



UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

**Centro de
e-Learning**

Professional backend developer

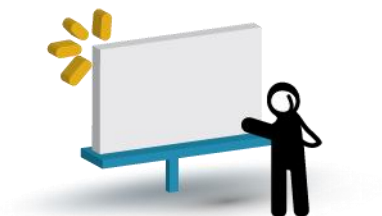
Centro de e-Learning SCEU UTN - BA.

Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148

www.sceu.frba.utn.edu.ar/e-learning

Modulo 2: Javascript

Unidad 1: Introducción a Javascript



Presentación:

En la presente unidad veremos los conceptos básicos de javascript, sintaxis y declaración de funciones.



Objetivos:

Que los participantes*:

- Aprendan que es javascript
- Comprendan su sintaxis básica
- Aprendan a declarar funciones



Bloques temáticos*:

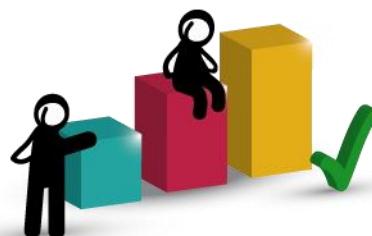
- Versiones
- Tipos de ejecución
- Incluir el archivo javascript en nuestro HTML
- Modo estricto
- Comentarios
- Separación de instrucciones
- Variables
- Declaración de Variables
- Ámbito de las Variables
- Operadores de cadenas
- Typeof
- Operadores lógicos
- Operadores de asignación
- Operadores de comparación
- Operadores aritméticos

Centro de e-Learning SCEU UTN - BA.

Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning



- Funciones
- Arrays
- Longitud de un array
- For
- While
- Condicional IF



Consignas para el aprendizaje colaborativo

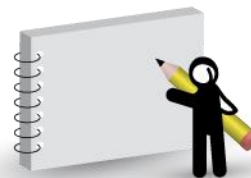
En esta Unidad los participantes se encontrarán con diferentes tipos de actividades que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

- Los foros proactivos asociados a cada una de las unidades.
- La Web 2.0.
- Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen algunas de las actividades sugeridas y compartan en los foros los resultados obtenidos.

Además, también se propondrán reflexiones, notas especiales y vinculaciones a bibliografía y sitios web.

El carácter constructivista y colaborativo del MEC nos exige que todas las actividades realizadas por los participantes sean compartidas en los foros.



Tomen nota*

Las actividades son opcionales y pueden realizarse en forma individual, pero siempre es deseable que se las realice en equipo, con la finalidad de estimular y favorecer el trabajo colaborativo y el aprendizaje entre pares. Tenga en cuenta que, si bien las actividades son opcionales, su realización es de vital importancia para el logro de los objetivos de aprendizaje de esta instancia de formación. Si su tiempo no le permite realizar todas las actividades, por lo menos realice alguna, es fundamental que lo haga. Si cada uno de los participantes realiza alguna, el foro, que es una instancia clave en este tipo de cursos, tendrá una actividad muy enriquecedora.

Asimismo, también tengan en cuenta cuando trabajen en la Web, que en ella hay de todo, cosas excelentes, muy buenas, buenas, regulares, malas y muy malas. Por eso, es necesario aplicar filtros críticos para que las investigaciones y búsquedas se encaminen a la excelencia. Si tienen dudas con alguno de los datos recolectados, no dejen de consultar al profesor-tutor. También aprovechen en el foro proactivo las opiniones de sus compañeros de curso y colegas.

Javascript

JavaScript (abreviado comúnmente JS) es un lenguaje de programación interpretado.

Se utiliza principalmente en su forma del lado del cliente (client-side), implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario

Tipos de ejecución

Ejecución directa

Es el método de ejecutar scripts más básico. En este caso se incluyen las instrucciones dentro de la etiqueta <SCRIPT>, cuando el navegador lee la página y encuentra un script va interpretando las líneas de código y las va ejecutando una después de otra.

Respuesta a un evento

Los eventos son acciones que realiza el usuario. Los programas como Javascript están preparados para atrapar determinadas acciones realizadas, en este caso sobre la página, y realizar acciones como respuesta.

Por ejemplo la pulsación de un botón, el movimiento del ratón o la selección de texto de la página.



Versiones

Año	Nombre	Descripción
1997	ECMAScript 1	Primer edición
1998	ECMAScript 2	Editorial con algunos cambios
1999	ECMAScript 3	Añadió try/catch y expresiones regulares
	ECMAScript 4	Nunca se puso en producción
2009	ECMAScript 5	Añadió soporte a JSON
2011	ECMAScript 5.1	Cambios editoriales
2015	ECMAScript 6	Añadió clases y módulos
2016	ECMAScript 7	Añadió operador exponencial



Incluir el archivo javascript en nuestro HTML

```
<html>
  <head>
    <title>Clase 1 react</title>
    <link rel="stylesheet" href="styles.css">
  </head>
  <body>
    </body>
  <script src="script.js"></script>
</html>
```

Esto debe ser incluido en el head del html o bien al finalizar el body, de esta última forma nos aseguramos que todos los elementos estén cargados al ejecutar el script.

También podemos introducir nuestro código javascript dentro de la etiqueta `<script></script>`. Por ejemplo:

```
<script>
function mi_funcion() {
  document.getElementById("mi_funcion").innerHTML = "Ejemplo Mi funcion";
}
</script>
```



Modo estricto

El modo estricto de ECMAScript 5 es una forma de elegir una variante *restringida* de *JavaScript*, así implícitamente se deja de lado el modo poco riguroso. El modo estricto no es sólo un subconjunto: *intencionalmente* tiene diferencia semántica del código normal. Los navegadores que no admiten el modo estricto ejecutarán el código con un comportamiento diferente a los que sí lo soportan, por lo tanto no confíes en el modo estricto sin antes hacer pruebas de sus características más relevantes. Los modos estricto y no estricto pueden coexistir, por lo tanto el código se puede transformar a modo estricto incrementalmente.

El modo estricto tiene varios cambios en la semántica normal de JavaScript:

1. Elimina algunos errores silenciosos de JavaScript cambiándolos para que lancen errores.
2. Corrige errores que hacen difícil para los motores de JavaScript realizar optimizaciones: a veces, el código en modo estricto puede correr más rápido que un código idéntico pero no estricto.
3. Prohíbe cierta sintaxis que probablemente sea definida en futuras versiones de ECMAScript.

Cambiar algunos errores sintácticos tolerados sin su uso

1. Hace imposible crear variables globales por accidente. En JavaScript no estricto, si se escribe mal una variable en una asignación, se creará una nueva propiedad en el objeto global y el código continuará "trabajando" como si nada (aunque es posible que el código así escrito falle en el futuro, en concreto, en JavaScript moderno). En modo estricto, cualquier asignación que produzca variables globales por accidente lanzará un error:

```
'use strict';  
  
// Asumiendo que exista una variable global llamada mistypedVariable  
mistypeVariable = 17; // esta línea lanza un ReferenceError debido a  
// una errata en el nombre de la variable
```

2. Lanza una excepción en asignaciones que de otro modo fallarían silenciosamente. Por ejemplo, NaN es una variable global que no puede ser asignada. En un código normal, asignar a NaN no tiene efecto; el programador no recibe ningún mensaje de error.



3. Lanza una excepción al intentar eliminar propiedades no eliminables (mientras que en código normal el intento no tendría ningún efecto):

```
'use strict';  
delete Object.prototype; // lanza un TypeError
```

4. Todas las propiedades nombradas en un objeto sean únicas. En código normal se pueden duplicar nombres, siendo el último el que determina el valor de la propiedad.

```
'use strict';  
var o = { p: 1, p: 2 }; // !!! error de sintaxis
```

5. Requiere que los nombres de los parámetros de una función sean únicos. En código normal, el último argumento repetido oculta argumentos anteriores con el mismo nombre.

```
function sum(a, a, c) { // !!! error de sintaxis  
  'use strict';  
  return a + a + c; // incorrecto si este código se ejecutó  
}
```

6. Prohíbe establecer propiedades en valores primitivos.

```
(function() {  
  'use strict';  
  
  false.true = ''; // TypeError  
  (14).sailing = 'home'; // TypeError  
  'with'.you = 'far away'; // TypeError  
  
})();
```



Comentarios

Los comentarios puede realizarse de una línea completa (//) o bien indicando la apertura y cierre del mismo con /* ... */

```
<script>
//Este es un comentario de una línea
/*Este comentario se puede extender
por varias líneas.
Las que quieras*/
</script>
```

También podemos desde nuestro HTML incluir un archivo con extensión .js de la siguiente manera:

```
<script src="/carrito_compra/assets/bootstrap/js/bootstrap.min.js"></script>
```

En este caso el archivo js no tendrá embebidas las etiquetas **<script>** sino que contendrá directamente el código programado bajo la sintaxis de javascript.

Propuesta:

- Crea un archivo .html y embebe código javascript dentro. Ejecute dicho archivo en el navegador (haciendo doble clic en el)
- Quita el código embebido del html, crea un archivo js con el mismo código js e incluirlo en el html.
- Abrí el navegador, apreté la tecla f12, hace clic en la pestaña consola y ejecuta el mismo código js. ¿Para qué crees que es útil esta herramienta?

El código js que puedes utilizar es el siguiente:

alert("Hola Mundo");



Separación de instrucciones

Javascript tiene dos maneras de separar instrucciones:

- La primera es a través del carácter punto y coma (;)

```
llamadoFuncion();
```

- La segunda es a través de un salto de línea.

```
llamadoFuncion()
```



Variables

Una variable es un espacio en memoria donde se almacena un dato, un espacio donde podemos guardar cualquier tipo de información que necesitemos para realizar las acciones de nuestros programas.

Los nombres de las variables han de construirse con caracteres alfanuméricos y el carácter subrayado (_).

Los nombres tienen que comenzar por un carácter alfabético o el subrayado. **No podemos utilizar caracteres raros como el signo +, un espacio.**

Las variables no pueden utilizar nombres reservados, por ejemplo if, for, while, etc

Declaración de Variables

Declarar variables consiste en definir y de paso informar al sistema de qué vas a utilizar una variable.

Javascript cuenta con la palabra "**var**" que utilizaremos cuando queramos declarar una o varias variables. Como es lógico, se utiliza esa palabra para definir la variable antes de utilizarla.

Ejemplo:

```
var operando1;  
var operando2;  
  
var operando1 = 23;  
var operando2 = 33;  
  
var operando1, operando2;
```




Ejemplo:

```
-----<script language="javascript">

    //creo una variable
    var x;
    //x actualmente no tiene ningún valor
    x = 25;
    //ahora x tiene el valor numérico 25

    //creo una segunda variable
    var y = 230;
    //he creado una variable y asignado un valor en un solo paso!!

    //las variables guardan datos con los que puedo realizar operaciones
    var suma;
    suma = x + y;
    //he creado la variable suma y he asignado la suma de x e y
    alert(suma);
    //he mostrado el valor de la variable suma

</script>
```



Nota: Si nosotros no declaramos el **modo estricto**, podremos utilizar variables sin necesidad de declararlas con la palabra reservada **var**. Ejemplo:

Para declarar el **strict mode** debemos anteponer al código “**strict mode**” por ejemplo:

```
v = 15
function f1() {
  "use strict";
  var v = "Hi! I'm a strict mode script!";
}
```

En este caso la primer asignación se hace sin modo estricto por lo cual como vemos no tiene la palabra var adelante.

En cambio dentro de la función f1 se hace en modo estricto por lo cual debe tener la palabra var delante.



Ámbito de las Variables

Se le llama **ámbito de las variables** al lugar donde estas están disponibles. Por lo general, cuando declaramos una variable hacemos que esté disponible en el lugar donde se ha declarado.

- Variables globales: son las que están declaradas en el ámbito más amplio posible.

```
<script>  
var variableGlobal  
</script>
```

- Variables locales: sólo podremos acceder a ellas dentro del lugar donde se ha declarado

```
<script>  
function miFuncion () {  
    var variableLocal  
}  
</script>
```

En este caso la variable “variableGlobal” se podrá acceder desde cualquier script en ejecución o a través de scripts embebidos

En el caso de “variableLocal” solo se accederá desde la función miFuncion.

NO se recomienda el uso de variables locales.



Operadores de cadenas

Las cadenas de caracteres, o variables de texto, también tienen sus propios operadores para realizar acciones típicas sobre cadenas. Aunque javascript sólo tiene un operador para cadenas se pueden realizar otras acciones con una serie de funciones predefinidas en el lenguaje que veremos más adelante.

Para concatenar 2 cadenas de texto debemos usar el operador “+”

```
<script>  
cadenaConcatenada = cadena1 + cadena2 /  
</script>
```

El operador + también es utilizado para la suma. En caso de querer sumar dos valores asegurarse que ambas variables tienen un tipo de dato Number para que el motor infiera que se quiere realizar una suma y no una concatenación. Para esto se puede utilizar la función parseInt o parseFloat, por ejemplo:

```
var resultado = parseInt(operador1) + parseInt(operador2)
```

De esta manera nos aseguramos que las variables operador1 y operador2 se tomen como number, se sumen y no se concatenen



Operadores lógicos

Estos operadores sirven para realizar operaciones lógicas, que son aquellas que dan como resultado un verdadero o un falso, y se utilizan para tomar decisiones en nuestros scripts.

```
<script>

prueba && prueba2 //Operador and

prueba || prueba2 //Operador or

</script>
```

Operador and: Retorna true si todos los operandos son true

Operador or: Retorna true si al menos un operador retorna true

Typeof

Para comprobar el tipo de un dato se puede utilizar otro operador que está disponible a partir de javascript 1.1, el **operador typeof**, que devuelve una cadena de texto que describe el tipo del operador que estamos comprobando.

```
<script>

document.write("<br>El tipo de booleano es: " + typeof booleano)

</script>
El tipo de booleano es: boolean
```



Operadores de asignación

Un operador de asignación asigna un valor al operando de la izquierda en función del valor del operando de la derecha. El operador básico de asignación es el de igual (=), que asigna el valor del operando de la derecha al operando de la izquierda. Por ejemplo, $x = y$, está asignando el valor de y a x .

Nombre	Operador abreviado	Significado
Operadores de asignación	$x = y$	$x = y$
Asignación de adición	$x += y$	$x = x + y$
Asignación de sustracción	$x -= y$	$x = x - y$
Asignación de multiplicación	$x *= y$	$x = x * y$
Asignación de división	$x /= y$	$x = x / y$
Asignación de resto	$x \% = y$	$x = x \% y$
Asignación de exponenciación	$x ** = y$	$x = x ** y$
Asignación de desplazamiento a la izquierda	$x << = y$	$x = x << y$
Asignación de desplazamiento a la derecha	$x >> = y$	$x = x >> y$
Asignación de desplazamiento a la derecha sin signo	$x >>> = y$	$x = x >>> y$
Asignación AND binaria	$x \& = y$	$x = x \& y$
Asignación XOR binaria	$x \wedge = y$	$x = x \wedge y$
Asignación OR binaria	$x = y$	$x = x y$



Operadores de comparación

Un operador de comparación compara sus operandos y devuelve un valor lógico en función de si la comparación es verdadera (true) o falsa (false). Los operadores pueden ser numéricos, de cadena de caracteres (Strings), lógicos o de objetos.


Operador	Descripción	Ejemplos devolviendo true
Igualdad (==)	Devuelve true si ambos operandos son iguales.	<code>3 == var1</code> <code>"3" == var1</code> <code>3 == "3"</code>
Desigualdad (!=)	Devuelve true si ambos operandos no son iguales.	<code>var1 != 4</code> <code>var2 != "3"</code>
Estrictamente iguales (===)	Devuelve true si los operandos son igual y tienen el mismo tipo. Mira también <code>Object.is</code> y <code>sameness in JS</code> .	<code>3 === var1</code>
Estrictamente desiguales (!==)	Devuelve true si los operandos no son iguales y/o no son del mismo tipo.	<code>var1 !== "3"</code> <code>3 !== "3"</code>
Mayor que (>)	Devuelve true si el operando de la izquierda es mayor que el operando de la derecha.	<code>var2 > var1</code> <code>"12" > 2</code>
Mayor o igual que (>=)	Devuelve true si el operando de la izquierda es mayor o igual que el operando de la derecha.	<code>var2 >= var1</code> <code>var1 >= 3</code>
Menor que (<)	Devuelve true si el operando de la izquierda es menor que el operando de la derecha.	<code>var1 < var2</code> <code>"2" < 12</code>
Menor o igual que (<=)	Devuelve true si el operando de la izquierda es menor o igual que el operando de la derecha.	<code>var1 <= var2</code> <code>var2 <= 5</code>



Operadores aritméticos

Los operadores aritméticos toman los valores numéricos (tanto literales como variables) de sus operandos y devuelven un único resultado numérico. Los operadores aritméticos estándar son la suma (+), la resta (-), la multiplicación (*) y la división (/).

Tabla 3.3 Operadores aritméticos

Operador	Descripción	Ejemplo
Resto (%)	Operador binario correspondiente al módulo de una operación. Devuelve el resto de la división de dos operandos.	12 % 5 devuelve 2.
Incremento (++)	Operador unario. Incrementa en una unidad al operando. Si es usado antes del operando (++x) devuelve el valor del operando después de añadirle 1 y si se usa después del operando (x++) devuelve el valor de este antes de añadirle 1.	Si x es 3, entonces ++x establece x a 4 y devuelve 4, mientras que x++ devuelve 3 y, solo después de devolver el valor, establece x a 4.
Decremento (--)	Operador unario. Resta una unidad al operando. Dependiendo de la posición con respecto al operando tiene el mismo comportamiento que el operador de incremento.	Si x es 3, entonces --x establece x a 2 y devuelve 2, mientras que x-- devuelve 3 y, solo después de devolver el valor, establece x a 2.
Negación Unaria (-)	Operación unaria. Intenta convertir a número al operando y devuelve su forma negativa.	- "3" devuelve -3. - true devuelve -1.
Unario positivo (+)	Operación unaria. Intenta convertir a número al operando.	+ "3" devuelve 3. + true devuelve 1.
Exponenciación (**) 	Calcula la potencia de la base al valor del exponente. Es equivalente a $\text{base}^{\text{exponente}}$	2 ** 3 devuelve 8. 10 ** -1 devuelve 0.1.



Funciones

Primero se escribe la palabra **function**, reservada para este uso. Seguidamente se escribe el nombre de la función, que como los nombres de variables puede tener números, letras y algún carácter adicional como en guion bajo.

```
<script>
function nombrefuncion () {
    instrucciones de la función
    ... |
}
</script>
```

Entre paréntesis, luego del nombre de la función, se definirán los parámetros que recibirá la misma. En el caso de js no hace falta el tipo de dato en cada parámetro recibido.

Arrays

Los arrays son objetos similares a una lista cuyo prototipo proporciona métodos para efectuar operaciones de recorrido y de mutación. Tanto la longitud como el tipo de los elementos de un array son variables.

Dicho en otras palabras, imaginemos que en una variable podemos guardar un solo elemento (nuestro teléfono). Bien un array seria como un organizador con varios bloques en el cual podemos almacenar un elemento por bloque (ejemplo teléfono en un bloque, auriculares en otro, la funda en el tercero, etc)

El primer paso para utilizar un array es crearlo. Para ello utilizamos un objeto Javascript ya implementado en el navegador.

```
<script>
//Array vacio
var miArray = new Array()
//Creacion de array con numero de compartimentos
var miArray = new Array(10)
//Setear datos
miArray[0] = 290
miArray[1] = 97
miArray[2] = 127
//Declaracion e inicializacion
var arrayRapido = [12,45,"array inicializado en su declaración"]
</script>
```

A diferencia de PHP no podemos tener vectores asociativos, es decir arrays cuya clave sea un string.



Longitud de un array

Para obtener la longitud de un array utilizaremos el método **length**

```
<script>  
document.write("Longitud del array: " + miArray.length)  
</script>
```

For

La estructura for nos permitirá recorrer un array en javascript.

Cada elemento recorrido se reconoce como una iteración, se “cortara” el for cuando la condición sea falsa.

```
<script>  
for (inicialización; condición; actualización) {  
    //sentencias a ejecutar en cada iteración  
}  
</script>
```

- Inicialización: Aquí inicializamos las variables. Por ejemplo: **i=0**
- Condición: Mientras la condición sea verdadera se seguirán produciendo iteraciones. Ejemplo **i<5**
- Actualización: se actualiza la variable de la condición. Ejemplo: **i++**



While

La estructura while es similar a la for, nos permite realizar iteraciones sobre un array

```
<script>
while (condición){
    //sentencias a ejecutar
}
</script>
```

Como vemos no cuenta con el elemento de inicialización ni el de actualización.

Condicional IF

La estructura if nos permitirá mediante el uso de operadores de comparación tomar una decisión a partir de si dicha comparación o valor de una expresión es verdadera o falsa (en este caso entra por el else)

```
<script>

if (expresión) { |
    //acciones a realizar en caso positivo
    //...
} else {
    //acciones a realizar en caso negativo
    //...
}

</script>
```



Bibliografía y Webgrafía utilizada y sugerida

<http://www.w3schools.com/Js/>

<http://www.desarrolloweb.com/manuales/20/>

<http://www.desarrolloweb.com/articulos/826.php>

<http://www.desarrolloweb.com/articulos/827.php>

<http://www.desarrolloweb.com/articulos/846.php>

<http://www.desarrolloweb.com/articulos/861.php>



Lo que vimos:

En esta unidad aprendimos los conceptos básicos de javascript, su sintaxis y declaración de funciones



Lo que viene:

En la próxima unidad aprendemos sobre el manejo del DOM de javascript

