetos literales, que contienen múltiples datos, son mejor representados por este método que por `console.log()`.

Más adelante veremos ejemplos concretos sobre cómo implementarlo de manera efectiva.



```
> const miArray = ['JavaScript', 'Node JS', 'React', 'VueJS', 'Angular', 'Svelte']
< undefined
> console.table(miArray)
```

(índice)	Valor
0	'JavaScript'
1	'Node JS'
2	'React'
3	'VueJS'
4	'Angular'
5	'Svelte'

▶ Array(6)



Revisión

- Repasar los conceptos básicos de un **lenguaje de programación**.
- Trabajar con **pseudocódigos** para empezar a adaptarse a esta lógica.
- Implementar un **cuadro de diálogo alert()** y **utilizar console.log()**, de forma interna y externa.
- Aplicar todas las propiedades en el **proyecto integrador**.
- Realizar todas las preguntas necesarias antes de continuar.



**¡Sigamos
trabajando!**