# JAVASCRIPT

- Declaración y Uso de Variables
- Operaciones Aritméticas
  - Operadores Aritméticos
  - Operadores de Asignación
  - Operadores de Comparación

## Declaración y Uso de Variables

- **Variable**: espacio en memoria donde se almacena un dato.
- En JavaScript se utiliza la palabra reservada **"var"** para declarar una variable.
- El tipo de dato será definido dependiendo de cómo asignemos valores a la variable.
- JavaScript acepta los siguientes tipos de datos: números (integer o float), string´y boolean, así como valores null.

## Ejemplo:

var a = "cadena"; --> tipo de dato **string**
var b = 2; --> tipo de dato **integer**
var c = true; --> tipo de dato **boolean**

Donde:

- los nombres de las variables son: a,b,c
- los valores almacenados en cada variable son: "cadena", 2, true.
- El nombre de la variable puede contener valores alfanuméricos Cualquier letra minúscula (a-z) o letra mayúscula (A-Z) alfabeto, números (0-9) o línea baja (_).

## Operadores Aritméticos

- Son los utilizados para la realización de operaciones matemáticas simples como la suma o resta.

| OPERADOR | DESCRIPCION | EJEMPLO |
|---|---|---|
| + (suma) | Usado para adicionar dos números. | a=3+2;<br>//a=5 |
| - (resta) | Usado para substraer dos números. | a=3-2:<br>//a=1 |
| * (multiplicación) | Usado para multiplicación de dos números. | a=3*2;<br>//a=6 |
| / (división) | Usado para división de dos números. | a=10/2;<br>//a=5 |
| % (modulo ) | Usado para encontrar el residuo. | a=3%2;<br>//a=1 |

## Operadores de Asignación

- Son utilizados para asignar un valor a una variable.

| OPERADOR | DESCRIPCION | EJEMPLO |
|---|---|---|
| = (asignación) | Asigna la parte de la derecha del igual a la parte de la izquierda. A al derecha se colocan los valores finales y a la izquierda generalmente se coloca una variable donde queremos guardar el dato. | b = 3; |
| += (asignación con suma) | Realiza la suma de la parte de la derecha con la de la izquierda y guarda el resultado en la parte de la izquierda. | b = 3;<br>b += 4;<br>// b=7 |
| -= (asignación con resta) | Realiza la resta de la pate de la derecha con la de la izquierda y guarda el resultado en la parte de la izquierda. | b = 4;<br>b -= 2;<br>//b=2 |
| *= (asignación con multiplicación) | Realiza la multiplicación de la parte de la derecha con la de la izquierda y guarda el resultado en la parte de la izquierda. | b = 4;<br>b *= 2;<br>//b=8 |
| /= (asignación de la división ) | Realiza la division de la parte de la derecha con la de la izquierda y guarda el resultado en la parte de la izquierda. | b = 6;<br>b /= 2;<br>//b =3 |

## Operadores de Comparación

- Son los utilizados para comparar valores numéricos, cadenas o booleanos.

| OPERADOR | DESCRIPCION | EJEMPLO |
|---|---|---|
| == | valida la condición cuando dos valores **son iguales .** | If (a==3) {...;} |
| != | valida la condición cuando dos valores **no son iguales .** | If (a!=3) {...;} |
| > | valida la condición cuando un numero es **mayor que otro.** | If (>3) {...;} |
| < | valida la condición cuando un numero es **menor que otro.** | If (a<3) {...;} |
| >= | compara los números y chequea si un numero es **mayor o igual que otro .** | If (a>=3) {...;} |
| <= | compara los números y chequea si un numero es **menor o igual que otro .** | If (a<=3) {...;} |

- Operadores Lógicos
- Uso de If
- Uso de Switch

## Operadores Lógicos

- Los operadores Lógicos se utilizan para combinar múltiples comparaciones en una expresión condicional.

| OPERADOR | DESCRIPCION | EJEMPLO |
|---|---|---|
| && | " Y " Devuelve true si ambos operadores son true. | If ((a==b) && (c>b)) {...;} |
| \|\| | " O " Devuelve true si uno de los operadores es true. | If ((a==b)\|\|(b==c)) {...;} |
| ! | "No" Devuelve true si la negación del operando es true. | If (!(b>c)) {...;} |

## Uso de If

- La declaración If es usada para verificar una condición y ejecutar un conjunto de declaraciones solamente si la condición es true ( verdadera ).
- **Sintasis:**

```
if (condición)
{
    // conjunto de declaraciones que serán ejecutadas
    // solamente si la condición es true
}
```

- **Código Ejemplo:**

```
<script language="javascript">
var a = "if check";
if(a == "if check")
{
document.write(" adentro if función ");
}
</script>
```

## If-Else

- mediante esta instrucción podemos hacer que el proceso que empieza por la comparación de una variable tome otro camino.
- **Síntasis:**

```
if  (condición){
    // conjunto de declaraciones si la condición es satisfactoria
}
else{
    // conjunto de declaraciones si la condición falla
}
```

- **Código Ejemplo:**

```
<script language="javascript">
var a = "1234abc";
if(a == "adcdefa"){
      document.write(" cumple la condición if ");
}else{
      document.write(“ no cumple la condición if ");
}
</script>
```

## Uso de Switch

- La declaración Switch Case es usada cuando una condición puede tener múltiples resultados y un conjunto diferente de operaciones a ejecutar.
- **Sintasis:**

```
 Switch(condición)
{
case result1:
    // Operación para result1
 break;
case result2:
    // Operación para result2
break;
.
.
default :
    // Si el resultado no pertenece a ninguno de los casos especificados
}
```

- **Código ejemplo:**

```
<script language="javascript">
for(var i=1; i<5; i++)
{
switch(i)
{
   case 1:
      document.write("mensaje para case 1 <br>");
      break;
   case 2:
      document.write("mensaje para case 2 <br>");
      break;
   case 3:
      document.write("mensaje para case 3 <br>");
      break;
   default:
      document.write("mensaje para case default<br>");
      break;
}
}
</script>
```

- Usos de Ciclos (For, While, Do-While)
  - Uso de break y continue

## Ciclos

## For

- El bucle FOR se utiliza para repetir instrucciones un determinado número de veces.
- De entre todos los bucles, el FOR se suele utilizar cuando sabemos el número de veces que queremos que se ejecute la sentencia.

## For

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

## For

```
for (i=1;i<=6;i++){
   document.write("<H" + i + ">Encabezado de
   nivel " + i + "</H" + i + ">");
}
```

## While

- Se utiliza cuando queremos repetir la ejecución de unas sentencias un número indefinido de veces, **siempre que se cumpla una condición**.
- Es más sencillo de comprender que el FOR, pues no incorpora en la misma línea la inicialización de las variables, su condición para seguir ejecutándose y su actualización.
- Sólo se indica, la condición que se tiene que cumplir para que se realice una iteración.

## While

```
while (condición)
{
    sentencias a ejecutar
}
```

## While

```
var suma = 0;
  while (suma < 1000){
      suma = suma + 100;
      document.write (suma + "<br>");}
```

## Do-While

- Se utiliza generalmente cuando no sabemos cuantas veces se habrá de ejecutar el ciclo, igual que el WHILE, con la diferencia de que sabemos que el ciclo **por lo menos** se ejecutará **una vez**.

## Do-While

```
do
{
    sentencias del bucle
} while (condición)
```

## Do-While

```
var suma = 1000;
do{
    suma = suma + 100;
    document.write (suma + "<br>");
}while (suma < 1000)
```

## Break

- Se detiene un bucle utilizando la palabra *break*.
- Detener un bucle significa salirse de él y dejarlo todo como está para continuar con el flujo del programa inmediatamente después del bucle.

## Break

```
var suma = 0;
while (suma < 1000){
  suma = suma + 100;
  document.write (suma + "<br>");
  if (suma == 500)
      break;
}
```

## Continue

- Sirve para volver al principio del bucle en cualquier momento, sin ejecutar las líneas que haya por debajo de la palabra *continue*.

## Continue

```
var suma = 0;
while (suma < 1000){
  suma = suma + 100;
  if (suma == 500)
      continue;
  document.write (suma + "<br>");
}
```

- Uso de Funciones: Function ()
  - Uso de return

## Funciones ¿Qué son?

- Procesos que se agrupan en uno solo.
- Pueden concebir de forma independiente
- Son más sencillos de resolver que el problema entero.
- Suelen ser utilizados repetidas veces a lo largo de la ejecución del programa

**Función**

## Funciones

### DEFINICIÓN

Así que podemos ver una función como una serie de instrucciones que englobamos dentro de un mismo proceso.

Este proceso se podrá luego ejecutar desde cualquier otro sitio con solo llamarlo.

## ¿Cómo se escribe una función?

Una función se debe definir con una sintaxis especial, al igual que otros lenguajes

```
function nombrefuncion (param1, param2, …)
{
  instrucciones de la función
  …
}
```

## Cómo llamar a una función

Para ejecutar la función utilizamos su nombre seguido de los paréntesis.

Así llamaríamos a la función:

```
nombrefuncion(param1,  param2,  …)
```

## Donde colocar las funciones

En cualquier parte de la página, siempre entre etiquetas <SCRIPT> (bloques).

Recomendación: colocar la función antes de cualquier llamada a la misma.

```
<SCRIPT>
function miFuncion(){
      document.write("Esto va bien") }
…
miFuncion()
</SCRIPT>
```

## Donde colocar las funciones

**Colocar la función en otro bloque de script:**

También es válido que la función se encuentre en un bloque <SCRIPT> anterior al bloque donde está la llamada.

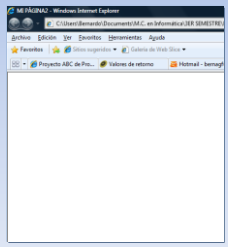
```
<HTML>
<HEAD>
  <TITLE>MI PÁGINA</TITLE>
<SCRIPT>
function miFuncion(){
   //hago algo…
   document.write("Esto va bien")
}
</SCRIPT>
</HEAD>
<BODY>

<SCRIPT>
miFuncion()
</SCRIPT>

</BODY>
</HTML>
```


## Donde colocar las funciones

### ERROR

La funcion es llamada antes de reconocerla como creada.

```
<html>
<head>
<title>MI PÁGINA2</title>
</head>
<SCRIPT>
miFuncion()
</SCRIPT>

<SCRIPT>
function miFuncion(){
  document.write("Esto va bien")
}
</SCRIPT>
<BODY>


</BODY>
</HTML>
```



## Función: Parámetros

Los parámetros se usan para mandar valores a las funciones.

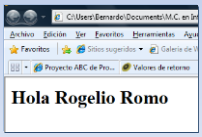
Una función trabajará con los parámetros para realizar las acciones.

Por decirlo de otra manera, los parámetros son los valores de entrada que recibe una función.

## Función: Parámetros

Ejemplo:

```
<html>
<head>
<title>MI PÁGINA</title>

<SCRIPT language=JavaScript>
function escribirNombre(nombre){
document.write("<H1>Hola " + nombre + "</H1>")
}

</SCRIPT>
</head>

<BODY>


<SCRIPT>
        escribirNombre("Rogelio Romo")
</SCRIPT>
</BODY>
</HTML>
```




## Función: Parámetros

Los parámetros pueden recibir cualquier tipo de datos, numérico, textual, booleano o un objeto.

Realmente no especificamos el tipo del parámetro, por eso debemos tener un cuidado al pasarle valores, para asegurarnos que todo es consecuente con los tipos de datos que esperamos tengan nuestras variables o parámetros.

## Función: Múltiples Parámetros

EJEMPLO:

```
<html>
<head>
<title>MI PÁGINA</title>

<SCRIPT language=JavaScript>
        function escribirBienvenida(nombre,colorTexto){
          document.write("<FONT color='" + colorTexto + "'>")
          document.write("<H1>Hola " + nombre + "</H1>")
          document.write("</FONT>")
        }
</SCRIPT>
</head>
<BODY>
<SCRIPT>
        var miNombre = "Pepe"
        var miColor = "red"
        escribirBienvenida(miNombre,miColor)
</SCRIPT>
</BODY>
</HTML>
```


## Función: Parámetros por valor

**Los parámetros se pasan por valor**

Esto quiere decir que estamos pasando valores y no variables.

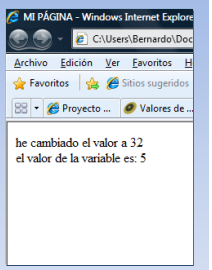
En la práctica, aunque modifiquemos un parámetro en una función, la variable original que habíamos pasado no cambiará su valor.

**En JavaScript sólo se pasan las variables por valor**

## Función: Parámetros por valor

EJEMPLO:

```
<html>
<head>
<title>MI PÁGINA</title>

<SCRIPT language=JavaScript>
        function pasoPorValor(miParametro){
          miParametro = 32
          document.write("he cambiado el valor a 32")
        }
</SCRIPT>
</head>
<BODY>
<SCRIPT>
        var miVariable = 5
        pasoPorValor(miVariable)
        document.write ("<br>el valor de la variable es: " +
miVariable)

</SCRIPT>
</BODY>
</HTML>
```



## Return: Devolución de valores en las funciones

Las funciones en JavaScript también pueden retornar valores.

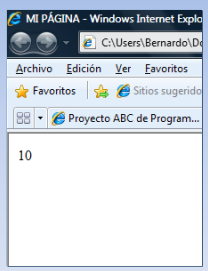
De hecho, ésta es una de las utilidades más esenciales de las funciones, que debemos conocer, no sólo en JavaScript sino en general en cualquier lenguaje de programación.

De modo que, al invocar una función, se podrá realizar acciones y ofrecer un valor como salida.

## Función: Return

### EJEMPLO:

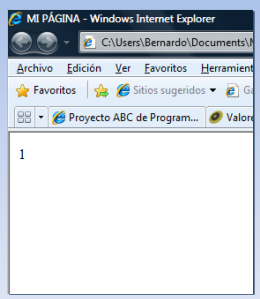
```
<html>
<head>
<title>MI PÁGINA</title>

<SCRIPT language=JavaScript>
            function media(valor1,valor2){
              var resultado
              resultado = (valor1 + valor2) / 2
              return resultado
            }
</SCRIPT>
</head>
<BODY>
<SCRIPT>
            var miMedia
            miMedia = media(12,8)
            document.write (miMedia)
</SCRIPT>
</BODY>
</HTML>
```




## Función: Múltiple Return

### EJEMPLO:

```
<html>
<head>
<title>MI PÁGINA</title>
<SCRIPT language=JavaScript>
function multipleRet(numero){
   var resto = numero % 2
   if (resto == 0)
     return 0
   else
     return resto
}
</SCRIPT>
</head>
<BODY>
<SCRIPT>
            var miResiduo
            miResiduo = multipleRet(9)
            document.write (miResiduo)
</SCRIPT>
</BODY>
</HTML>
```




- Manejo de Strings
- Declaración y Uso de Arreglos

## MANEJO DE STRING

El objeto STRING se usa para manipular cadenas de caracteres.

En JavaScript todo texto encerrado entre comillas , dobles o simples
Se interpreta como una cadena.

La operación de crear una variable de este tipo se lleva a cabo con el
Operador New.

Este objeto nos permite hacer diversas manipulaciones con las cadenas,
para que trabajar con ellas sea más sencillo.

## PROPIEDADES:

**length**: devuelve la longitud de la cadena.
**prototype**: permite agregar métodos y propiedades al objeto

```
<HTML>
<HEAD>
  <title>Ejemplo de Propiedad length </title>
</HEAD>
<BODY>
<script LANGUAGE="JavaScript">
<!--
var cad = «Ejemplo de length";

with(document) {
write("La cadena es: "+cad+"<BR>");
write("Longitud de la cadena: "+cad.length+"<BR>") ;
}
</script>
</BODY>
</HTML>
```

## Métodos de String

Los objetos de la clase String tienen una buena cantidad de métodos.

BIG / SMALL

**big()**
Aumenta el tamaño de letra del string.

**small()**

Se utiliza para mostrar una cadena en una fuente pequeña

## Ejemplo:

```
<HTML>
<HEAD>
  <title>Ejemplo de Propiedad length </title>
</HEAD>
<BODY>
<script LANGUAGE="JavaScript">
<!--
var gran = "Grande";
var peq= "Pequeño";

//with(document) {
document.write("La cadena en Grande: "+gran.big()+"<BR>");
document.write("La cadena en pequeño: "+peq.small()+"<BR>") ;
//}
</script>
</BODY>
</HTML>
```

## toLowerCase / toUpperCase

**toLowerCase()**

Pone todas los caracteres de un string en minúsculas.

**toUpperCase()**

Pone todas los caracteres de un string en mayúsculas.

## EJEMPLO:

```
<HTML>
<HEAD>
  <title>Ejemplo toLowercase/toUpperCase</title>
</HEAD>
<BODY>
<script LANGUAGE="JavaScript">
<!--
var may = "MAYUSCULAS";
var min= "minusculas";

with(document) {
write("La cadena en Grande: "+may.toLowerCase()+"<BR>");
write("La cadena en pequeño: "+min.toUpperCase()+"<BR>") ;
}
</script>
</BODY>
</HTML>
```

## indexOf (carácter,desde)

Devuelve la posición de la primera vez que aparece el carácter indicado por parámetro en un string.

Si no encuentra el carácter en el string devuelve -1.
El segundo parámetro es opcional y sirve para indicar a partir de que posición se desea que empiece la búsqueda.

## lastIndexOf (carácter,desde)

Busca la posición de un carácter exactamente igual a como lo hace la función indexOf pero desde el final en lugar del principio.
El segundo parámetro indica el número de caracteres desde donde se busca, igual que en indexOf.

## Ejemplo:

```
<HTML>
<HEAD>
  <title>Ejemplo indexOf/lastIndexOf</title>
</HEAD>
<BODY>
<script LANGUAGE="JavaScript">
<!--
var cad = "Posicion de la cadena";


with(document) {
write("En la posicion: "+cad.lastIndexOf("n")+"<BR>");
write("En la posicion: "+cad.indexOf("z")+"<BR>") ;
}
</script>
</BODY>
</HTML>
```

## EJEMPLO GENERAL.

```
<HTML>
<HEAD>
  <title>Ejemplo de Genral de JavaScript</title>
</HEAD>
<BODY>
<script LANGUAGE="JavaScript">
<!--
var cad = "Hola Mundo",i;
var ja = new Array();

ja = cad.split("o");

with(document) {
write("La cadena es: "+cad+"<BR>");
write("Longitud de la cadena: "+cad.length+"<BR>");
write("Haciendola ancla: "+cad.anchor("b")+"<BR>");
write("En grande: "+cad.big()+"<BR>");
write("Parpadea: "+cad.blink()+"<BR>");
write("Caracter 3 es: "+cad.charAt(3)+"<BR>");
write("Fuente FIXED: "+cad.fixed()+"<BR>");
write("De color: "+cad.fontcolor("#FF0000")+"<BR>");
write("De color: "+cad.fontcolor("salmon")+"<BR>");
write("Tamaño 7: "+cad.fontsize(7)+"<BR>");
write("<I>orl</I> esta en la posicion: "+cad.indexOf("orl"));
```

```
write("<BR>En cursiva: "+cad.italics()+"<BR>");
write("La primera <I>l</I> esta, empezando a contar por detras,");
write(" en la posicion: "+cad.lastIndexOf("l")+"<BR>");
write("Haciendola enlace: "+cad.link("doc.htm")+"<BR>");
write("En pequeño: "+cad.small()+"<BR>");
write("Tachada: "+cad.strike()+"<BR>");
write("Subindice: "+cad.sub()+"<BR>");
write("Superindice: "+cad.sup()+"<BR>");
write("Minusculas: "+cad.toLowerCase()+"<BR>");
write("Mayusculas: "+cad.toUpperCase()+"<BR>");
write("Subcadena entre los caracteres 3 y 10: ");
write(cad.substring(2,10)+"<BR>");
write("Entre los caracteres 10 y 3: "+cad.substring(10,2)+"<BR>");
write("Subcadenas resultantes de separar por las <B>o:</B><BR>");
for(i=0;i<ja.length;i++) write(ja[i]+"<BR>");
}
//-->
</script>
</BODY>
</HTML>
```

# ARRAYS

Los arrays son estructuras para almacenar una lista de valores.

En javascript los arrays no son un tipo de variable sino que fueron Implementados por Netscape como un objeto

### Ejemplo:

```
<HTML>
<HEAD>
  <title>Ejemplo de  Array en JavaScript</title>
</HEAD>
<BODY>
<script LANGUAGE="JavaScript">
<!--

vsemana = new Array(7);
miLista = new Array(1,5,9);
nombres= new Array('Juan', 'Luis', 'María');
vacio = new Array();
interesante = new Array(4);

document.write("El array vsemana: "+vsemana);
document.write("El array miLista: "+miLista);
document.write("El array nombres: "+nombres);
document.write("El array vacio: "+vacio);
document.write("El array interesante: "+interesante);

//-->
</script>
</BODY>
</HTML>
```

Los **arrays** poseen sus **propiedades** y **métodos** predefinidos, que son ampliables por el usuario.

## Propiedades

### Length

Como su nombre indica esta propiedad nos devuelve la longitud del array, es decir, el número de elementos que puede almacenar.

Su uso es muy simple:

```
var lista = new Array(50);
tamagno = lista.length;
```

**Prototype**

Esta es una propiedad muy potente en el sentido que nos permite agregar al objeto Array las propiedades y métodos que queramos.

```
Array.prototype.descriptor = null;
dias = new Array ('lunes', 'Martes', 'Miercoles', 'Jueves',
'Viernes');
dias.descriptor = "Dias laborables de la semana";
```

En este ejemplo hemos creado una nueva propiedad para el objeto array, la propiedad descriptor que podría utilizarse para darle un título a la matriz.

**concat(objArray)**

Une el objeto Array con el array que se le pasa como argumento y devuelve el resultado en un nuevo array, sin modificar los arrays que se concatenan.

**join()**

Convierte los elementos de un array en una cadena separados por el carácter que se le indique. El separador por defecto es la coma.

```
a= new Array("Hola","Buenos","días");
document.write(a.join() +" <br>");
document.write(a.join(", ") +" <br>");
document.write(a.join(" + ") +" <br>") ;
```

La salida de este programa sería

Hola,Buenos,Días
Hola, Buenos, Días
Hola+Buenos+Días

**reverse()**
Invierte el orden de los elementos de un Array en el propio array, sin crear uno nuevo.

**slice(ini, fin)**
Extrae parte de un Array devolviéndolo en un nuevo objeto Array.

**sort(rutord)**
Ordena alfabéticamente los elementos de un objeto Array

- Declaración de Clases y uso de objetos
- Uso de For…in

## Bloque For…In

- El bloque For…In permite iterar a través de todos los elementos de un arreglo o a través de las propiedades de un objeto
- Sintaxis

```
for (variable in objeto)
  {
      [Código a ejecutarse]

  }
```

## Bloque For…In

- Funcionamiento
  - El código en el cuerpo del bloque se ejecuta para cada elemento/propiedad
  - El argumento variable puede ser
    - Una variable definida
    - Un elemento del arreglo
    - La propiedad de un objeto

## Bloque For…In

- Ejemplo 1

```
<html>
<body>

<script type="text/javascript">
var x;
var mycars = new Array();
mycars[0] = "Saab";
mycars[1] = "Volvo";
mycars[2] = "BMW";

for (x in mycars)
  {
  document.write(mycars[x] + "<br />");
  }
</script>

</body>
</html>
```

## Bloque For…In

- Ejemplo 2

```
<script type="text/javascript">
<!--
var aProperty;
document.write("Propiedades del Browser<br /><br /> ");
for (aProperty in navigator)
{
  document.write("<b>" + aProperty + ":</b>" + navigator[aProperty]);
  document.write("<br />");
}
//-->
</script>
```

## Clases y objetos

## Clases en Javascript

- Javascript es un lenguaje orientado a objetos
  - Podemos crear nuestros propios objetos
  - Podemos crear nuestros propios métodos
- Javascript NO está basado en clases, sino en prototipos
- Un prototipo es un objeto abstracto, capaz de contener otros objetos dentro, los cuales pueden ser distintos tipos de datos
  - variables (numeros, cadenas de texto, valores lógicos)
  - Vectores
  - funciones
  - otros grupos de objetos

## Clases en Javascript

- En vez de programar una clase, para estar orientados a objetos en JS definimos un prototipo
- Las variables dentro de este serán las propiedades, y las funciones serán los métodos

```
[Objeto = Prototipo]
{
     [ Propiedad = Variable ]
     [ Método = Función ]
 }
```

## Clases en Javascript

- Creación de objetos
  - Hay dos formas de definir la función constructora
    - Definir una función

```
function NombreFunción( parametros )
{
    // Código
};
```

    - Definir una variable cuyo contenido sea una función

```
var NombreFunción = function( parametros )
{
    // Código
};
```

## Clases en Javascript

```
function MiClase (valor_inicializacion){
    //Inicializo las propiedades y métodos
    this.miPropiedad = valor_inicializacion
    this.miMetodo = nombre_de_una_funcion_definida
}
```

## Un ejemplo…

- El constructor de la clase

```
function Alumno(nombre, edad){
    this.nombre = nombre
    this.edad = edad
    this.boleta = null
}
```

## Un ejemplo…

- Para construir un método debemos crear una función. Una función que se construye con intención de que sea un método para una clase puede utilizar también la variable this, que hace referencia al objeto sobre el que invocamos el método
- El método

```
function asignaBoleta(numBoleta){
    this.boleta = numBoleta
}
```

## Un ejemplo…

- Otro método, para mostrar la información

```
function imprimirDatos(){
        document.write("Nombre: " + this.nombre)
        document.write("<br>Edad: " + this.edad)
        document.write("<br>Número boleta: " + this.boleta)
}
```

## Un ejemplo…

- La clase, incluyendo los métodos

```
function Alumno(nombre, edad){
    this.nombre = nombre
    this.edad = edad
    this.boleta = null
    this.asignaBoleta = asignaBoleta
    this.imprimirDatos = imprimirDatos
}
```

## Un ejemplo…

- Instanciando el objeto y accediendo a sus componentes

```
//Creamos un alumno
miAlumno = new Alumno("Juan Pérez", "20")

//Imprimimos los datos iniciales
miAlumno.imprimirDatos()

//Guardamos la boleta
miAlumno.asignaBoleta("BC0001")

//Imprimimos los datos actualizados
miAlumno.imprimirDatos()
```

- Verificación de Formularios
  - Manejo de Eventos
    - Onload y Onunload
    - Onfocus, Onblur, Onchange

## Manejo de eventos

- Los eventos son la manera que tenemos en Javascript de controlar las acciones de los visitantes y definir un comportamiento de la página cuando se produzcan. Cuando un usuario visita una página web e interactúa con ella se producen los eventos y con Javascript podemos definir qué queremos que ocurra cuando se produzcan.

## Cómo se define un evento

- Para definir las acciones que queremos realizar al producirse un evento utilizamos los manejadores de eventos. Existen muchos tipos de manejadores de eventos, para muchos tipos de acciones del usuario. El maneador de eventos se coloca en la etiqueta HTML del elemento de la página que queremos que responda a las acciones del usuario.

## onLoad

- Este evento se desata cuando la página, o en Javascript 1.1 las imágenes, ha terminado de cargarse.
  Javascript 1.0

```
<body onLoad="alert('El documento se cargo por completo')">

<body onLoad="document.getElementById('clave').value=1;">
```

## onUnload

- Al abandonar una página, ya sea porque se pulse sobre un enlace que nos lleve a otra página o porque se cierre la ventana del navegador, se ejecuta el evento onunload.
  Javascript 1.0

```
<body onunload="alert('Saliendo de la pagina!')">

<body onunload="validar()">
```

## onBlur

- Se desata un evento onblur cuando un elemento pierde el foco de la aplicación. El foco de la aplicación es el lugar donde está situado el cursor, por ejemplo puede estar situado sobre un campo de texto, una página, un botón o cualquier otro elemento.
  Javascript 1.0

```
<input type="text" value="122" onfocus="this.blur()">
```

## onFocus

- El evento onfocus es lo contrario de onblur. Se produce cuando un elemento de la página o la ventana ganan el foco de la aplicación. Javascript 1.0

```
<input type="text" value="122" onfocus="this.blur()">
```

```
<input type="text" value="122" onfocus="this.blur()">
```

## onChange

- Se desata un evento onblur cuando un elemento pierde el foco de la aplicación. El foco de la aplicación es el lugar donde está situado el cursor, por ejemplo puede estar situado sobre un campo de texto, una página, un botón o cualquier otro elemento. Javascript 1.0

```
<select name="secciones" onchange="destino()">
```

- Verificación de Formularios
  - Manejo de Eventos
    - Onsubmit
    - Onmouseover y Onmouseout

## onSubmit

Una de las grandes aportaciones de JavaScript a la creación de interfaces web es la posibilidad de acceder al contenido de los campos de los formularios para realizar acciones sobre los valores introducidos por el usuario, modificarlos y, en última instancia, validarlos. La validación de los datos de un formulario mediante scripts JavaScript no sustituye a la validación que debe realizarse, por motivos de seguridad, en la aplicación del servidor que recibe la información.

## onSubmit

JavaScript, desde sus comienzos, introdujo los mecanismos necesarios para validar campos de formulario. Estas son algunas de las validaciones típicas:

- Comprobar que se han suministrado todos los campos obligatorios
- Comprobar que el formato de un campo es el esperado (fechas, teléfonos, etc.)
- Comprobar la validez (sintáctica) de las direcciones de correo y URLs
- Comprobar que no se sobrepasa la longitud, número de líneas o tamaño de la entrada

## onSubmit

- La validación de campos de formulario se basa en interceptar el momento en que el usuario realiza el envío de los datos del formulario (es decir, pulsa sobre el botón de enviar).
- Cuando el usuario pulsa sobre el botón de enviar, se genera el evento **submit**, asociado al envío de datos de un formulario. JavaScript proporciona un mecanismo para capturar este evento, lo que nos permite ejecutar un script justo antes de que se realice el envío de los datos.
- La forma de capturar el evento consiste en introducir el atributo **onSubmit** en la etiqueta del formulario cuyo evento **submit** queremos capturar.

## onsubmit

```
<form name="miFormulario" action="mailto:mi@mail.com"
onSubmit="alert('Has pulsado enviar.'); return false;">
<input type="submit" value="Enviar" name="enviar">
</form>
```

## onMouseOver y onMouseOut

El evento onMouseOver se ejecuta cuando pasamos la flecha del mouse sobre un elemento y el evento onMouseOut cuando la flecha abandona el mismo.

- **onmouseout**

El ratón "sale" del elemento (pasa por encima de otro elemento)

- **onmouseover**

El ratón "entra" en el elemento (pasa por encima del elemento)

## onMouseOver y onMouseOut

```
<script language="JavaScript">
function pintar(col)
{
  document.bgColor=col;
}
</script>
<a href="pagina1.html" onMouseOver="pintar('#ff0000')"
onMouseOut="pintar('#ffffff')">Rojo</a>
<a href="pagina1.html" onMouseOver="pintar('#00ff00')"
onMouseOut="pintar('#ffffff')">Verde</a>
<a href="pagina1.html" onMouseOver="pintar('#0000ff')"
onMouseOut="pintar('#ffffff')">Azul</a>
<a href="pagina2.html">Pagina 2</a>
</body>
</html>
```