

# Introducción a la Programación en Javascript

## Aprender a programar

Podemos definir un programa como un conjunto de instrucciones que ejecuta un procesador de computadora. Todo programa tendrá un conjunto finito de instrucciones, las cuales se van ejecutando 1 a 1 en cadena hasta finalizar la ejecución.

Para empezar a trabajar con un programa es importante, entender que un programa es un plan que se va a llevar a cabo. Nada mejor que planificar cómo será ese programa a través de herramientas.

### Pseudocódigo

Se base en convenciones de un lenguaje de programación cualquiera pero lo hace **entendible** para las personas, y aparte lo realiza independientemente de cualquier lenguaje específico de programación.

De hecho el **pseudocódigo** omite detalles que quizás se vayan a trabajar con el lenguaje de programación elegido, pero porque estas no son esenciales para que comprendamos en sí de qué se trata y cuál es el fin del mismo.

Si bien es un elemento que hallaremos en cualquier libro de texto o en internet, pero en definitiva, cada vez que empieces o pienses realizar un programa es ideal que realices un **pseudocódigo** con la intención de planificar correctamente aquello que llevarás a cabo en el lenguaje de programación que decidas implementar.

```
algoritmo Sumar

variables
  entero a, b, c

inicio
  escribir( "Introduzca el primer número (entero): ")
  leer( a )
  escribir( "Introduzca el segundo número (entero): ")
  leer( b )
  c ← a + b
  escribir( "La suma es: ", c )
fin
```

## Aplicación

Una aplicación es un programa específico, que resuelve un problema concreto. A menudo hablamos de aplicaciones contables, aplicaciones de gestión de RR.HH, aplicaciones de liquidación de sueldos, etc.

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 hardware (partes físicas: monitores, teclados, impresoras, etc), redes de comunicación, bases de datos, servidores, etc.

Cuando hablamos de sistema informático también involucramos a la parte HUMANA (usuarios finales, personal de soporte, programadores, etc).

## Tipos de aplicaciones

### Aplicaciones de escritorio

Son aquellas que típicamente corren en un sistema Windows; las cuales pueden ser abiertas yendo a la lista de programas instalados en el sistema operativo. Dichas aplicaciones trabajan con ventanas, tienen un menú en la parte superior (Con opciones tales como: archivos, herramientas, configuración, etc).

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 nos 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 aquellas aplicaciones que utilizan una ventana de MS-DOS como salida. Quizás el definirlo de esta manera no te ayude demasiado a entender de que se trata, pero básicamente Visual Basic.Net y C# quienes utilizan este tipo de consola para poder programar y crear aplicaciones de escritorio.

## **Aplicaciones Web**

Son aquellas que son accedidas desde un browser (Internet Explorer, Firefox, Chrome, etc) a través de alguna dirección web o url. El lenguaje web ha invadido diferentes espacios, y por esa razón no necesariamente esto es o puede ser un sitio web, sino qué empresas pueden requerir aplicaciones web para manejar cuestiones internas por el mero hecho de la facilidad que estas permiten de acceder desde cualquier lugar mientras haya conexión a internet.

## **Aplicaciones Mobile**

Aquellas que funcionan sobre dispositivos mobile (tablets, celulares, etc). Se trata nada más y nada menos de las famosas "apps". Corren en sistemas operativos móviles como Android, iOS, BlackBerry OS, Windows Phone, etc.

Resuelven de forma más simple gestiones y operaciones del usuario. Por ejemplo la app de mercado libre, o la de instagram seguramente son aquellas que más utilices en tu teléfono o dispositivo móvil.

Si bien es importante entender que todos los ejemplos anteriores pueden hacer complejo el mundo de las aplicaciones o entender qué es un programa, la realidad es que no es más que una sucesión de pasos ordenados y planificados que resuelven un problema específico.

# **Lenguajes de Programación**

Se trata de un lenguaje formal, con reglas estrictas de escritura, el cual permite comunicarle a una computadora que 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 podemos crear nuestros programas.

Existen docenas y docenas de lenguajes de programación hoy día, muchos con similitudes entre sí, como también así con sus diferencias, pero lo más importante es entender como Javascript y todo lo que este lenguaje de programación hoy deriva es el centro del universo tecnológico. Es impensado casi en todos los ámbitos que un programador aunque este sea o maneje otro lenguaje específicamente no sepa Javascript, y vamos a contarte por qué.

## Javascript, el centro de todo

Javascript, es el centro de todo porque es un lenguaje de programación interpretado por el propio navegador (Chrome, Firefox, Opera, IE, etc), sin necesidad de absolutamente nada más. 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 movilizarse, lo cual reduce costos y mejora el rendimiento.

### **Entre las tantas cosas qué podemos hacer con Javascript están:**

- Abrir ventanas
- Mostrar mensajes
- Validar datos en un formulario
- Hacer una galería de imágenes
- Añadir dinámicamente nueva información dentro del cuerpo de la página web.
- Crear juegos
- Crear animaciones

## Implementar Javascript

### **Interna**

Lo haremos a través de la etiqueta script, simplemente ubicamos la misma tanto en el head como en el body de cualquier documento html y comenzaremos a trabajar con nuestro código.

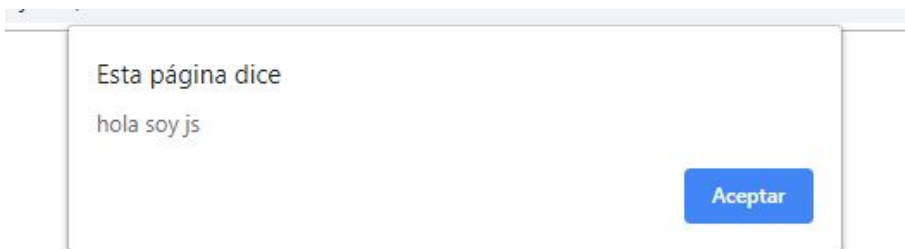
```
<script>
alert('hola soy js')
</script>
```

## En línea o semántica

La idea es hacerlo dentro de las propias etiquetas de HTML, por ejemplo en el ejemplo siguiente la misma alerta anterior se dispara al momento de levantar la página en el navegador.

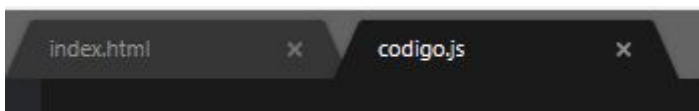
```
<body onload="alert('hola soy js')">
```

En ambos casos el resultado será el siguiente:

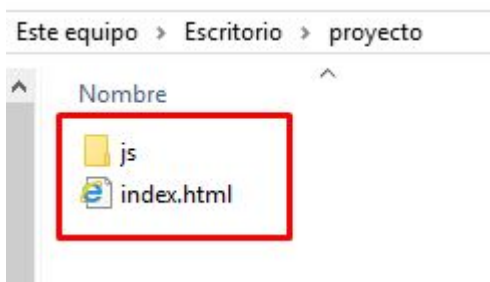


## Externa

Esta forma, es la ideal pues trabajaremos con un archivo externo de .js , por ejemplo guardaremos los siguientes elementos para trabajar:



Para , lograr un proceso más ordenado te recomendamos generar una carpeta js, como en la siguiente imagen:



Dentro de este archivo.js, lo que haremos será empezar a trabajar con nuestro lenguaje de programación. Pero para poder hacerlo debemos vincularlo con tu HTML

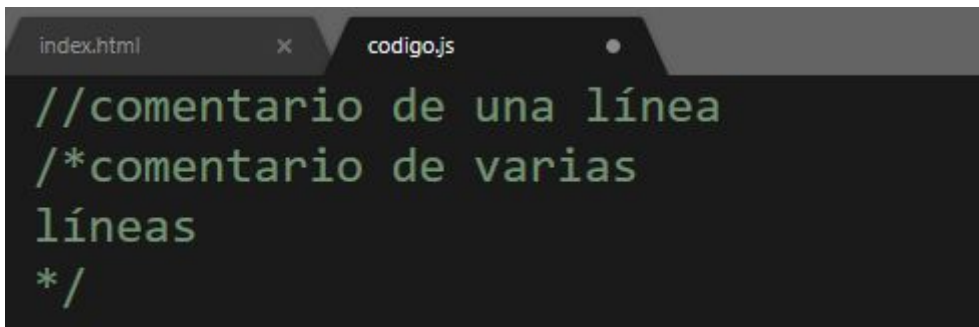
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Desarrollo</title>
  <script src="js/codigo.js"> </script>
</head>

<body>

</body>
</html>
```

## Aprendiendo la sintaxis de JS

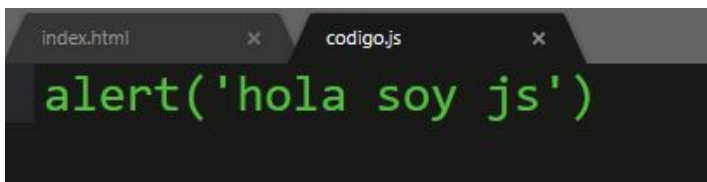
Javascript se maneja a través de sentencias que le dicen al navegador (Chrome, Firefox, etc) qué hacer. En base a eso, es interesante también saber qué tenemos la posibilidad de hacer comentarios. Estos no afectan a nuestro programa y nos permiten hacer qué un código deje de ejecutarse o dejar mensajes que facilitan el desarrollo o la comunicación entre quienes estamos trabajando en una determinada aplicación

A screenshot of a code editor with two tabs: 'index.html' and 'codigo.js'. The 'codigo.js' tab is active, showing the following JavaScript comments in green text on a dark background:

```
//comentario de una línea
/*comentario de varias
líneas
*/
```

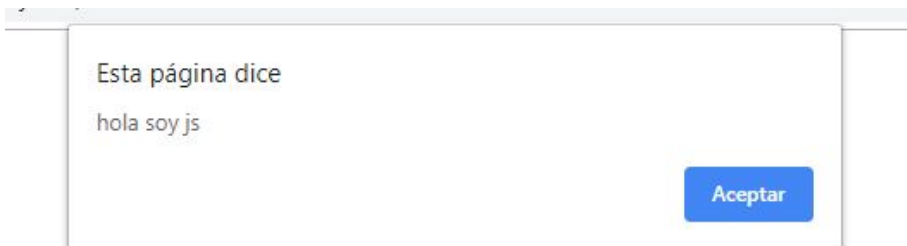
## Primer Script: Ventana de Alerta

Esta ventana alerta al usuario sobre diferentes situaciones, si bien hoy en día las ventanas de alertas fueron reemplazadas en su gran mayoría por ventanas modales más vistosas , la realidad es que no ayudarán a aprender la sintaxis de JS y son la base de todo lo que aprenderás más adelante. En tu codigo.js escribirás lo siguiente:

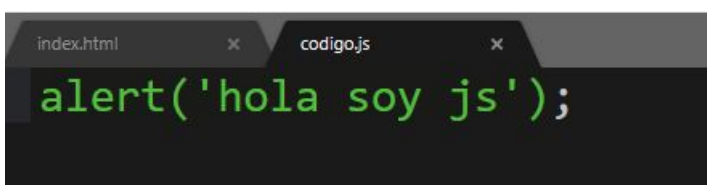
A screenshot of a code editor with two tabs: 'index.html' and 'codigo.js'. The 'codigo.js' tab is active, showing the following JavaScript code in green text on a dark background:

```
alert('hola soy js')
```

El resultado será el siguiente:



Si bien no es obligatorio se recomienda el uso de ; (punto y coma) al final de cada sentencia para poder así evitar errores y separarla de las otras. En otros lenguajes de programación como PHP , estos separados son obligatorios, pero en este caso no es necesario, de todas formas para realizar un código más ordenado y prolijo terminaremos nuestra sentencia así:

A screenshot of a code editor with two tabs: 'index.html' and 'codigo.js'. The 'codigo.js' tab is active, showing the following JavaScript code in green text on a dark background:

```
alert('hola soy js');
```


# Variables

Las variables en los lenguajes de programación en general , no solamente en JS, tienen la misma lógica que en una operación matemática.

Una variable es un elemento que se emplea para almacenar, guardar en un cajón la información y **utilizarla nuevamente (por eso el nombre de variable)**.

La verdad es que sin variables cualquier programa es inútil y sin sentido por eso es interesante conocerlas.

Las variables se crean o definen mediante la palabra reservada **var**.




```
index.html x  codigo.js x
var numero = 2;
```

## Reglas que no debemos olvidar al nombre una variable

- Los nombres de variables puede contener letras, números, \_ (underscore) y signo de dólar (\$)
- No se puede comenzar con un número
- Los nombres son **caseSensitive** (esto significa que no es lo mismo A qué a)
- Hay palabras reservadas que no las podés usar para nombres de variables, no te preocupes a medida que vayamos avanzado vas a saber cuales son

Te quiero contar qué algo que usualmente hacemos en JS es trabajar con camelCase, por ejemplo si yo quiero generar una variable que tiene un nombre complejo para luego poder identificarla lo haremos con esta regla:



```
index.html x  codigo.js x
var primerVariable;
```

## Tipos de Variables

- **String** : Son fundamentalmente cadenas de texto, se escriben entre comillas dobles o comillas simples. Vamos a probarlo en nuestro codigo.js , por ejemplo:



```
index.html x código.js x
var nombre = "Carlos";
```

Por supuesto, que nada de esto tiene sentido si yo no puedo mostrar la información, por lo tanto, vamos a mostrar el nombre del empleado a través de una ventana de alerta

```
var nombre = "Carlos";
alert(nombre)
```

Cuándo se muestra el dato, no es necesario como en el primer ejemplo ponerlo entre comillas ya que la variable es un objeto y no requiere de las mismas.

- **Numéricas** : Cualquier expresión numérica, esta puede ser float (decimal) o entera. En nuestro archivo, trabajaremos de la siguiente manera:

```
index.html x código.js x
var edad = 45;
alert(edad)
```

Si queremos distinguir decimales de enteros lo haremos de la siguiente forma:

```
index.html x código.js x
var iva = 21; // variable tipo entero
var total = 9234.65; // variable tipo decimal
```

**También existen otras variables que más adelante profundizaremos tales como:**

- **Boolean** : Expresiones booleanas TRUE ó FALSE.
- **Undefined** : Toda variable declarada sin valor o cualquier propiedad interna no existente de un objeto
- **Object** : Vector asociativo en n dimensiones. El mismo se inicializa de manera literal con {}(llaves) y contiene en su interior pares de índices asociados a valores por el operador : (dos puntos) separados por , (coma). Ej.: {nombre:"Educacion IT"}. Un objeto puede contener cualquier tipo de dato en su interior.
- **Array** : Objeto especializado en poder tener además de su comportamiento habitual la habilidad de guardar datos de manera secuencial, es decir bajo índices numéricos auto incrementales. Los mismos se inicializan de manera literal con [](corchetes) y contienen en su interior cualquier tipo de dato separado por , (coma). Ej.: [1,2,"Educacion IT"]

---

Estas variables serán vistas en las clases siguientes con mayor detalles y puestas en práctica.


---

## Output en Javascript

Hay variadas formas de mostrar nuestra información en JS, por ejemplo, hemos visto como a través de una ventana de alerta podemos hacerlo. Sin embargo existen otras maneras de interactuar con nuestro HTML, por ejemplo:

### document.write()

Esta nos permite escribir directamente en nuestro documento, para lograr tal fin, generamos las siguientes líneas de código en nuestro archivo.js

A screenshot of a code editor with two tabs: 'index.html' and 'codigo.js'. The 'codigo.js' tab is active, showing the code `document.write('Hola estoy dentro de mi documento')` in a dark-themed editor with syntax highlighting.

El resultado será el siguiente,

Hola estoy dentro de mi documento

Si queremos generar un mejor formato, podemos implementar etiquetas de HTML, por ejemplo, nuestra intención es que este texto sea un título, lo haremos de la siguiente forma:

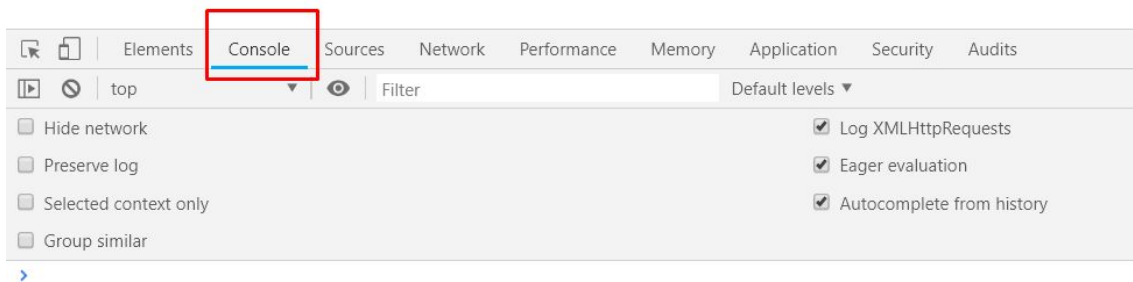
```
document.write('<h1>Hola estoy dentro de mi documento</h1>')
```

Podemos también implementar , varias líneas dentro de nuestro archivo con diferentes etiquetas válidas de HTML, por ejemplo:

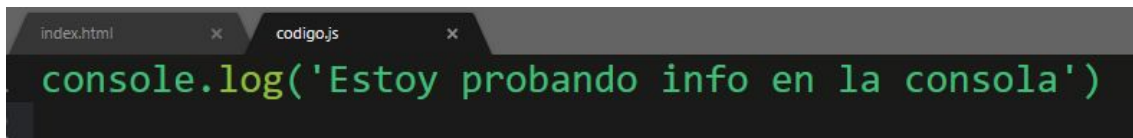
```
document.write('<h1>Hola estoy dentro de mi documento</h1>')
document.write('<p>Este es un párrafo con<strong>negrita</strong></p>')
```

## console.log()

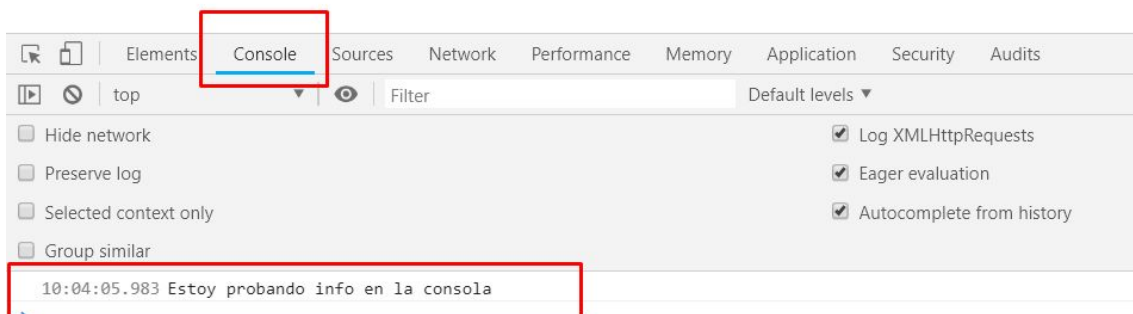
Esta forma de generar contenido se hace a través de la consola, a la cual podemos acceder presionando la tecla f12 desde en nuestro navegador.



La consola, nos permite conocer errores, datos y demás cuestiones sumamente importantes para el trabajo con JS, pero puntualmente en este caso la utilizaremos para generar información de la siguiente manera



El resultado será el siguiente,



# Operadores

Como en su momento habíamos establecido qué poco podíamos hacer en JS, si no contamos con variables, los operadores nos brindan herramientas para generar que nuestra aplicación o programa cumpla su fin.

Los operadores permiten trabajar con variables, realizar operaciones matemáticas y compararlas. Nos permiten avanzar en nuestro programa y cumplir el fin que nos hayamos propuesto.

## Asignación

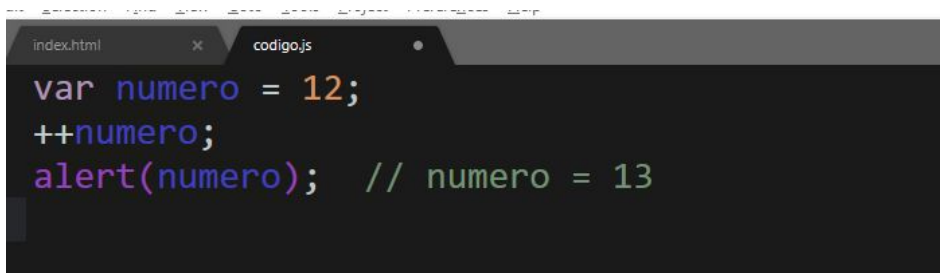
Este tipo de operador, es simplemente aquel se utiliza para guardar un valor en una variable.

Ya lo hemos utilizado previamente, pero veamos un ejemplo:

```
var iva = 21;
```


## Incremento y decremento

El operador de incremento se indica mediante el prefijo ++, incrementa la variable en una unidad, el operador de decremento, por el contrario, la decrementa en una unidad



```
index.html x código.js
var numero = 12;
++numero;
alert(numero); // numero = 13
```

Con el operador de decremento, el ejemplo sería el siguiente

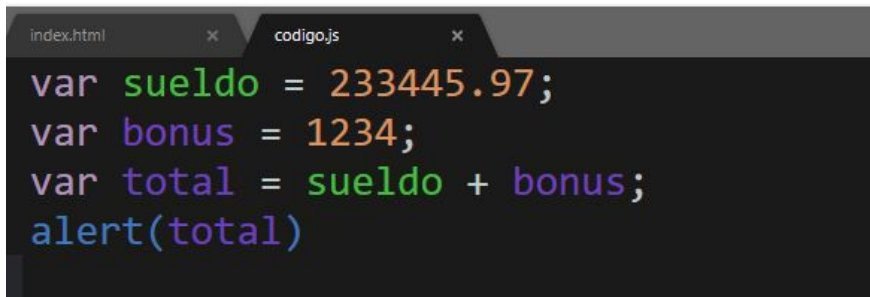


```
index.html x código.js x
var numero = 12;
--numero;
alert(numero); // numero = 11
```

En este momento, quizás este operador no cobre tanto sentido, pero en clases posteriores, verás como él mismo es utilizado para programas más complejos que involucran arreglos e iteración.

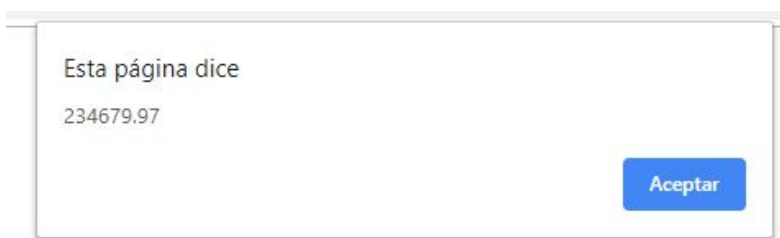
## Matemáticos

JavaScript permite hacer operaciones matemáticas con los operadores ya conocidos por todos como : suma (+), resta (-), multiplicación (\*) y división (/)

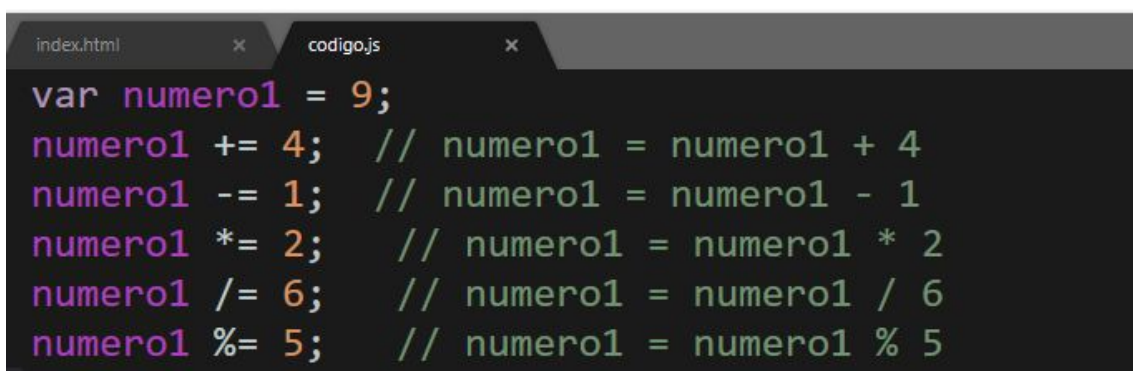


```
index.html x  codigo.js x
var sueldo = 233445.97;
var bonus = 1234;
var total = sueldo + bonus;
alert(total)
```

El resultado en nuestro navegador será el siguiente:



Por otro lado, el operador matemático puede combinarse con el de asignación para lograr el siguiente resultado más rápido y conciso



```
index.html x  codigo.js x
var numero1 = 9;
numero1 += 4; // numero1 = numero1 + 4
numero1 -= 1; // numero1 = numero1 - 1
numero1 *= 2; // numero1 = numero1 * 2
numero1 /= 6; // numero1 = numero1 / 6
numero1 %= 5; // numero1 = numero1 % 5
```

También con el (+) podemos concatenar texto para obtener resultados para accesibles y entendibles para el usuario, por ejemplo:

```
var sueldo = 233445.97;
var bonus = 1234;
var total = sueldo + bonus;
alert("El total del sueldo y el bonus es" + total)
```

Es decir podemos concatenar información para lograr mejores resultados para el usuario, también incluyendo etiquetas de HTML, por caso

```
var sueldo = 233445.97;
var bonus = 1234;
var total = sueldo + bonus;
document.write("<p>El total del sueldo y el <strong>bonus</strong> es " + total + "</p>")
```

La última línea fue reemplazada por document.write(), ya que las ventanas de alertas no soportan formato, es decir que una etiqueta en una ventana de alerta se vería de la siguiente forma

```
alert('<p> Esta etiqueta, se ve de verdad acá </p>')
```



Un detalle interesante, es que seguramente te llamó la atención el operador de módulo (%), ya que no es utilizado en matemática. Este operador se utiliza para calcular el resto de la división entre 2 valores enteros. Tiene la misma precedencia que el producto y el cociente.

## Lógicos

Previamente habíamos visto qué existían variables booleanas (true/false), estos operadores siempre dan como resultado que algo sea verdadero o falso

- **NEGACIÓN**, se utiliza para obtener el contrario o la negación de la variable

```
index.html x codigo.js
var dato = 2;
var vacio = !dato;
alert(vacio) //resultado false
```

- **AND**, La operación obtiene un resultado donde ambos valores deben ser verdaderos

```
index.html x codigo.js
var valorA = true;
var valorB = false;
resultado = valor1 && valor2; // resultado = false

valorA = true;
valorB = true;
resultado = valor1 && valor2; // resultado = true
```

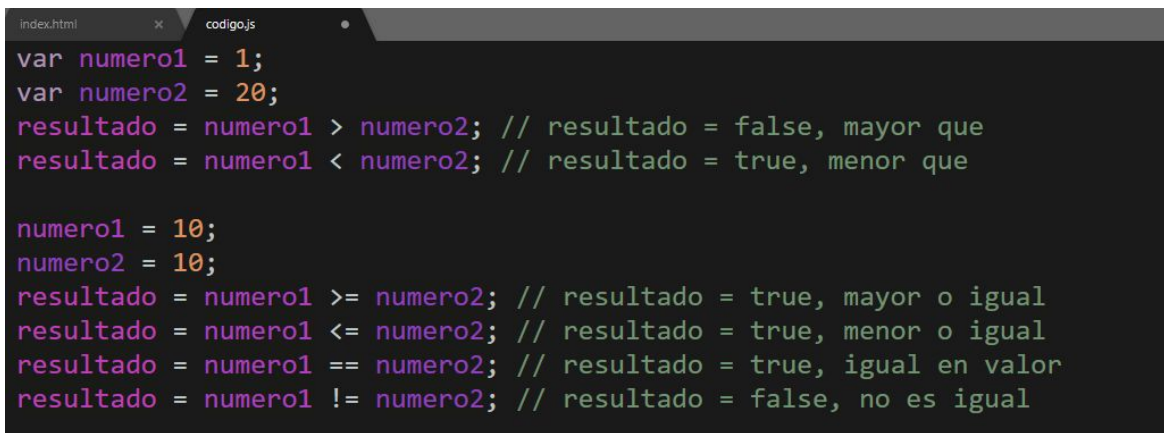
- **OR**, en este caso alguno de los dos puede no ser verdadero, ya que la consigna es que uno de ellos lo sea para obtener un resultado true

```
index.html x codigo.js x
var valorA = true;
var valorB = false;
resultado = valor1 || valor2; // resultado = true

valorA = false;
valorB = false;
resultado = valor1 || valor2; // resultado = false
```

## Relacionales

Los operadores relacionales son iguales a los que usamos en matemáticas, nos permite realizar programas complejos, y comparar datos, el resultado será también al igual que los operadores lógicos true/false



```
index.html x codigo.js
var numero1 = 1;
var numero2 = 20;
resultado = numero1 > numero2; // resultado = false, mayor que
resultado = numero1 < numero2; // resultado = true, menor que

numero1 = 10;
numero2 = 10;
resultado = numero1 >= numero2; // resultado = true, mayor o igual
resultado = numero1 <= numero2; // resultado = true, menor o igual
resultado = numero1 == numero2; // resultado = true, igual en valor
resultado = numero1 != numero2; // resultado = false, no es igual
```

---

El caso anterior está representado por operadores que utilizaremos más adelante cuando veamos estructuras de control de flujo.

---

## Errores

Dentro de nuestro código es normal que cometamos errores, estos pueden ser de diferentes tipos y lo importante es aprender a detectarlos y corregirlos en consecuencia

### Errores de Sintaxis

Podemos comparar un error sintáctico con un error “ortográfico”, propio del lenguaje natural de las personas. ¿Qué consecuencias trae un error sintáctico? En lenguajes compilados: Un error sintáctico NO permitirá que el programa se compile: Debemos corregir dichos errores para compilar el programa y luego poder ejecutarlo.

### Errores en Ejecución

Un programa puede estar perfectamente bien escrito, libre de errores de sintaxis, pero puede cometer errores durante su ejecución. Estos errores ocurren en “tiempo de ejecución” (“runtime”: Intervalo de tiempo que va desde que el programa inicia su ejecución hasta que finaliza). Este tipo de errores son producidos por acciones / operaciones imposibles de realizar (incompatibilidad entre tipos de dato y operadores u operaciones sin solución).

Ejemplos:

- Una división por cero.
- Cálculo de la raíz cuadrada de un número negativo.
- Bucles infinitos.
- Que el programa intente hacer un cálculo aritmético con valores de tipo string.



Los errores en tiempo de ejecución muchas veces DETIENEN nuestro programa.

A menudo usamos la expresión: “el programa se rompe”, “el programa pincha” cuando esto ocurre. Una causa frecuente a este tipo de problemas se debe a datos ingresados por el usuario; los cuales no han sido bien validados (Ejemplo: Se esperaba que el usuario ingrese un valor numérico, cuando en realidad ingresó una cadena de texto).

## Errores Lógicos

Son aquellos relacionados con acciones inesperadas que comete nuestro programa.

Ejemplos:

- Que el programa realice una suma aritmética cuando en realidad debió haber restado.
- Que el programa borre datos de la aplicación en lugar de agregar datos nuevos.
- Que el programa no arroje mensajes en pantalla cuando debió haberlo hecho.
- Etc.

Estos errores son los peores con los que nos podemos encontrar ya que requerirá una revisión en la lógica de alguna funcionalidad en particular o bien requerirá una revisión de toda la aplicación. Estos errores muchas veces están relacionados con el pseudo-código o los diagramas de flujo que se armaron previamente al código real del programa. (La lógica de acción y ejecución del programa previamente pensada resulta que era EQUIVOCADA). Sea cual sea el tipo de error, siempre será deber del programador corregirlos.

El trabajo de buscar errores en nuestros programas se conoce como TESTING

## Debugger

Un debugger (depurador) es una aplicación complementaria al lenguaje de programación con el cual estemos trabajando, el cual nos ayuda a encontrar y solucionar posibles errores en el código de nuestro programa.

Las herramientas de desarrollador de los navegadores son un buen complemento para detectar estos de forma más rápida por ejemplo,

Chrome: <https://developers.google.com/web/tools/chrome-devtools/javascript/?hl=es>

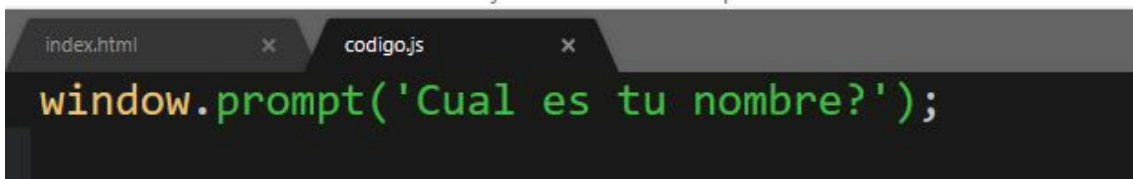
IE: <https://msdn.microsoft.com/es-es/communitydocs/web-dev/ie11/ie11-el-depurador>

# Introducción a Ventanas

Las ventanas de usuario, como la ventana de alerta que hemos visto anteriormente nos permiten brindar información, pedir un dato o confirmar una decisión para ejecutar luego una serie de pasos.

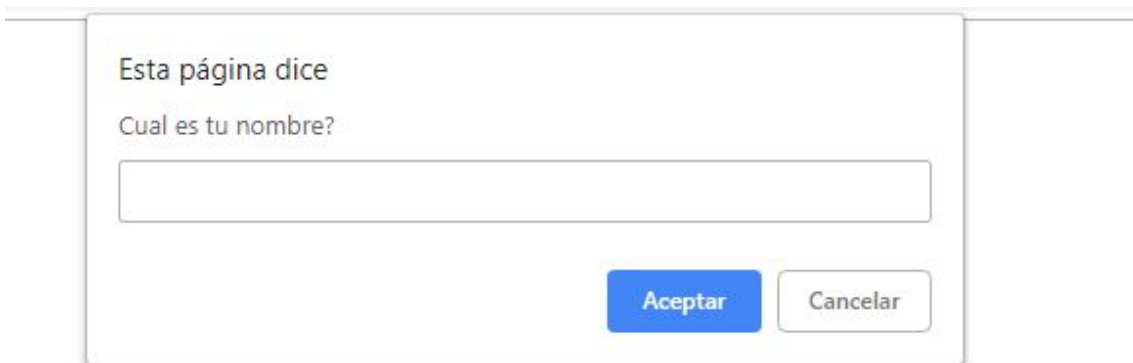
## Ventana Prompt o de ingreso de datos

Esta ventana permite ingresar dato, por ejemplo en nuestro código haremos lo siguiente



```
index.html x  codigo.js x
window.prompt('Cual es tu nombre?');
```

El resultado en nuestro navegador será el siguiente

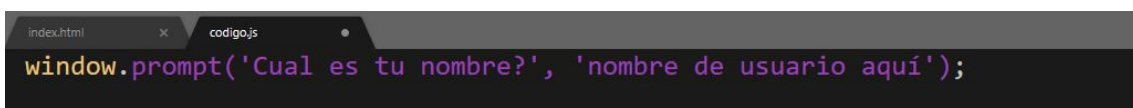


Esta página dice

Cual es tu nombre?

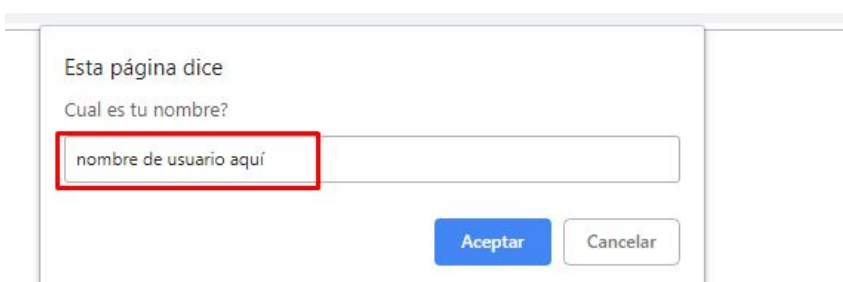
Aceptar Cancelar

Si queremos agregar un valor predeterminado, podremos hacerlo de la siguiente forma



```
index.html x  codigo.js
window.prompt('Cual es tu nombre?', 'nombre de usuario aquí');
```

El resultado será



Esta página dice

Cual es tu nombre?

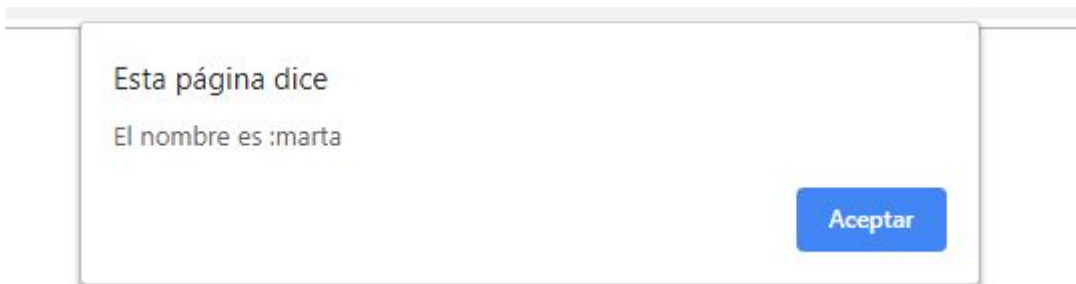
Aceptar Cancelar

Esa información obtenida, también podemos mostrarla, por ejemplo utilizando variables, de la siguiente manera

```
index.html x código.js
var nombre = window.prompt('Cual es tu nombre?', 'nombre de usuario aquí');

alert('El nombre es :' + nombre);
```

El resultado en el navegador será el siguiente,

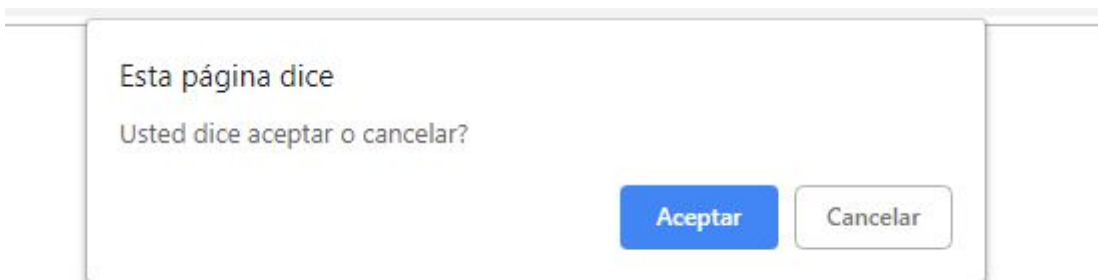


## Ventana de Confirmación

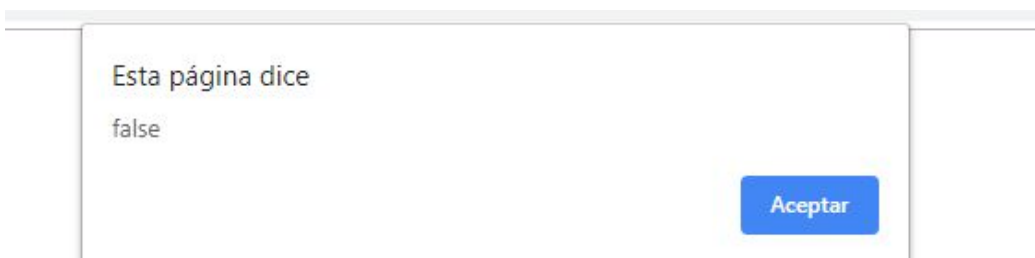
La ventana de confirmación almacena un valor boolean true/false,

```
index.html x código.js
var confirmacion = window.confirm('Usted dice aceptar o cancelar?');
```

Se mostrará en el navegador lo siguiente,



En este caso mostraremos el resultado de la variable confirmación, si el usuario cancela,



---

Este tipo de ventanas, se resignifican al momento de trabajar con ventanas de control de flujo en la clase siguiente

---