



Aplicaciones WEB

JavaScript Básico

Oscar Fernando Aristizábal Cardona
Ingeniero de Sistemas y Telecomunicaciones
ofaczero@gmail.com

Lo que usted ya debe saber

Antes de comenzar el estudio de javascript, usted debe tener un conocimiento básico de:

- HTML
- CSS

Qué es js

Javascript es el lenguaje scripting orientado a la web.

Javascript es un lenguaje que se ejecuta del lado del cliente (navegador)

JavaScript se utiliza en la mayoría de páginas web para agregar funcionalidad, validar formularios, comunicarse con el servidor, y mucho más.

Por Qué js

- ❖ Javascript es el lenguaje de programación más popular en todo el mundo.
- ❖ Es un lenguaje para HTML y la web, para servidores, PCs, laptops, tablets, teléfonos inteligentes y más.
- ❖ Es un lenguaje de programación ligero.
- ❖ Javascript es insertado en páginas HTML, puede ser ejecutado por todos los navegadores web modernos.
- ❖ Javascript es fácil de aprender.

Cómo escribir js

Los scripts en HTML deben ser insertados entre las etiquetas **<script>** y **</script>**.

Los scripts se pueden poner en el **<body>** y en la sección **<head>** de una página HTML.

El **<script>** y **</script>** indica que el javascript inicia y termina.

Las líneas entre **<script>** y **</ script>** contiene el código javascript.

```
<script>
    alert("Mi primer javascript");
</script>
```

HTML DOM

JavaScript puede cambiar elementos HTML:

El HTML DOM (Document Object Model) es el estándar oficial de W3C para el acceso a los elementos HTML.

JavaScript puede manipular el DOM.

Usted puede usar JavaScript para:

- Cambiar los elementos HTML
- Eliminar los elementos HTML
- Crear nuevos elementos HTML
- Copiar y clonar los elementos HTML.
- Y mucho más ...

Escribiendo HTML

La manipulación de elementos HTML:

Para acceder a un elemento HTML desde JavaScript, puede utilizar el método **document.getElementById(id)**

```
<script>
    document.getElementById("mostrar").innerHTML="JavaScript";
</script>
```

Escribiendo en el documento HTML:

Escribir contenido directamente en la salida de documentos HTML.

```
<script>
    document.write("<p>My First JavaScript</p>");
</script>
```

Declaraciones de JS

Código Javascript:

JavaScript es una secuencia de instrucciones(declaraciones) a ser ejecutadas por el navegador.

El propósito de las declaraciones es decirle al navegador qué hacer.

Punto y coma:

Punto y coma separa sentencias de JavaScript.

Normalmente se agrega un punto (;) y coma al final de cada instrucción ejecutable, con punto y coma también permite escribir muchas declaraciones en una línea.

Javascript es sensible a mayúsculas:

JavaScript distingue entre mayúsculas y minúsculas.

Una función getElementByld no es la misma que getElementByID.

Y la variable llamada miVariable no es lo mismo que MiVariable.

Espacio en blanco:

JavaScript ignora espacios adicionales. Usted puede agregar espacio en blanco a su script para que sea más legible.

Romper una línea de código:

Puede dividir una línea de código dentro de una cadena de texto con una barra invertida (\).

Comentarios JS

Los comentarios de Javascript se pueden utilizar para hacer el código más legible.

Los comentarios no serán ejecutados por JavaScript.

Comentarios de una sola línea con //:

```
<script>
    // Esto es un comentario de una sola linea.
    var x = 10; // Comentario al final de linea
</script>
```

Comentarios de multiples líneas con /:**

```
<script>
    /* Esto es un comentario
       multiples lineas */
</script>
```

1. Variables JS

- Las variables de JavaScript son "contenedores" que almacenan información:
 - Al igual que con el álgebra, las variables de Javascript pueden ser usados para almacenar los valores ($x = 3$) o expresiones ($z = x + y$).
 - Las variable pueden tener nombres cortos (como x,y,z) o nombres más descriptivos como (edad, nombre, total).
 - Los nombres de variables deben comenzar con una letra.
 - Los nombres de variables distinguen entre mayúsculas y minúsculas.

2. Variables JS

Tipos de datos:

Las variables en Javascript también pueden contener otros tipos de datos, como valores de texto (nombre = "Jose").

En Javascript un texto como "Jose" se llama una cadena.

Al asignar un valor a una variable de texto, hay que poner comillas dobles o simples alrededor del valor.

Al asignar un valor numérico a una variable, no ponga comillas alrededor del valor.

Si usted pone comillas alrededor de un valor numérico, será tratado como texto.

3. Variables JS

Declarando (Creación) de variables:

La creación de una variable en Javascript es llamado a menudo como "declarando" una variable.

Para declarar variables en Javascript se usa la palabra clave **var**:

var vehiculo;

Después de la declaración, la variable está vacía (no tiene valor).

Para asignar un valor a la variable, utilice el signo igual:

vehiculo = "Renault";

Sin embargo, también se puede asignar un valor a la variable cuando se declara:

var vehiculo = "Renault";

1. Tipos de Datos JS

JavaScript tiene tipos de datos dinámicos. Esto significa que la misma variable puede tomar diferentes tipos de valores:

```
<script>
    var tipo;          // La variable tipo es indefinido(undefined)
    var tipo = 9;      // La variable tipo es un número
    var tipo = "AppWeb"; // La variable tipo es una cadena(String)
</script>
```

Tipo Cadena(String)

Una cadena es una variable que almacena una serie de caracteres como "App Web". Se pueden utilizar comillas simples o dobles:

```
<script>
    var nomfruta = "Manzana";
    var nomfruta = 'Manzana';
</script>
```

2. Tipos de Datos JS

Números(Numbers)

JavaScript tiene sólo un tipo de números. Los números pueden ser escritos con o sin decimales:

```
<script>
    var num1 = 32.00;      //Con decimales
    var num2 = 32;         //Sin decimales
</script>
```

Para declarar números muy pequeños o números muy grandes pueden ser escritos con notación científica (exponencial):

```
<script>
    var n1 = 124e5;        // 12400000
    var n2 = 124e-5;       // 0.00124
</script>
```

3. Tipos de Datos JS

Booleanos(Booleans)

Los variables de tipo booleano solo pueden tomar 2 valores, verdadero(true) y falso(false):

```
<script>
    var x = true      //Verdadero
    var y = false     //Falso
</script>
```

Arreglos, Vectores Unidimensionales (Arrays)

Un arreglo es una colección ordenada de elementos de un mismo tipo:

```
<script>
    var frutas = new Array();
    cars[0] = "Manzana";
    cars[1] = "Naranja";
    cars[2] = "Mandarina";
    var verduras = new Array("Zanahoria", "Lechuga", "Espinaca");    //Condensado
    var carros = ["Renault", "Fiat", "Mazda"];                         //Literal
</script>
```

4. Tipos de Datos JS

Objeto(Objects)

Un objeto está delimitado por llaves { }. Entre las llaves las propiedades del objeto se definen como pares de nombre y valor (**nombre: valor**). Las propiedades están separados por comas:

```
<script>
    var persona = { nombre:"Jose", apellido:"Mutis", telefono:8885555 };

    // Otra forma
    var persona = {
        nombre    : "Maria",
        apellido  : "Ruiz",
        telefono  : 8884444
    };

    // Acceder a los atributos del objeto.
    tel = persona.telefono;
    tel = persona["telefono"];
</script>
```

5. Tipos de Datos JS

Acceso a Métodos del objeto

```
<script>
  var persona = {
    nombre : "Jeremias",
    apellido : "Sprinfield",
    id : 681446,
    nombreCompleto : function (){return this.nombre + " " + this.apellido}
  };

  document.getElementById("mostrar").innerHTML = persona.
nombreCompleto();

</script>
```

6. Tipos de Datos JS

Indefinidos, Nulos(Undefined, Null)

Indefinido es el valor de una variable sin valor.

Las variables pueden ser vacias estableciendo el valor en null.

```
<script>
    var vehiculo; //Indefinido
    var persona = null;
</script>
```

Declarando tipos de variables

Cuando se declara una nueva variable, se puede declarar el tipo con la palabra clave (**new**):

```
<script>
    var a = new String;
    var b = new Number;
    var c = new Boolean;
    var d = new Array;
    var f = new Object;
</script>
```

Objetos JS

"Todo" en Javascript es un objeto: una cadena, un número, un vector, una fecha

En JavaScript, un objeto es un dato, con propiedades y métodos.

Propiedades y métodos:

Las propiedades son valores asociados a un objeto.

Los métodos son acciones que se pueden realizar sobre los objetos.

Objeto: Carro



Propiedades:

```
carro.nombre = "Fiat";  
carro.modelo = "500";  
carro.color    = "rojo";
```

Métodos:

```
carro.arrancar();  
carro.conducir();  
carro.frenar();
```

1. Funciones JS

Una función es un bloque de código que se ejecutará cuando "alguien" llame la función.

Sintaxis:

```
<script>
    function nombrefuncion()
    {
        //Código que será ejecutado.
    }
</script>
```

La función puede ser llamada directamente cuando se produce un evento (como cuando un usuario hace clic en un botón).

```
<script>
    <input type="button" value="Llamar Función" onclick="nombrefuncion();">
</script>
```

2. Funciones JS

Llamar a una función con argumentos

Cuando se llama a una función, se puede pasar algunos valores, estos valores se llaman argumentos o parámetros.

Estos argumentos pueden ser utilizados dentro de la función.

Usted puede enviar tantos argumentos como sea necesario, separados por comas (,)

```
<script>
    function miFuncion(var1,var2)
    {
        var resul = var1 + var2;
    }
    miFuncion(3,4);
</script>
```

Las variables y los argumentos deben estar en el orden esperado. La primera variable se le da el valor del primer argumento etc.

3. Funciones JS

Funciones con valor devuelto

A veces se requiere que una función retorne o devuelva un valor nuevo a donde se realizó la llamada.
Esto es posible mediante el uso de la instrucción **return**.

```
<script>
    function miFuncion(a,b)
    {
        return a*b;
    }
    document.getElementById("mostrar").innerHTML = miFuncion(5,4);
</script>
```

Operadores JS

Operadores Aritméticos:

Los operadores aritméticos se usan para realizar operaciones aritméticas entre las variables o valores.

Teniendo en cuenta que $y = 5$, la siguiente tabla se explican los operadores aritméticos:

Operador	Descripción	Ejemplo	Resultado X	Resultado Y
+	Suma	$x=y+2$	7	5
-	Resta	$x=y-2$	3	5
*	Producto	$x=y*2$	10	5
/	División	$x=y/2$	2.5	5
%	Módulo(residuo)	$x=y \% 2$	1	5
++	Incremento	$x=++y$	6	6
		$x=y++$	5	6
--	Decremento	$x=--y$	4	4
		$x=y--$	5	4

Operadores JS

Operadores de Asignación:

Los operadores de asignación se utilizan para asignar valores a las variables de Javascript.

Dado que $x = 10$, $y = 5$, la tabla a continuación explica los operadores de asignación:

Operador	Ejemplo	Igual	Resultado
=	$x=y$	$x=5$	$x=5$
+=	$x+=y$	$x=x+y$	$x=15$
-=	$x-=y$	$x=x-y$	5
=	$x=y$	$x=x*y$	50
/=	$x/=y$	$x=x/y$	2
%=	$x\%=y$	$x=x\%y$	0

Operadores JS

Operadores de Comparación:

Los operadores de comparación se utilizan en instrucciones lógicas para determinar la igualdad o la diferencia entre las variables o valores.

Dado que $x = 5$, la tabla a continuación explica los operadores de comparación:

Operador	Descripción	Comparación	Resultado
<code>==</code>	Es igual a	<code>x==8</code>	falso
<code>===</code>	Es exactamente igual a(valor y tipo)	<code>x === 5</code>	verdadero
<code>!=</code>	No es igual	<code>x != 8</code>	Verdadero
<code>!==</code>	No es igual(valor y tipo)	<code>x !== 5</code>	Falso
<code>></code>	Es mayor que	<code>x > 8</code>	Falso
<code><</code>	Es menor que	<code>x < 8</code>	Verdadero
<code>>=</code>	Es mayor o igual que	<code>x >= 8</code>	Falso
<code><=</code>	Es menor o igual que	<code>x <= 8</code>	Verdadero

Operadores JS

Operadores Lógicos:

Los operadores lógicos se utilizan para determinar la lógica entre variables o valores.

Dado que $x = 6$, $y = 3$, la siguiente tabla se explican los operadores lógicos:

Operador	Descripción	Ejemplo
<code>&&</code>	Y	$(x < 10 \&\& y > 1)$ es verdadero
<code> </code>	O	$(x==5 y==5)$ es falso
<code>!</code>	No	$!(x==y)$ es verdadero

Condicionales JS

Muy a menudo, cuando se escribe código Javascript, se desea llevar a cabo diferentes acciones para diferentes decisiones. En estos casos se debe utilizar sentencias condicionales.

Sentencia Si(if)

Utilice la sentencia **if** para ejecutar algún código sólo si una condición especificada es verdadera.

Sintaxis

```
<script>
  if(condición)
  {
    // Código que será ejecutado si la condición es verdadera.
  }
</script>
```

Condicionales JS

Sentencia Si(if)... No(else)

Utilice la sentencia **if... else** para ejecutar algún código si una condición es verdadera y otro código si la condición no es verdadera(falsa).

Sintaxis

```
<script>
  if(condición)
  {
    // Código que será ejecutado si la condición es verdadera.
  }
  else
  {
    // Código que será ejecutado si la condición no es verdadera.
  }
</script>
```

Condicionales JS

Sentencia Switch

Utilice la sentencia **switch** para seleccionar uno de muchos bloques de código que será ejecutado.

Sintaxis

```
<script>
  switch(variable)
  {
    case 1:
      // Bloque de código que se ejecutará
      break;
    case 2:
      // Bloque de código que se ejecutará
      break;
    default:
      // En cualquier otro caso el bloque de código que se ejecutará
  </script>
```

Ciclos JS

Los **ciclos** pueden ejecutar un bloque de código un número de veces establecido.

Si desea ejecutar el mismo código una y otra vez, se recomienda utilizar ciclos (bucles).

Ejemplo

```
<script>
    // En lugar de escribir:
    document.write(frutas[0] + "<br>");
    document.write(frutas[1] + "<br>");
    document.write(frutas[2] + "<br>");
    document.write(frutas[3] + "<br>");
    document.write(frutas[4] + "<br>");

    // Usted puede escribir:
    for (var i=0 ; i<frutas.length; i++)
    {
        document.write(frutas[i] + "<br>");
    }
</script>
```

Ciclos JS

JavaScript soporta diferentes tipos de **ciclos** (bucles):

- **for:**

Ciclos a través de un bloque de código un número de veces

- **for / in:**

Recorre las propiedades de un objeto.

- **while:**

Ciclos a través de un bloque de código mientras una condición especificada sea verdadera.

- **do / while:**

También ciclos a través de un bloque de código mientras una condición especificada sea verdadera

Ciclos JS

El Ciclo “for”

Sintaxis

```
<script>
  for (declaración1 ; declaración2 ; declaración3)
  {
    //Bloque de Código que será ejecutado.
  }
</script>
```

Declaración1:

Se ejecuta antes de que comience el ciclo (bloque de código).

Declaración2:

Define la condición para ejecutar el ciclo (bloque de código).

Declaración3:

Se ejecuta cada vez después de que el ciclo (bloque de código) se ha ejecutado.

Ciclos JS

El Ciclo “for/in”

El ciclo for/in es utilizado para recorrer las propiedades de un objeto.

Sintaxis

```
<script>
  var persona = {nombre:"Jeremias", apellido:"Sprinfield", edad:33};

  for (var x in persona) {
    console.log(persona[x]);
  }
</script>
```

Ciclos JS

El Ciclo “while”

Sintaxis

```
<script>
    while (condición)
    {
        //Bloque de Código que será ejecutado.
    }
</script>
```

El ciclo “**while**” ejecuta un bloque de código, siempre y cuando la condición especificada sea verdadera.

```
<script>
    while (i < 10) {
        console.log("El número es: " + i;
        i++;
    }
</script>
```

Ciclos JS

El Ciclo “do/while”

Sintaxis

```
<script>
  do {
    //Bloque de Código que será ejecutado.
  }
  while (condición);
</script>
```

Este ciclo ejecutará el bloque de código una vez, antes de verificar si la condición es verdadera, entonces se repite el ciclo mientras la condición es verdadera.

```
<script>
  do {
    console.log("El número es: " + i;
    i++;
  }
  while (i < 10);
</script>
```

Break / Continue JS

La sentencia **break** "salta" de un ciclo.

La sentencia **continue** "salta por encima" de una iteración en el ciclo.

Sintaxis

```
<script>
  for (var i = 0; i < 10; i++) {
    if (i == 3) {
      break;
    }
    console.log("El número es: "+i);
  }
</script>
```

```
<script>
  for (var i = 0; i <= 10; i++) {
    if (i == 3) continue;
    console.log("El número es: "+i);
  }
</script>
```

Eventos

Los eventos **HTML** son "cosas" que le ocurren a elementos **HTML**, cuando se utiliza **JavaScript** en páginas **HTML**, **JavaScript** puede "reaccionar" a estos eventos.

Eventos **HTML**

Un evento **HTML** puede ser algo que el navegador hace, o lo que hace un usuario.

Ejemplos de eventos de **HTML**:

- Una página web **HTML** ha terminado de cargar.
- Un campo de entrada **HTML** que se cambió.
- Un botón **HTML** al que se le hizo click.

Sintaxis

```
<elementoHTML algunEvento='Código JavaScript'>
```

Eventos

Lista de algunos eventos comunes en HTML:

Evento	Descripción
onchange	Un elemento HTML se ha cambiado
onclick	El usuario hace click en un elemento HTML
onmouseover	El usuario mueve el ratón sobre un elemento HTML
onmouseout	El usuario mueve el ratón fuera de un elemento HTML
onkeydown	El usuario presiona una tecla del teclado
onload	El navegador ha terminado de cargar la página

Métodos de Cadena

- **length**

```
var txt = "abcdefghijklmnopqrstuvwxyz";
console.log(txt.length);
```

- **substring()**

```
var str = "Microsoft, Google, Apple";
console.log(str.substring(11,17));
```

- **replace()**

```
var str = "Microsoft, Google, Apple";
console.log(str.replace("Microsoft","ADSI"));
```

- **toUpperCase()**

```
var texto = "Hola Mundo!";
console.log(texto.toUpperCase());
```

- **toLowerCase()**

```
var texto = "Hola Mundo!";
console.log(texto.toLowerCase());
```

Métodos de Números

- **toString()**

```
var num = 789;  
console.log(num.toString());
```

- **toFixed()**

```
var num = 7.356;  
console.log(num.toFixed(0));
```

- **toPrecision()**

```
var num = 7.356;  
console.log(num.toPrecision(2));
```

- **parseInt()**

```
var num = “12”;  
console.log(parseInt(num));
```

- **parseFloat()**

```
var num = “15.55”;  
console.log(parseFloat(num));
```

Matemáticas

- **Math.random()**

```
console.log(Math.random()*5);
```

- **Math.min() / Math.max()**

```
console.log(Math.min(3,5,2,8));  
console.log(Math.max(3,5,2,8));
```

- **Math.round()**

```
console.log(Math.round(3.8));
```

- **Math.ceil()**

```
console.log(Math.ceil(3.3));
```

- **Math.floor()**

```
console.log(Math.floor(3.7));
```

- **Math.pow()**

```
console.log(Math.pow(3,2));
```

- **Math.abs()**

```
console.log(Math.abs(-55));
```

Fechas - métodos get

- **getDate()**

```
var dt = new Date();
console.log(dt.getDate());
```

- **getFullYear()**

```
var dt = new Date();
console.log(dt.getFullYear());
```

- **getDay()**

```
var dt = new Date();
console.log(dt.getDay());
```

- **getMonth()**

```
var dt = new Date();
console.log(dt.getMonth());
```

Fechas - métodos set

- **setDate()**

```
var dt = new Date();
console.log(dt.setDate(30));
```

- **setFullYear()**

```
var dt = new Date();
console.log(dt.setFullYear(2020,10,18));
```

- **setMonth()**

```
var dt = new Date();
console.log(dt.setMonth(6));
```

- **setMinutes()**

```
var dt = new Date();
console.log(dt.setMinutes(40));
```

1. Métodos de Vectores

- **valueOf()**

```
var carros = ["BMW", "WolksWagen", "Fiat", "Renault"];
document.getElementById("mostrar").innerHTML = carros.valueOf();
```

- **toString()**

```
document.getElementById("mostrar").innerHTML = carros.toString();
```

- **join()**

```
document.getElementById("mostrar").innerHTML = carros.join(" * ");
```

- **sort()**

```
carros.sort();
```

- **reverse()**

```
carros.reverse();
```

2. Métodos de Vectores

- **pop()**

```
var carros = ["BMW", "WolksWagen", "Fiat", "Renault"];
carros.pop();
```

- **shift()**

```
carros.shift();
```

- **push()**

```
carros.push("Ferrari");
```

- **unshift()**

```
carros.unshift("Mazda");
```

Eventos de Tiempo

Es posible ejecutar un código en intervalos de tiempo especificados. Esto se conoce como eventos de tiempo.

Los dos métodos principales que se utilizan son:

setInterval()

- Ejecuta una función, una y otra vez, en intervalos de tiempo específicos.

```
window.setInterval("función javascript", milisegundos);  
window.clearInterval(variableIntervalo)
```

setTimeout ()

- Ejecuta una función, solo una vez, después de esperar un número especificado de milisegundos.

```
window.setTimeout("función javascript", milisegundos);  
window.clearTimeout(variableTiempoFuera)
```

Ventanas Emergentes

JavaScript tiene tres tipos de ventanas emergentes:

alert()

- Un cuadro de alerta se utiliza a menudo si usted quiere asegurarse de que la información llega hasta el usuario, cuando un cuadro de alerta aparece, el usuario tendrá que hacer clic en "Aceptar" para continuar.

```
window.alert("Hola Mundo...");
```

confirm ()

- Un cuadro de confirmar a menudo se utiliza si desea que el usuario pueda verificar o aceptar algo, cuando un cuadro confirmar aparece, el usuario tendrá que hacer clic en "Aceptar" o "Cancelar" para continuar.
- Si el usuario hace clic en "Aceptar", el cuadro retorna verdadero. Si el usuario hace clic en "Cancelar", el cuadro retorna falso.

```
window.confirm("Usted estudia ADSI?");
```

Ventanas Emergentes

prompt()

- Un cuadro de mensaje se utiliza a menudo si desea que el usuario introduzca un valor antes de entrar en una página, cuando un cuadro de mensaje aparece, el usuario tendrá que hacer clic en "OK" o "Cancelar" para continuar después de introducir un valor de entrada.
- Si el usuario hace clic en "Aceptar" del cuadro devuelve el valor de entrada. Si el usuario hace clic en "Cancelar" el cuadro devuelve null.

```
window.prompt("Cuál es su Nombre?");  
window.prompt("Cuál es su Nombre?", "Jeremias");
```