



*JavaScript: fundamentos*

**¿Qué es JavaScript?**

JavaScript es un lenguaje de programación que se utiliza principalmente para construir aplicaciones y sitios web dinámicos. Con esta tecnología, podemos desarrollar softwares como los que empleamos a diario: tiendas online, buscadores, gestores de contenido, entre otros.

Se dice que es un lenguaje interpretado porque el código que escribimos se puede ejecutar directamente en el navegador, lo cual significa que los archivos HTML con código JavaScript asociado se interpretan inmediatamente.

JavaScript se utiliza para crear el “cerebro” de la aplicación, lo que permite dotar a tecnologías como HTML y CSS de la interactividad y el dinamismo necesarios para construir soluciones web modernas.

**Aplicación Web**

La necesidad de usar JavaScript aumenta con la cantidad de funcionalidades que buscamos construir en un software. Es importante tener en cuenta que en la actualidad el desarrollo desde cero de sitios web estáticos, es decir, aquellos cuya información no cambia en respuesta a las acciones del usuario, es poco frecuente.

Lo que se busca construir en el ámbito laboral son plataformas que ofrezcan un alto nivel de interactividad, y un variado número de funcionalidades. Ya no hablamos de sitios, sino de aplicaciones web que permiten realizar tareas importantes a sus usuarios.

Por esta razón, JavaScript se vuelve cada día más importante, ya que es la tecnología más empleada para construir el funcionamiento de “la parte de la aplicación que las personas usan”.

**Front End y Back End**

Con “la parte de la aplicación que las personas usan” nos referimos a todo lo que vemos en el navegador cuando empleamos una plataforma, y el código necesario para que la interfaz funcione. La misma está construida con HTML, CSS y JavaScript, y se la conoce como el Front End.

Por otro lado, toda aplicación web tiene otra sección con la cual el usuario no interactúa directamente, pero es la responsable de dar respuesta a acciones que se realizan en el Front End, almacenando la información que el usuario produce, o solicitando datos para que la aplicación pueda funcionar correctamente. A esta sección se la conoce como Back End, y la misma puede presentar diversidad en las tecnologías con las que se construye, ya que esta parte de las aplicaciones puede integrar varios lenguajes de programación además de JavaScript.

No obstante, como estamos iniciando en el desarrollo de aplicaciones modernas, a lo largo del curso vamos a centrarnos en la construcción del Front End con JavaScript, ya que es necesario entender primero cómo desarrollar nuestra aplicación en respuesta a las acciones del usuario, para luego poder enfocarnos en cómo debe responder el Back End.

En resumen, una aplicación web tiene dos caras: su Front y su Back. Como frontenders, nos dedicamos a desarrollar la parte de la aplicación que corre en el navegador y con la que interactúa el usuario, pero a la vez orquestamos y sincronizamos los pedidos y envíos de información al backend, trabajando siempre en relación con este. Con el desarrollo y evolución de las APIs, hay que saber que no nos comunicamos sólo con ***un*** backend, sino que podemos estar consumiendo muchos servicios externos a la vez.

**Herramientas**

Durante el curso, emplearemos distintas herramientas para construir nuestra aplicación, y podrás escoger las que te sean de mayor utilidad. Principalmente trabajaremos con el navegador y el editor de código, y a ésto puedes sumarle otras herramientas útiles de desarrollo como Frameworks de CSS o librerías de JS. Este es un listado sugerido de algunas herramientas que veremos en el curso, pero puedes agregar y/o utilizar las que desees:

* Navegador Web: [Chrome](https://www.google.com/intl/es/chrome/?brand=UUXU&gclid=Cj0KCQiA3Y-ABhCnARIsAKYDH7siyIILz6sp-rc9s7Gz41xrMQsGR3WyCY2D0t0XDIvQ3VnIZj_d43MaAhbJEALw_wcB&gclsrc=aw.ds), [Firefox](https://www.mozilla.org/es-AR/firefox/new/), [Edge](https://www.microsoft.com/es-es/edge), Opera, Brave.
* Editor de Código Fuente:  [Visual Studio Code](https://code.visualstudio.com/), [Sublime Text](https://www.sublimetext.com/3), [Atom](https://atom.io/).
* Framework CSS: [Bootstrap 5](https://getbootstrap.com/docs/5.0/getting-started/download/), [Bulma](https://bulma.io/) o [Milligram](https://milligram.io/#getting-started).
* Librerías: [jQuery](https://jquery.com/), [momentJS](https://momentjs.com/), [Sweet Alert](https://sweetalert2.github.io/), [Axios](https://axios-http.com/).
* Servidor Local: [Live Server (VS Code)](https://marketplace.visualstudio.com/items?itemName=ritwickdey.LiveServer), [XAMPP](https://www.apachefriends.org/es/index.html), [WampServer](https://www.wampserver.com/en/).

**Evolución de JavaScript**

Antes de iniciar con la manera de emplear JavaScript, vamos a repasar rápidamente su historia para resaltar el contexto que dio origen a esta tecnología . La primera versión de JavaScript se publicó en 1997, y dado que el lenguaje de programación se basa en el estándar ECMA Script, se denomina comúnmente a las versiones de JavaScript por las siglas ES, acompañadas del número de versión. Así, la versión de 1997 se denominó ES1. En cada versión se incluyeron nuevas herramientas en el lenguaje. Se hace mucho énfasis en la versión ES6 de Javascript porque fue la que introdujo más cambios radicales al lenguaje, y a lo largo del curso estaremos utilizando herramientas propias de ésta versión y posteriores ya que son las más recomendadas hoy. Veremos en contraste algunas contrapartes de la versión ES5, casos de buenas y malas prácticas, y la importancia de mantenerse actualizado.

Es importante entender que los lenguajes de programación se actualizan periódicamente, siendo necesario verificar los cambios entre versiones. No obstante, cuando una aplicación se desarrolla con una determinada versión, sólo se actualiza si es estrictamente necesario.

*Sintaxis y código*

**Código JavaScript**

JavaScript, como todos los lenguajes de programación, tiene su propia sintaxis, que es el conjunto de reglas que empleamos para escribir el [código fuente](https://es.wikipedia.org/wiki/C%C3%B3digo_fuente). En el Front End, el código JS está siempre asociado a un documento HTML, el cual podemos abrir con nuestro navegador.

**¿Cómo escribir código JavaScript?**

Existen dos formas de asociar código JavaScript a un HTML.:

1. Empleando la etiqueta <script></script> en alguna parte del documento HTML, y escribiendo dentro de ella el código JavaScript. Podemos usar dos barras para tipear una línea de comentario, la cual es una anotación que realiza el programador para orientarse respecto a lo que está codificando (esta línea no es interpretada por el navegador).
2. La segunda forma de incluir código JavaScript en nuestra aplicación es contar con un archivo .js y asociarlo al documento .html usando la etiqueta <script>, y la ruta del archivo JavaScript. Por ejemplo: <script src="js/main.js"></script>

**Sintaxis: reglas básicas**

Cuando escribimos código JavaScript, hay que tener en cuenta que no importa la cantidad de espacios en blanco que ingresemos, pero tenemos que tener especial cuidado en respetar las mayúsculas y minúsculas al tipear los elementos a utilizar. JavaScript es un lenguaje *case sensitive* porque distingue entre mayúsculas y minúsculas. Es decir, no es lo mismo escribir “UNO” que “uno”: se consideran diferentes. En comparación, HTML es un lenguaje *case insensitive*, donde <p> y <P> se entienden iguales.

Como desarrolladores y al igual que en otros lenguajes, también podemos insertar comentarios en nuestro código. En JS podemos agregar comentarios de una sóla línea con la combinación “//”, o comentarios de *bloque*, de varias líneas, envolviendo el comentario con los signos “/\* \*/”:

*// comentario de una línea*

*/\* comentario de  
 bloque de varias líneas \*/*

**Sintaxis: palabras reservadas**

Ya sabemos dónde escribimos el código JavaScript, así que ahora vamos a ver cómo lo hacemos. Usamos palabras para escribir las líneas de código, palabras que no determinan el funcionamiento de instrucción (línea de código) que estamos creando. Entonces, para aprender a programar en JavaScript debemos entender el significado de las ***palabras reservadas***, y cómo podemos emplearlas para programar.

Algunos ejemplos de estos términos son: break, case, catch, continue, default, let, delete, do, else, finally, for, function, if, in, instanceof, new, return, switch, this, throw, try, typeof, var, void, while, with, etcétera. En el curso aprenderemos a utilizar los más importantes para construir aplicaciones.

*Variables y valores*

**Variables**

Vamos a aprender ahora nuestras primeras palabras reservadas. Los términos a continuación se utilizan para crear elementos con valores asociados, a los cuales los conocemos por el nombre de variables. Es decir, una variable es un espacio reservado en la memoria dónde almacenamos algún tipo de dato, y que puede variar con el tiempo y transcurso de nuestra aplicación (de ahí su nombre *variable*).

Constantemente estamos trabajando con datos y operaciones de diverso tipo y transformándolos durante la ejecución. Esta información la almacenamos en estos contenedores que llamamos variables, a fin de tenerla disponible luego y llamarla cuando lo necesitemos.

**Declaración y asignación**

Al trabajar con variables tenemos que diferenciar estos dos procesos que llamamos declaración y asignación.

Declarar una variable significa *crearla*. Para esto usamos la palabra reservada **var**, **let** o **const** (ya veremos las diferencias). Escribimos una de estas palabras claves seguido del nombre de nuestra variable (este nombre es el que usamos para referenciar luego a la variable creada y acceder a sus valores almacenados):

***var*** *nombre*

***let*** *apellido*

***const*** *telefono*

Recordemos que para los nombres de nuestras variables no debemos utilizar ni espacios ni caracteres especiales (ñ, acentos, etc.). La palabra **var** corresponde a la versión ES5 para crear variables y actualmente se encuentra en desuso, ya que fue reemplazada por las palabras **let** y **const** de ES6 que implementan mejoras en el control de variables, y se recomienda utilizar exclusivamente estas dos últimas, evitando la primera.

Estas variables creadas en el ejemplo anterior están *vacías*;se declararon como corresponde pero no almacenan ningún valor. Para eso está el proceso de *asignación*, que es cuando definimos qué valores queremos guardar en ellas.

**Tipos de valores**

Podemos asociar distintos valores a una variable en JavaScript. Para empezar trabajaremos con los tipos de datos más comunes, que son las cadenas de texto y los números; en programación conocidos como tipos de dato ***string*** y ***number***respectivamente:

* *Number:* un valor numérico puede ser entero o decimal; cuando es entero se le dice “integer”, mientras que se lo denomina “float” cuando es decimal.
* *String:* por otro lado, un *string* o cadena de texto, es un valor compuesto por uno o más caracteres, y se caracteriza porque se define siempre entre comillas simples o dobles.

Es importante aclarar que en JavaScript las variables pueden guardar cualquier tipo de valores. Esto significa que si asignar primero un número no impide luego asignar un string a la misma variable. A esta característica en los lenguajes de programación se la llama “tipado dinámico”, e implica que las variables pueden guardar cualquier valor, sin que sea necesario especificar qué tipo de dato contiene la variable en la declaración de la misma.

Conociendo ésto ya podemos **asignar** valores a nuestras variables anteriores. La asignación de un valor a una variable se hace mediante el operador **=** (igual), siguiendo la sintaxis de *variable = valor*. Es decir, yo escribo la referencia a mi variable por su nombre, seguido del operador = , que significa asignación, y luego el valor que quiero guardar en esa variable. Por ejemplo, siguiendo los ejemplos anteriores:

*// asignando valores tipo string*

***nombre = “John”***

***apellido = “Doe”***

*// y asignando valores number*

***telefono = 11333444***

Nótese que el valor asignado en *telefono* no contiene espacios u otros caracteres como podría hacerse con normalmente con un teléfono, porque el *number* es siempre número entero o decimal y no contempla esos caracteres. Para ese fin podría almacenar el teléfono como un *string*, supongamos de la froma *“+54 9 (011) 555-4444”*, entendiendo que los caracteres numéricos forman parte de la cadena de texto.

También se puede simplificar el proceso anterior y *asignar* valores a una variable en el mismo momento en que la *declaro*:

***const*** *nombre = “John”*

***const*** *apellido = “Doe”*

***let*** *telefono = 11333444*

**Let y Const**

Las declaraciones con let y const tienen controles adicionales para las variables. Principalmente impiden que se puedan crear dos variables con el mismo nombre. El uso de *var* permitía este comportamiento y era propenso a generar errores, de ahí su posterior modificación.

Una variable declarada con *let* puede recibir múltiples asignaciones en el transcurso de la aplicación, es decir que puede cambiar de valor varias veces. Mientras que una variable declarada como *const* (de constante) solo puede recibir una **única asignación** al momento de su declaración, impidiendo que su valor se modifique luego. Usos prácticos de esto se verán a lo largo del curso.

**Operaciones básicas con variables numéricas**

Si tenemos variables con valores numéricos, podemos hacer operaciones matemáticas entre ellas. Es decir sumarlas, restarlas, multiplicarlas, dividirlas, etc. Para ésto utilizamos otros tipos de *operadores*, que son los símbolos matemáticos típicos, y lo hacemos siguiendo la lógica matemática tradicional. Ejemplo:

| <script>  var numeroA = 1;  let numeroB = 2;  const NUMEROC = 3;  //Suma de dos números (1 + 2 = 3)  let resultadoSuma = numeroA + numeroB;  //Resta de dos números (2 - 1 = 1)  let resultadoResta = numeroB - numeroA;  //Producto de dos números (2 \* 3 = 6)  let resultadoProducto = numeroB \* NUMEROC;  </script> |
| --- |

**Operaciones básicas con variables string - concatenación**

Si tenemos variables con valores string podemos concatenarlas para crear nuevas cadenas. Es decir, un único texto compuesto por el texto de las otras variables. Ejemplo:

| <script>  var textoA = "CODER";  let textoB = "HOUSE";  const BLANCO = " ";  //Concatenar textoA y textoB ("CODER" + "HOUSE" = "CODERHOUSE")  let resultadoA = textoA + textoB;  //Concatenar textoB y 1 ("HOUSE" + 1 = "HOUSE1")  let resultadoB = textoB + 1;  //Concatenar textoA, BLANCO y textoB ("CODER" + " " + "HOUSE" = "CODER HOUSE")  let resultadoC = textoA + BLANCO + textoB;  </script> |
| --- |

*Prompt, consola y alert*

**Prompt**

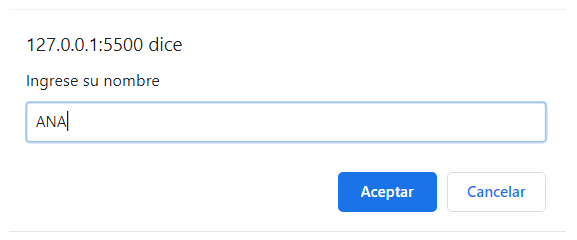
Ahora vamos aprender tres herramientas del lenguaje JavaScript que nos permiten interactuar con el usuario:

* La primera es la instrucción ***prompt()***, con la cual podemos solicitar un dato al usuario. Cuando la usamos, se visualiza en el navegador un cuadro de texto que pide un dato. Se busca asignar el valor ingresado por ***prompt()*** a una variable. Dicho valor que el usuario escribe es siempre una cadena de caracteres, es decir que se recibe como *string*. No podemos hacer operaciones matemáticas con strings, necesitamos que sean de tipo *number* para ello. Por lo tanto, si queremos recibir números por prompt debemos transformarlos, o *parsearlos*, a tipo number con una función particular.

Veamos primero un ejemplo de cómo usar prompt:

| let nombreIngresado = prompt("Ingrese su nombre"); |
| --- |

El usuario puede ingresar cualquier valor en este cuadro, y no puede limitarse el tipo de valor ingresado. A los datos que ingresa mediante prompt se los conoce con el nombre de ***entradas***, porque son datos que ingresan a la aplicación por la acción del usuario para ser procesados.



**Consola**

Si queremos mostrar el resultado de una variable tenemos dos formas de hacerlo. La primera, muy útil para el programador, es utilizar la instrucción ***console.log()***, la cual permite visualizar en la consola del navegador el valor asignado, una variable, o un valor cualquiera especificado entre los paréntesis. Veamos un ejemplo de cómo usar console.log:

| console.log("Mensaje de prueba"); |
| --- |
| let nombre = “John Doe”  console.log(nombre) |

La consola es una herramienta fundamental para cualquier desarrollador ya que nos permite ir evaluando los datos almacenados en nuestras variables en diversos momentos, y con esto llevar control del correcto funcionamiento de nuestra aplicación.

**Alert**

La segunda forma de mostrar al usuario el resultado de una variable es mediante la instrucción ***alert()***, la cual hace aparecer un cuadro de texto con el valor asignado de una variable, o un valor cualquiera especificado entre los paréntesis. Veamos un ejemplo de cómo usar alert:

| alert("¡Hola Coder!"); |
| --- |

El usuario visualizará el valor en un cuadro hasta que presione aceptar. A los datos que mostramos mediante alert se los conoce con el nombre de “salidas”, porque salen de la aplicación por el resultado de un procesamiento, siendo visibles para el usuario. 

El procesamiento implica una o más instrucciones que transforman el dato ingresado por el usuario (entrada) en información de valor, para él y la muestra (salida).

**Algoritmo**

Mencionamos anteriormente que necesitamos construir un procesamiento para transformar lo ingresado por el usuario en algo de valor para el mismo. Este proceso comprende una serie de pasos ordenados para transformar algo.

Pensemos en un ejemplo de la vida cotidiana: imaginemos que estamos cocinando arroz blanco, y los pasos para hacerlo podrían ser hervir el agua, arrojar el arroz, esperar 10 minutos. Si bien esta secuencia de pasos puede servir para las personas, ya que podemos interpretar cómo ejecutar cada uno de ellos, esto no es igual para las computadoras donde tenemos que escribir específicamente qué se debe hacer para llegar a un resultado, sentencia por sentencia. Se conoce con el nombre de **algoritmo** al conjunto de instrucciones ordenadas que programamos para realizar una tarea.

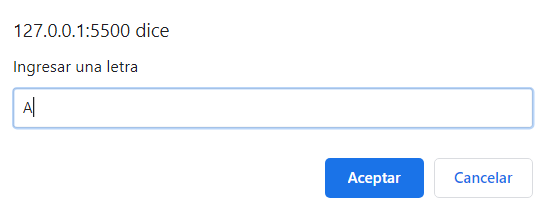
Pensar en algoritmos es una práctica que debemos fortalecer como desarrolladores, ya que consiste en encarar un problema complejo y dividir su resolución en diversos pasos, pensar cómo resolver cada paso y secuenciarlos correctamente para llegar al resultado esperado.

**Ejemplo de Script Completo**

| <!DOCTYPE *html*>  <html>  <head>  <title>Mi primer App - CoderHouse</title>  <script>  let entrada = prompt("Ingresar una letra");  let salida = entrada + " " + "ingresada";  alert(salida);  </script>  </head>  <body>  <h2>Esta página contiene una app</h2>  </body>  </html> |
| --- |

**Ejemplo de Entrada y Salida del Algoritmo Ejemplo**

**Si ingreso “A” como entrada:**



Obtengo como salida:

