	<p align="center"><b>UNIVERSIDAD DON BOSCO</b>  <b>FACULTAD DE INGENIERÍA</b>  <b>ESCUELA DE COMPUTACIÓN</b></p>
<p><b>CICLO: 02/2021</b></p>	<p align="center"><b>GUIA DE LABORATORIO #2</b></p> <p><b>Nombre de la Practica:</b> Estructuras de control: sentencias repetitivas y matrices  <b>Lugar de Ejecución:</b> Centro de Cómputo  <b>Tiempo Estimado:</b> 2 horas con 30 minutos  <b>MATERIA:</b> Desarrollo de Aplicaciones Web con Software Interpretado en el Cliente</p>

## I. OBJETIVOS

Competencias a adquirir:

- ☐ Dominio del uso de las sentencias repetitivas, ciclos o lazos del lenguaje JavaScript.
- ☐ Aplicación de sentencias repetitivas en la solución de problemas que requieran repetir un conjunto de instrucciones.
- ☐ Utilización de sentencias repetitivas anidadas con lenguaje JavaScript.
- ☐ Uso apropiado de sentencias de control de ciclos o lazos (*break* y *continue*).
- ☐ Utilización de matrices en la solución de problemas prácticos.
- ☐ Capacidad de definir matrices unidimensionales y multidimensionales para resolver problemas.
- ☐ Manejo de las estructuras de control repetitivas para asignar, acceder, eliminar y ordenar los elementos de una matriz.
- ☐ Destreza en el uso de funciones para facilitar el manejo de matrices en JavaScript.

## II. INTRODUCCION TEORICA

### Sentencias repetitivas

Las sentencias repetitivas son el medio que brindan los lenguajes de programación para poder repetir un bloque o conjunto de instrucciones más de una vez. Estas sentencias suelen llamarse lazos, ciclos o bucles. El número de veces que se repite el ciclo o lazo es controlado mediante una condición o mediante el valor de un contador. Cuando se trata de una condición, se involucra una variable cuyo valor cambia cada vez que se ejecuta el lazo. En el caso de una variable contador aumenta de valor de forma automática, cada vez que se ejecuta el lazo hasta llegar a un valor máximo definido en el contador.

JavaScript proporciona varios tipos de sentencias repetitivas o lazos, entre ellos se pueden mencionar: *for*, *while* y *do ... while*. Otras instrucciones particulares de JavaScript, relacionadas con el uso de objetos, son *for ... in* y *with*.

#### Sentencia *for*

Permite crear un lazo o bucle que se ejecutará un número determinado de veces. Utiliza una variable contador, una condición de comparación, una instrucción de incremento (o decremento) del contador y una o más instrucciones que forman parte del cuerpo del lazo o bucle. Estas instrucciones se repetirán tantas veces hasta que la condición de comparación se evalúe como falsa.

La sintaxis de la sentencia *for* es la siguiente:

```
for (inicialización; condicion; incremento/decremento){
    //instrucción o bloque de instrucciones;
```

```
}
```

### Ejemplo:

```
document.write("Conteo hacia atrás<br>");
for(var i=10; i>=0;i--){
    document.write("<b>" + i + "</b>");
    document.write("<br>");
}
document.write("Fin del conteo.");
```

### El bucle *while*

Otra forma de crear un bucle es utilizando la sentencia *while*. Funciona de forma similar al *for*, pero su estructura sintáctica es completamente diferente. La estructura básica es:

```
while (condicion) {
    //bloque de código ;
}
```

Donde, *condicion* es cualquier expresión JavaScript válida que se evalúe a un valor booleano. El bloque de código se ejecuta mientras que la condición sea verdadera. Por ejemplo:

```
var nTotal = 35;
var con = 3;
while (con <= nTotal) {
    alert("El contador con es igual a: " + con);
    con++;
}
```

El resultado es el mismo que en el ciclo *for*, aunque su construcción sintáctica es muy diferente. El ciclo comprueba la expresión y continúa la ejecución del código mientras la evaluación sea *true*. El bucle *while* no tiene requiere que la variable de control del ciclo o lazo se modifique dentro del bloque de instrucciones para que en determinado momento se pueda detener la ejecución del lazo.

### El bucle *do ... while*

Esta instrucción es muy similar al ciclo *while*, con la diferencia que en esta, primero se ejecutan las instrucciones y luego, se verifica la condición. Esto significa que, el bloque de instrucciones se ejecutará con seguridad, al menos, una vez. Su sintaxis es:

```
do {
    //bloque de código ;
} while (expresión_condicional);
```

La diferencia con el anterior bucle estriba en que la *expresión condicional* se evalúa después de ejecutar el *bloque de código*, lo que garantiza que al menos una vez se ha de ejecutar, aún cuando la condición sea *false* desde el principio.

```
var nTotal = 35;
var con = 36;
do {
    alert("El contador con es igual a: " + con);
    con++;
} while (con <= nTotal);
```

### Las sentencias *break* y *continue*.

Para agregar aún más utilidad a los bucles, JavaScript incluye las sentencias *break* y *continue*. Estas sentencias se pueden utilizar para modificar el comportamiento de los ciclos más allá del resultado de su expresión condicional. Es decir, nos permite incluir otras expresiones condicionales dentro del bloque de código que se encuentra ejecutando el bucle.

El comando **break** causa una interrupción y salida del bucle en el punto donde se lo ejecute.

```
var nTotal = 35;
var con = 3;
do {
    if (con == 20)
        break;
    alert("El contador con es igual a: " + con);
    con++;
} while (con <= nTotal);
```

En este ejemplo, el bucle se repetirá hasta que la variable **con** llegue al valor 20, entonces se ejecuta la sentencia **break** que hace terminar el bucle en ese punto.

El comando **continue** es un tanto diferente. Se utiliza para saltar a la siguiente repetición del bloque de código, sin completar el pase actual por el bloque de comandos.

```
var nTotal = 70;
var con = 1;
do {
    if ( con == 10 || con == 20 || con == 30 || con == 40)
        continue;
    alert("El contador con es igual a: " + con);
    con++;
} while (con <= nTotal);
```

En el ejemplo, cuando la variable **con** alcance los valores 10, 20, 30 y 40 **continue** salta el resto del bloque de código y comienza un nuevo ciclo sin ejecutar el método `alert()`.

### Sentencia *for ... in*

Este es un lazo o bucle relacionado con objetos y se utiliza para recorrer las diversas propiedades de un objeto. La sintaxis es la siguiente:

```
for (nombredevariable in objeto) {
    //instruccion o bloque de instrucciones;
}
```

### Sentencia *with*

Sintaxis:

```
with(objeto){
    instruccion o bloque de instrucciones;
}
```

La instrucción *with* permite utilizar una notación abreviada al hacer referencia a los objetos. Es muy conveniente a la hora de escribir dentro del código del script un conjunto de instrucciones repetitivas relacionada con el mismo objeto. Por ejemplo, si se tiene el siguiente conjunto de instrucciones:

```
document.write("Hola desde JavaScript.");
document.write("<br>");
document.write("Estás aprendiendo el uso de una nueva instrucción de JavaScript.");
```

Podría escribirse abreviadamente utilizando el *with*, lo siguiente:

```
with(document){
    write("Hola desde JavaScript.");
    write("<br>");
    write("Estás aprendiendo el uso de una nueva instrucción de JavaScript.");
}
```

## Matrices o arreglos

Una matriz es una colección ordenada de valores, donde cada valor individual se denomina elemento y cada elemento es ubicado en una posición numérica dentro de la matriz. Esta posición es conocida como índice.

Tome en cuenta que como JavaScript es un lenguaje débilmente tipificado, los elementos de la matriz pueden ser de cualquier tipo y los distintos elementos de una misma matriz pueden ser también de tipos diferentes. Incluso, estos elementos pueden ser también matrices, lo que permitiría definir una estructura de datos que sea una matriz de matriz.

Un arreglo en JavaScript es tratado como un **objeto especial** llamado **Array**. Los elementos de un arreglo en JavaScript se comienzan a contar desde el elemento cero [0]; es decir, el elemento con índice cero [0] es el primer elemento del arreglo, el elemento con índice uno [1] es el segundo, y así sucesivamente.

Para acceder a los elementos de un arreglo individualmente, debe utilizarse el nombre del arreglo y el índice que indica la posición del elemento deseado entre corchetes ("[" , "]" ). Primero se coloca el nombre y, a continuación, encerrado entre corchetes el índice. Por ejemplo, para acceder al tercer elemento del arreglo llamado miArreglo, debe digitar lo siguiente: miArreglo[2].

### DECLARACIÓN DE MATRICES EN JAVASCRIPT

La declaración de arreglos en JavaScript puede hacerse de dos formas. La primera utilizando corchetes, como se muestra a continuación:

```
var impares = [];
```

La segunda utilizando el constructor Array() de la siguiente forma:

```
var pares = new Array();
```

Cuando se usa el constructor Array() es posible definir el tamaño del arreglo colocando entre paréntesis el número de elementos que tendrá el arreglo con un valor entero. Como se muestra a continuación:

```
var dias = new Array(5);
```

La instrucción anterior crea un arreglo llamado dias y define que el número de elementos de este arreglo será cinco.

### Introducción de elementos en una matriz

Se pueden introducir elementos a un arreglo de varias formas. Entre las que se pueden mencionar:

- Asignando un dato a un elemento del arreglo de forma directa,
- Asignando una lista de valores al arreglo,
- Pasando argumentos al constructor Array()

A continuación, se muestran tres ejemplos de cada una de las formas de introducir elementos en un arreglo de JavaScript:

```
//Asignando un dato a un elemento del arreglo
var dias = [];
dia[0] = "Domingo"; //Al primer elemento del arreglo se le ha asignado el valor Domingo
dia[1] = "Lunes"; //Al segundo elemento del arreglo se le ha asignado el valor Lunes
//Asignando una lista de valores al arreglo
var dias = ["Domingo", "Lunes", "Martes", "Miércoles"];
//Pasando argumentos al constructor Array()
var dias = new Array("Domingo", "Lunes", "Martes", "Miércoles");
```

### Acceso a los elementos de una matriz

Para poder acceder a los elementos de un arreglo es necesario escribir el nombre del arreglo y, a continuación, entre corchetes el índice del elemento. Por ejemplo, si queremos acceder al segundo elemento de un arreglo llamado `dias`. Debemos escribir una instrucción como la siguiente:

```
x = dias[1];
```

La instrucción anterior asigna el valor "Lunes" (suponiendo que ese es el valor almacenado en `dias[1]`) a la variable `x`.

Si se quiere imprimir en la ventana del navegador el valor del sexto elemento de un arreglo, debería escribir una instrucción como la siguiente:

```
document.write("Hoy es " + dias[5] + " 13 de Agosto del 2004");
```

Cuando se intente acceder a un elemento de un arreglo que no ha sido asignado todavía, obtendrá un valor `undefined`.

### Agregar elementos a una matriz

En JavaScript no es necesario asignar más memoria de forma explícita para agregar elementos a un arreglo, aunque inicialmente se haya declarado de un número específico de elementos. Por ejemplo, si se declaró un arreglo con 4 elementos, podemos agregarle dos más sin necesidad de reservar más memoria para dichos elementos. Esto significa que podemos escribir el siguiente script y debería funcionar correctamente:

```
var lenguajes = [4];
lenguajes[0] = "JavaScript";
lenguajes[1] = "Visual Basic";
lenguajes[2] = "PHP";
lenguajes[3] = "C/C++";
document.writeln("Aprenderás a usar los siguientes lenguajes:<br> ");
document.writeln("<OL type='1'>");
document.writeln("<LI>" + lenguajes[0]);
document.writeln("<LI>" + lenguajes[1]);
document.writeln("<LI>" + lenguajes[2]);
document.writeln("<LI>" + lenguajes[3]);
document.writeln("</OL>");
lenguajes[4] = "Java";
lenguajes[5] = "ASP";
document.writeln("Además usarás:<br> ");
document.writeln("<OL type='1' start='5'>");
document.writeln("<LI>" + lenguajes[4]);
document.writeln("<LI>" + lenguajes[5]);
document.writeln("</OL>");
```

Algo que debe tener en cuenta es que en JavaScript no es necesario agregar elementos de forma consecutiva, esto significa que si se tienen cuatro elementos, como en el ejemplo anterior, puede agregar dos elementos más en cualquier posición. Es decir, si vemos el ejemplo anterior, podríamos haber agregado en `lenguajes[7]` y `lenguajes[8]` los elementos "Java" y "ASP", en lugar de hacerlo en `lenguajes[4]` y `lenguajes[5]` como se hizo en el ejemplo.

### Eliminar elementos de una matriz

Para eliminar elementos de un arreglo se usa el operador **delete**. Al utilizar dicho operador JavaScript establece el elemento al valor *undefined*, como que si no tuviera asignado valor alguno. Hay que notar que la propiedad *length* no se modifica al eliminar el elemento del arreglo; es decir, que el arreglo seguirá teniendo el mismo número de elementos que tenía antes de eliminar el elemento.

Ejemplo:

```
var colores = ["rojo", "verde", "azul"];
delete colores[1];
alert("colores[" + 1 + "] = " + colores[1]);
```

Para poder eliminar realmente un elemento para que todos los elementos con índice de posición superior a este se desplacen a posiciones con índice menor, se tienen que utilizar el método de matriz **Array.shift()** para eliminar el primer elemento de la matriz y **Array.pop()** para eliminar el último elemento, y **Array.splice()** para eliminar un rango contiguo de elementos desde una matriz.

Ejemplo:

```
var lugares = ["primero", "segundo", "tercer", "cuarto"];
var primerlugar = lugares.shift(); //El primer elemento es eliminado del arreglo
                                   //y asignado a la variable primerlugar
alert(lugares.toSource());
```

### Obtener el número de elementos de una matriz

Para obtener el número de elementos de un arreglo, o su tamaño se utiliza la propiedad **length**. Esta propiedad lo que obtiene realmente es el índice de la siguiente posición disponible o no ocupada al final del arreglo. Incluso si existen elementos con índices menores no ocupados.

Por ejemplo:

```
var trimestre = new Array("Enero", "Febrero", "Marzo");
alert("El tamaño del arreglo es: " + trimestre.length);
```

El valor mostrado será de tres. Sin embargo, si más adelante decide agregar otros elementos sin tener cuidado del índice último del arreglo ocupado. Puede darse lo siguiente:

```
trimestre[7] = "Julio";
trimestre[8] = "Agosto";
trimestre[9] = "Septiembre";
alert("El tamaño del arreglo es: " + trimestre.length);
```

El nuevo tamaño mostrado será de 10. Por esta razón parece razonable utilizar posiciones adyacentes en el índice de los arreglos para no obtener resultados inesperados en un script.

### Métodos comunes para trabajar con arreglos

JavaScript proporciona algunos métodos para trabajar con arreglos. Algunos de ellos de uso frecuente pueden utilizarse a conveniencia dentro de los scripts que desarrollemos.

#### concat()

Este método devuelve el arreglo que resulta de añadir los argumentos al arreglo sobre el que se ha invocado.

Por ejemplo, las siguientes instrucciones:

```
var miArreglo = ["rojo", "verde", "azul"];
alert(miArreglo.concat("amarillo", "morado"));
```

Mostrarían el siguiente mensaje en un cuadro de alerta:

rojo, verde, azul, amarillo, morado

Debe tomar en cuenta que **concat()** no modifica el arreglo original. Es decir, si se manda a imprimir **miArreglo** solamente se imprimirían los tres colores que le fueron asignados.

#### join()

El método **join()** convierte el arreglo en una cadena y permite al programador especificar cómo se separan los elementos en la cadena resultante. Normalmente, cuando se imprime un arreglo, la salida es una lista de elementos separados por coma. Si se utiliza **join()** se puede establecer el carácter de separación entre elementos. Por ejemplo, en el siguiente código:

```
var miArreglo = ["rojo", "verde", "azul"];
var stringVersion = miArreglo.join(" | ");
alert(stringVersion);
```

Se imprimiría:

rojo | verde | azul

El arreglo original no se destruye como efecto secundario al devolver la cadena de sus elementos enlazada. Si se desea hacer esto deberá asignar al mismo objeto la cadena devuelta. Así:

```
var miArreglo = ["rojo", "verde", "azul"];
miArreglo = miArreglo.join(" | ");
```

### reverse()

Este método invierte el orden de los elementos de un arreglo en particular. Este método si altera el arreglo original, de modo que si se manda a imprimir, nos mostraría los elementos invertidos. Por ejemplo, si se tiene:

```
var miArreglo = ["rojo", "verde", "azul"];
miArreglo.reverse();
alert(miArreglo);
```

La salida será:

azul, verde, rojo

### slice()

Este método devuelve un subarreglo; del arreglo sobre el que se invoca. Como no actúa sobre el arreglo, el arreglo original queda intacto. El método tiene dos argumentos que son el índice inicial y el índice final. Devuelve un arreglo que contiene los elementos desde el índice inicial hasta el índice final, excluyéndolo. Si sólo se proporciona un argumento, el método devuelve los elementos desde el índice dado hasta el final del arreglo. Observe el siguiente ejemplo:

```
var miArreglo = [1, 2, 3, 4, 5];
miArreglo.slice(2);           //Devuelve [3, 4, 5]
miArreglo.slice(1, 3);        //Devuelve [2, 3]
miArreglo.slice(-3);          //Devuelve [3, 4, 5]
miArreglo.slice(-4, 3);       //Devuelve [2, 3]
```

Observe el resultado cuando se utilizan índices negativos y trate de determinar por qué el resultado es el que se muestra en los comentarios.

### splice()

Este método se utiliza para añadir, reemplazar o eliminar elementos de un arreglo particular. Devuelve los elementos que se eliminan. Posee tres argumentos, que se muestran en la siguiente sintaxis:

```
splice(inicio, cantidad_borrar, valores);
```

Donde, inicio es el índice donde se va a realizar la operación, cantidad\_borrar es la cantidad de elementos a eliminar comenzando por el índice de inicio. Si se omite cantidad\_borrar, se eliminan todos los elementos desde el inicio hasta el final del arreglo y a la vez se devuelven. Cualquier argumento adicional presentado en valores (que se separan por comas, si es más de uno) se introduce en el lugar de los elementos eliminados.

### sort()

Es uno de los métodos más útiles utilizados con arreglos en JavaScript. El método clasifica los elementos del arreglo lexicográficamente. Lo hace convirtiendo primero los elementos del arreglo en cadenas y luego los ordena. Esto significa que si ordena números podría obtener resultados inesperados. Vea el siguiente ejemplo:

```
var miArreglo = [14, 52, 3, 14, 45, 36];
miArreglo.sort();
alert(miArreglo);
```

Como el orden es lexicográfico, al ejecutar el script el resultado sería:

14, 14, 3, 36, 45, 52

La función `sort` es muy flexible y permite pasar como argumento una función de comparación que permita ordenar numéricamente y no alfabéticamente los elementos. Las funciones se analizarán en la guía sobre funciones.

### `toString()`

El método `toString()` devuelve una cadena que contiene los valores del arreglo separados por comas. Este método se invoca automáticamente cuando se imprime un arreglo. Equivale a invocar `join()` sin argumentos.

Ejemplo:

```
var numeros = [1, 2, 3, 4, 5];
var letras = ['a', 'b', 'c', 'd'];
alert(numeros.toString()); //Devuelve '1, 2, 3'
alert(letras.toString()); //Devuelve 'a, b, c'
```

## III. MATERIALES Y EQUIPO

Para la realización de la guía de práctica se requerirá lo siguiente:

No.	Requerimiento	Cantidad
1	Guía #2: Estructuras de control: sentencias repetitivas	1
2	Computadora con editor HTML/JavaScript y navegadores instalados	1
3	Memoria USB	1

## IV. PROCEDIMIENTO

**Ejercicio #1: Cálculo del promedio de notas ingresado, solicitando la cantidad de notas a ingresar. En este ejemplo el promedio se calcula como la suma de las notas entre la cantidad de notas ingresadas.**

### Guión 1: promedio.html

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Promedio de notas</title>
  <meta charset="utf-8" />
  <link rel="stylesheet" href="css/font.css" />
  <link rel="stylesheet" href="css/promedio.css" />
  <script src="js/promedio.js"></script>
</head>
<body>

</body>
</html>
```

### Guión 2: promedio.js

```
//Declaración de las variables a utilizar
var i,num,nota,notaactual,promedio;

//Captura de la cantidad de notas que se ingresarán
num = parseInt(prompt("¿Cuántos notas va a ingresar?",""));

//Definiendo un valor de inicialización para la variable que almacenará el promedio
nota = 0;
header = "<header class=\"masked\">";
header += "<h1>Selectable Text</h1>";
```



```
header += "</haeder>";
document.write(header);

//Ciclo o lazo que permitirá ingresar cada nota
document.write("<section>");
document.write("<article>");
for (i=1;i<=num;i++){
    notaactual = parseFloat(prompt("Ingrese la nota " + i + ": ", ""));
    document.write("<h3>La nota " + i + ": " + Math.round(notaactual * Math.pow(10,2)) /
Math.pow(10,2) + "</h3>");
    nota += notaactual;
}

//Obteniendo el cálculo del promedio
promedio = nota / (i-1);

//Imprimir el valor redondeado a dos decimales del promedio
document.write("<h3>El promedio de las notas es: " + Math.round(promedio * Math.pow(10,2)) /
/ Math.pow(10,2) + "</h3>");
document.write("</article>");
document.write("</section>");
```

Indicaciones:

- Crear una carpeta con el nombre js y dentro de esta colocar los archivos presupuesto.js, creado anteriormente.
- Crear una carpeta con el nombre css y dentro de esta colocar los archivos fonts.css y promedio.css, brindado en los recursos de la guía.

Resultado:

The screenshot shows a web application interface. At the top, there is a dialog box with two input fields. The first field is labeled "¿Cuántas notas va a ingresar?" and contains the number "4". The second field is labeled "Ingrese la nota 1:" and contains the number "7.8". Below these fields is a checkbox labeled "Prevenir esta página desde la creación de cuadros de diálogo adicionales". At the bottom of the dialog box are two buttons: "Aceptar" and "Cancelar". Below the dialog box is a main display area with a blue background. It features the text "Selectable Text" in large, bold, blue letters. Below this text, it lists the entered notes: "La nota 1: 7.8", "La nota 2: 8.4", "La nota 3: 7.9", and "La nota 4: 9.2". At the bottom of the display area, it shows the calculated average: "El promedio de las notas es: 8.32".

### Ejercicio #2: Propiedades del navegador.

#### Guión 1: navegador.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Propiedades del navegador</title>
    <meta charset="utf-8" />
    <link rel="stylesheet" href="css/fonts.css" />
    <link rel="stylesheet" href="css/nav.css" />
    <link rel="stylesheet" href="css/tab-x.css" />
    <script src="js/propiedadesnav.js"></script>
</head>
<body>
```

```
</body>
</html>
```

### Guión 5: propiedadesnav.js

```
//Declarar variables e inicializarlas, si es conveniente
var propiedad, str="";
var cantidad = 0;
str += "<header id=\"three-d\">";
str += "<h1>Propiedades del objeto navigator</h1>";
str += "</header>";
str += "<section>";
str += "<article>";
str += "<table class=\"tab-x\" summary=\"Propiedades del navegador\">";
str += "<thead>";
str += "<tr><th>Propiedad</th><th>Valor</th></tr>";
str += "</thead>";
str += "<tbody>";
for(propiedad in navigator){
    str += "<tr>";
    str += "<td>" + propiedad + " </td><td> " + navigator[propiedad] + "</td>";
    str += "</tr>";
    cantidad++;
}
str += "</tbody>";
str += "<tfoot>";
str += "<th colspan=\"2\">";
str += "Total:" + cantidad + " propiedades";
str += "</th>";
str += "</tfoot>";
str += "</table>";
str += "</article>";
str += "</section>";
document.write(str);
```

#### Indicaciones:

- Crear una carpeta con el nombre js y dentro de esta colocar los archivos propiedadesnav.js, creado anteriormente.
- Crear una carpeta con el nombre css y dentro de esta colocar los archivos fonts.css, tab-x.css y nav.css, brindado en los recursos de la guía.
- Crear una carpeta con el nombre fonts y dentro de esta colocar los archivos

chunkfive-webfont.eot, chunkfive-webfont.woff, chunkfive-webfont.svg, chunkfive-webfont.ttf, brindado en los recursos de la guía.

Resultado en el navegador:

Propiedades del objeto navigator	
Propiedad	Valor
cookieEnabled	true
productSub	20030107
vendor	Google Inc.
vendorSub	
maxTouchPoints	10
hardwareConcurrency	4
appName	Mozilla
appName	Netscape
appVersion	5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/44.0.2403.130 Safari/537.36
platform	Win32
product	Gecko
userAgent	Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/44.0.2403.130 Safari/537.36
language	es
languages	es-ES,es,en,fr
onLine	true
getStorageUpdates	function getStorageUpdates() { [native code] }
doNotTrack	null
geolocation	[object Geolocation]
plugins	[object PluginArray]
mimeTypes	[object MimeTypeArray]
webkitTemporaryStorage	[object DeprecatedStorageQuota]
webkitPersistentStorage	[object DeprecatedStorageQuota]
getBattery	function getBattery() { [native code] }
sendBeacon	function sendBeacon() { [native code] }
getGamepads	function getGamepads() { [native code] }
webkitGetUserMedia	function webkitGetUserMedia() { [native code] }

**Ejercicio #3: Encuesta que permite votar y seguir opinando sobre una pregunta.** Cada vez que se emite la opinión se vuelve a preguntar si desea seguir votando. Mientras responda que si desea continuar, se le volverá a pedir la opinión, hasta que responda que ya no. En ese momento se emitirán los resultados. Se utiliza una técnica conocida como ciclo infinito para implementar la solución.

### Guión 1: encuesta.html

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Encuesta sobre ley antimaras</title>
  <meta charset="utf-8" />
  <link rel="stylesheet" href="css/encuesta.css" />
  <link rel="stylesheet" href="css/zui-table.css" />
</head>
<body>
<script src="js/encuesta.js"></script>
</body>
</html>
```

### Guión 4: encuesta.js

```
//Declaración de variables
var voto;
var opcion=true;
var cont1=cont2=cont3=0;
var total;
var per1, per2, per3;
//Mostrar las instrucciones para responder
document.write("<h1 align='center'>Encuesta para determinar cuántas personas están a favor
de la portabilidad numérica de teléfonos celulares.</h1><hr>");
document.write("<b>Digite  \"1\" si esta a favor</b><br>");
document.write("<b>Digite  \"2\" si esta en contra</b><br>");
document.write("<b>Digite  \"3\" si se abstiene de opinar</b><br>");
//Ciclo repetitivo infinito donde se captura voto por voto
//en tanto no se de por terminada el ingreso de respuesta de la encuesta
while(opcion==true){
  voto = parseInt(prompt('¿Cuál es su voto?',''));
  switch (voto){
    case 1:
      cont1++;
      break;
    case 2:
      cont2++;
      break;
    case 3:
      cont3++;
      break;
    default:
      alert('¡Voto no válido!');
  }
  //Se pregunta si se desea terminar la encuesta o continuar
  opcion = confirm('¿Desea continuar s/n?');
}
//Obtener el total de respuestas de la encuesta
total = cont1 + cont2 + cont3;
//Obtener el porcentajes de la primera respuesta
per1 = Math.round((cont1/total) * Math.pow(10,2))/Math.pow(10,2);
```

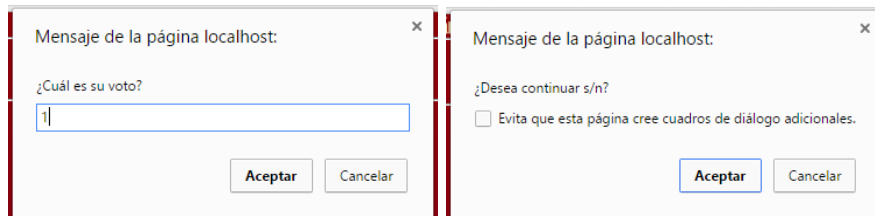
## Guía # 2: Estructuras de control: sentencias repetitivas y matrices

```
//Obtener el porcentajes de la segunda respuesta
per2 = Math.round((cont2/total) * Math.pow(10,2))/Math.pow(10,2);
//Obtener el porcentajes de la tercera respuesta
per3 = Math.round((cont3/total) * Math.pow(10,2))/Math.pow(10,2);
//Mostrar los resultados de la encuesta
with(document){
    write("<table class=\"zui-table zui-table-rounded\">");
    write("<thead>");
    write("<tr><th>Resultado de los votos</th>");
    write("<th>Votos obtenidos</th>");
    write("<th>Porcentaje</th></tr>");
    write("</thead>");
    write("<tbody>");
    write("<tr><td>Votos a favor </td><td class=\"number\">" + cont1 + "</td>");
    write("<td class=\"number\">" + per1*100 + " %</td></tr>");
    write("<tr><td>Votos en contra </td><td class=\"number\">" + cont2 + "</td>");
    write("<td class=\"number\">" + per2*100 + " %</td></tr>");
    write("<tr><td>Se abstienen de opinar </td><td class=\"number\">" + cont3 + "</td>");
    write("<td class=\"number\">" + per3*100 + " %</td></tr>");
    write("</tbody>");
    write("<tfoot>");
    write("<tr><th>Totales</th>");
    write("<th class=\"number\">" + parseInt(cont1+cont2+cont3) + "</th>");
    write("<th class=\"number\">" + (parseFloat(per1+per2+per3))*100 + " %</th>");
    write("</tfoot>");
    write("</table>");
}
```

Indicaciones:

- Crear una carpeta con el nombre js y dentro de esta colocar los archivos encuesta.js, creado anteriormente.
- Crear una carpeta con el nombre css y dentro de esta colocar los archivos encuesta.css y zui-table.css, brindado en los recursos de la guía.

Resultado:



Encuesta para determinar cuántas personas están a favor de la portabilidad numérica de teléfonos celulares.		
Digite "1" si esta a favor Digite "2" si esta en contra Digite "3" si se abstiene de opinar		
Resultado de los votos	Votos obtenidos	Porcentaje
Votos a favor	6	67 %
Votos en contra	2	22 %
Se abstienen de opinar	1	11 %
Totales	9	100%

**Ejemplo #4:** Uso de la sentencia *with* para reducir la cantidad de código a escribir cuando se recorren las propiedades de un objeto.

Guión 1: math.html

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Uso del With</title>
  <meta charset="utf-8" />
  <link rel="stylesheet" href="css/math.css" />
  <script src="js/with.js"></script>
</head>
<body>
</body>
</html>
```

**Guión 3: with.js**

```
//Inicializar variables
var area, peri, coorx, coory;
//Solicitar el valor para el radio del círculo
var radio = parseInt(prompt('Ingrese un numero',''));
document.write("<header>");
document.write("<h1>Área del círculo</h1>");
document.write("</header>");
document.write("<section>");
document.write("<article>");
/* document.write("<div id=\"ballWrapper\">");
document.write("<div id=\"ball\"></div>");
document.write("<div id=\"ballShadow\"></div>");
document.write("</div>"); */
document.write("<div id=\"circle\">");
document.write("</div>");
document.write("<p>");
//Recorrer propiedades del objeto Math usando la instrucción with
with(Math){
  //Área de un círculo de radio "radio"
  area = PI*radio*radio;
  //Valor del lado horizontal definido por el radio
  coorx = abs(radio*cos(PI/4));
  //Valor del lado vertical definido por el radio
  coory = abs(radio*sin(PI/4));
  pericir = 2*PI*radio;
  perirec = 2*coorx + 2*coory;
  //Invocar la propiedad write del objeto documento con with
  with(document){
    write("El área es: " + area);
    write("<br>El lado x del rectángulo generado es: " + coorx);
    write("<br>El lado y del rectángulo generado es: " + coory);
    write("<br>El perímetro del círculo es: " + pericir);
    write("<br>El perímetro del rectángulo es: " + perirec);
  }
}
document.write("</p>");
document.write("</article>");
document.write("</section>");
```

**Indicaciones:**

- Crear una carpeta con el nombre js y dentro de esta colocar los archivos with.js, creado anteriormente.
- Crear una carpeta con el nombre css y dentro de esta colocar los archivos math.css, brindado en los recursos de la guía.

Resultado en el navegador:

The screenshot shows a web application interface. At the top, there is a form with a label 'Ingrese un numero' and a text input field containing the number '7'. Below the input field are two buttons: 'Aceptar' and 'Cancelar'. Below the form is a large rectangular area with a light green background. In the center of this area is a large purple circle. Below the circle, there is a blue rectangular box containing the following text:

Área del círculo

El área es: 153.93804002589985  
El lado x del rectángulo generado es: 4.949747468305833  
El lado y del rectángulo generado es: 4.949747468305832  
El perímetro del círculo es: 43.982297150257104  
El perímetro del rectángulo es: 19.798989873223327

**Ejercicio #5: Creación de campos de formulario del tipo seleccionado en tiempo de ejecución con JavaScript.**

**Primer archivo: forms.html**

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Creación de controles de formulario dinámicamente</title>
  <meta charset="utf-8" />
  <!-- <link rel="stylesheet" href="css/basic.css" /> -->
  <!-- <link rel="stylesheet" href="css/estilosform.css"> -->
  <link rel="stylesheet" href="css/styleform.css" />
  <link rel="stylesheet" href="css/dynamic-form.css" />
  <script src="js/formcontrols.js"></script>
</head>
<body>
<header id="insetBg">
  <h1 class="insetType">Creación de formularios dinámicos</h1>
</header>
<section>
<article id="design">
<form action="javascript:void(0);" action="POST" name="frmconf" id="form" class="contact">
<fieldset class="contact-inner">
<div class="contact-input">
  <label for="selcontrol" class="select">
    <select name="selcontrol" id="selcontrol">
      <option value="" selected="selected">Tipo de control</option>
      <option value="text">Cuadro de texto</option>
      <option value="password">Cuadro de contraseña</option>
      <option value="textarea">Áreas de texto</option>
      <option value="checkbox">Casillas de verificación</option>
      <option value="radio">Botones de opción</option>
      <option value="file">Control de archivo</option>
      <option value="button">Botones genéricos</option>
    </select>
  </label>
</div>
</fieldset>
</form>
</article>
</section>
</body>
</html>
```

```

        </label>
    </div>
    <div class="contact-input">
        <input type="text" name="txtnum" maxlength="2" autofocus placeholder="Número de controles" />
    </div>
    <div class="contact-submit">
        <input type="submit" name="cmdenviar" value="Enviar" />
    </div>
</fieldset>
</form>
</article>
<article id="view"></article>
</section>
</body>
</html>

```

#### Guión 4: formcontrols.js

```

function init(){
    var form = document.getElementById('form');
    form.onsubmit = function(){
        createform(document.frmconf.selcontrol.value,document.frmconf.txtnum.value);
    }
}

function createform(control, numero){
    var htmlform, tag, i;
    //Referenciar al elemento de la página web donde se
    //mostrará el formulario creado
    var formview = document.getElementById('view');
    htmlform = "<div class=\"dynamic-form\">";
    htmlform += "<form name=\"miform\">\n";
    htmlform += "<fieldset>\n";
    htmlform += "<legend><span class=\"number\">1</span> Formulario dinámico</legend>";

    /* html += "<html lang=\"es\">\n<head>\n";
    html += "<title>\nCreación de formularios dinámicos\n</title>\n";
    html += "<meta charset=\"utf-8\" />\n";
    html += "<link rel=\"stylesheet\" href=\"css/estilosform.css\" />\n";
    html += "</head>\n";
    html += "<body>\n";
    html += "<header>\n";
    html += "<h1>Formulario creado desde JavaScript</h1>\n";
    html += "</header>\n"; */
    with(document){
        //Dependiendo del tipo de control
        switch(control){
            case "text":
            case "password":
                for(i=0; i<numero; i++){
                    tag = "<input type=\"\" + control + \"\" name=\"\" + control + (i+1) +
\"\" required placeholder=\"Ingrese los datos en el campo \" + control + \"\" /><br>\n";
                    htmlform += tag;
                }
                break;
            case "textarea":
                for(i=0; i<numero; i++){
                    tag = "<textarea name=\"\" + control + (i+1) + \"\" required
placeholder=\"Ingrese los datos en el campo \" + control + \"\"></textarea><br />\n";
                    htmlform += tag;
                }
                break;
            case "checkbox":

```

```

        for(i=0; i<numero; i++){
            tag = "<div>\n<input type=\"\" + control + "\" name=\"\" + control +
(i+1) + "\" id=\"\" + control + (i+1) + "\" />\n";
            tag += "<label for=\"\" + control + (i+1) + "\">\n";
            tag += control + (i+1) + "</label>\n</div>" + "\n";
            htmlform += tag;
        }
        break;
    case "radio":
        for(i=0; i<numero; i++){
            tag = "<div>\n<label for=\"\" + control + (i+1) + "\">\n";
            tag += "\t<input type=\"\" + control + "\" name=\"\" + control + "\"
id=\"\" + control + (i+1) + "\" />\n";
            tag += "\t<span>" + control + (i+1) + "</span>\n</label>\n</div>" +
"\n";
            htmlform += tag;
        }
        break;
    case "file":
        for(i=0; i<numero; i++){
            tag = "<label class=\"custom-file-input file-blue\"><br />\n";
            tag += "\t<input type=\"file\" name=\"\" + control + (i+1) + "\" /><br
/>\n";
            tag += "</label><br />\n";
            htmlform += tag;
        }
        break;
    case "button":
        for(i=0; i<numero; i++){
            tag = "<button name=\"\" + control + (i+1) + "\">" + control + (i+1) +
"</button><br />\n";
            htmlform += tag;
        }
        break;
    default:
        alert("No ha seleccionado el tipo de control");
        return;
        break;
    }
    htmlform += "</fieldset>\n";
    htmlform += "</form>\n";
}
htmlform += "</div>";
formview.innerHTML = htmlform;
}

window.onload = init;

```

**Indicaciones:**

- Crear una carpeta con el nombre js y dentro de esta colocar los archivos with.js, creado anteriormente.
- Crear una carpeta con el nombre css y dentro de esta colocar los archivos basic.css y estilosform.css, brindado en los recursos de la guía.

Resultado en el navegador:





**Ejercicio #6: Método de ordenación por burbuja, solicitando los valores a ingresar y luego mostrando el listado ingresado y el listado de valores ordenados usando el método de la burbuja.**

**Guión 1: burbuja.html**

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Ordenación por burbuja</title>
  <meta charset="utf-8" />
  <link rel="stylesheet" href="css/burbuja.css" />
</head>
<body>
<div id="datos"></div>
<script src="js/burbuja.js"></script>
</body>
</html>
```

### Guión 3: burbuja.js

```
function inicializar(){
    //Inicialización de variables
    var numeros = new Array();
    var i, max, temp, contenido="", tdelement;
    //Validación para que el número de elementos del arreglo sea
    //numérico y mayor o igual que 2
    do {
        max = prompt("Cuántos números va a ingresar (valor entero):", "");
        //Verificar que se ingrese un dato numérico
        if(isNaN(max)){
            alert("El valor digitado no es numérico.");
            continue;
        }
        //Verificar que el valor ingresado sea mayor o igual que 2
        if(max < 2){
            alert("El arreglo debe ser de dimensión 2 o superior");
        }
    }while(isNaN(max) || max < 2);

    //Lazo para solicitar el ingreso de los elementos del arreglo
    for(i=0; i<max; i++){
        numeros[i] = parseInt(prompt("Número " + (parseInt(i) + 1), ""));
    }
    //Obteniendo el elemento donde se cargará el contenido
    //generado dinámicamente desde JavaScript
    var datos = document.getElementById('datos');
    with(document){
        contenido += "<h1>Números ingresados</h1>\n";
        //Lazo para ingresar los elementos ingresados en el arreglo
        contenido += "<table>\n\t<tbody>\n\t\t<tr>\n";
        //Lazo que muestra los elementos del arreglo en una tabla
        for(i=0; i<max; i++){
            contenido += "\t\t\t<td class=\"Off\">" + numeros[i] + "</td>\n";
        }
        contenido += "\t\t</tr>\n\t</tbody>\n</table>\n<br />\n\n";
        //Lazo que ordena el arreglo mediante el método de la burbuja
        for(i=0; i<max-1; i++){
            for(j=i+1; j<max; j++){
                if(numeros[i]>numeros[j]){
                    temp = numeros[j];
                    numeros[j] = numeros[i];
                    numeros[i] = temp;
                }
            }
        }
        contenido += "<h1>Números ordenados ascendentemente</h1>\n";
        contenido += "<table>\n\t<tbody>\n\t\t<tr>\n";
        //Lazo que muestra los elementos del arreglo que han sido
        //ordenados con el método de la burbuja
        for(i=0; i<max; i++) {
            contenido += "\t\t\t<td class=\"Off\">" + numeros[i] + "</td>\n";
        }
        contenido += "\t\t</tr>\n\t</tbody>\n</table>\n";
    }
    datos.innerHTML = contenido;
    //Capturando los elemento con clase Off
    tdelement = document.getElementsByClassName('Off');
    alert(tdelement.length);
    for(var i=0; i<tdelement.length; i++){
```

```

        tdelement[i].onmouseover = function(){
            this.className = 'On';
        }
        tdelement[i].onmouseout = function(){
            this.className = 'Off';
        }
    }
}

window.onload = inicializar;

```

Indicaciones:

- Crear una carpeta con el nombre js y dentro de esta colocar los archivos burbuja.js, creado anteriormente.
- Crear una carpeta con el nombre css y dentro de esta colocar los archivos burbuja.css brindado en los recursos de la guía.

Resultado en el navegador:

Cuántos números enteros va a ingresar (valor entero):

Aceptar Cancelar

Número 1

Número 2

Número 3

Número 4

Número 5

Número 6

Número 7

Número 8

**Números ingresados**

61	29	56	17	94	35	43	22
----	----	----	----	----	----	----	----

**Números ordenados ascendentemente**

17	22	29	35	43	56	61	94
----	----	----	----	----	----	----	----

**Ejercicio #7: Manejo de dos listas desplegables dependientes con JavaScript.** El ejemplo muestra cómo al cambiar de país, se pueden cambiar las ciudades en el control de lista desplegable dependiente. Además, se ha incorporado la funcionalidad de agregar nuevas ciudades mediante un botón Agregar.

**Guión 1: ciudades.html**

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Interacción con menús de selección</title>
  <meta charset="utf-8" />
  <link rel="stylesheet" href="css/ciudades.css" />
  <link rel="stylesheet" href="css/slick.css" />
  <script src="js/ciudades.js"></script>
</head>
<body>
<header>
  <h1>Selector de destino de vacaciones</h1>
</header>
<section>
<article id="slick">
  <!-- Contact form -->
  <div class="contact-form">
    <!-- Title -->
    <div class="title">País destino</div>
    <!-- Intro text -->
    <p class="intro">
      Seleccione <b>el país</b> al que desea ir:
    </p>
    <!-- Inicio de Contact Form -->
    <div class="w-100">
      <!-- Campos de formulario -->
      <form action="javascript:void(0);" name="testform" id="testform" method="post">
        <!-- Campo de selección del país -->
        <div class="field">
          <select name="country" id="country" class="grayed">
            <option selected="selected" class="disabled">Italia</option>
            <option>Francia</option>
            <option>España</option>
            <option>Estados Unidos</option>
          </select>
          <div id="arrow-select"></div>
          <svg id="arrow-select-svg"></svg>
          <span class="entypo-book icon"></span>
          <span class="slick-tip left">Seleccione el país destino</span>
        </div>
        <!-- Campo de selección de la ciudad destino -->
        <div class="field">
          <select name="city" id="city" class="grayed" size="4">
            <option>Roma</option>
            <option>Turín</option>
            <option>Milán</option>
            <option>Venecia</option>
            <option>Verona</option>
          </select>
        </div>
        <!-- Send button -->
        <input type="button" name="btnagregar" id="btnagregar" value="Agregar ciudad ->"
class="send" />
      </form>
    </div>
  </div>
</article>
</section>
</body>
</html>
```

```

        <!-- / Form fields -->
    </div>
</div>
<!-- / Final del Contact form -->
</article>
</section>
</body>
</html>

```

### Guión 3: ciudades.js

```

function iniciar(){
    //Elementos de la página sobre los que se detectarán eventos
    var selcountry = document.getElementById('country');
    var addcity = document.getElementById('btnagregar');
    //Creando un arreglo para guardar las ciudades de cada país
    var cities = new Array(4);
    cities["Italia"] = ["Roma", "Turín", "Milán", "Venecia", "Verona"];
    cities["Francia"] = ["Paris", "Lion", "Niza", "Mónaco"];
    cities["España"] = ["Madrid", "Barcelona", "Valencia", "Sevilla"];
    cities["Estados Unidos"] = ["Washington", "Florida", "San Francisco", "New York",
    "Houston"];

    //Asociando el manejador de evento click al elemento select country
    selcountry.onclick = function(){
        this.className = this.options[this.selectedIndex].className;
    }

    selcountry.onchange = function(){
        addOptions(cities[this.options[this.selectedIndex].text], document.testform.city);
    }

    //Asociando el manejador de evento click
    addcity.onclick = function(){
        addCity(cities[document.testform.country.options[document.testform.country.selectedIndex].
        text], document.testform.city)
    }
}

//Esta función limpia todas las opciones del menú desplegable de las ciudades
function removeOptions(optionMenu){
    for(i=0; i<optionMenu.options.length; i++){
        optionMenu.options[i] = null;
    }
}

//Esta función agrega nuevas opciones en el menú desplegable
//de las ciudades, dependiendo del país seleccionados.
//Las ciudades para llenar el campo son tomadas del
//array asociativo correspondiente
function addOptions(optionList, optionMenu){
    var i=0;
    //Limpia las opciones
    removeOptions(optionMenu);
    for(i=0; i<optionList.length; i++){
        optionMenu[i] = new Option(optionList[i], optionList[i]);
    }
}

//Permite agregar dinámicamente una ciudad al país actual
function addCity(optionList, optionMenu){
    var newcity;

```

```
do{
    newcity = prompt("Ingrese la ciudad que desea agregar:", "");
}while(newcity == null || newcity == undefined || newcity.length == 0);
optionList.push(newcity);
removeOptions(optionMenu);
addOptions(optionList, optionMenu);
}

window.onload = iniciar;
```

Indicaciones:

- Agregar a nuestro proyecto el archivo ciudades.html que se encuentra en los recursos proporcionados para esta guía.
- Crear una carpeta con el nombre js y dentro de esta colocar los archivos ciudades.js, creado anteriormente.
- Crear una carpeta con el nombre css y dentro de esta colocar los archivos ciudades.css y slick.css brindado en los recursos de la guía.

Resultado:



IMPORTANTE: El archivo slick.css se proporcionará junto a los recursos de la guía de práctica, junto a las fuentes e imágenes de los otros ejemplos de la guía.

## V. DISCUSIÓN DE RESULTADOS

1. Cree un script que permita el ingreso de un número entero y muestre en pantalla la siguiente

- información: 1) Cantidad de cifras, 2) Cantidad de cifras impares, 3) Cantidad de cifras pares, 4) Suma de cifras impares, 5) Suma de cifras pares, 6) Suma de todas las cifras, 7) Cifra mayor, 8) Cifra menor.
2. Cree un script que utilice arreglos que le permita a un vendedor ingresar los precios de los productos que vende. Cada vez que ingrese un nuevo producto y su precio debe generarse una nueva celda de una tabla que va mostrando el producto ingresado y su respectivo precio. Al terminar de ingresar los productos, para lo cual deberá preguntar luego de cada producto ingresado, si se van a ingresar más productos, debe mostrar el total de la venta del día. Asuma que sólo se puede ingresar un solo producto por compra.
  3. En el ejemplo del ordenamiento por el método de la burbuja muestre el ordenamiento del arreglo pero permitiendo que el usuario decida si desea un ordenamiento ascendente o descendente. Para ello solicite al usuario mediante un diálogo de ingreso de datos (prompt) el tipo de ordenamiento. Sólo debe aceptar dos posibles valores: "ascendente" o "descendente". Si no se ingresa ninguna de estas cadenas, muestre un mensaje de error y vuelva a solicitar el ingreso del tipo de ordenamiento. Al final de ingresar todos los números para el arreglo, muéstrelos ordenados en la forma que el usuario lo haya pedido (ascendente o descendente).

### VI. INVESTIGACIÓN COMPLEMENTARIA

1. Investigue para qué se utilizan las siguientes funciones del objeto Math: `abs()`, `round()`, `ceil()`, `floor()`, `exp()`, `log()` y `random()`. Ponga un ejemplo breve, de su utilización.
2. Investigue para qué se utilizan los métodos `push()` y `pop()` en JavaScript utilizando matrices o arreglos. Muestre algún ejemplo sencillo que le ayude a comprender la utilidad de ambas funciones. Realice un único ejemplo que ilustre el funcionamiento de ambas funciones.
3. investigue qué tarea realiza la función matricial `reverse()`. Muestre un pequeño script de ejemplo en donde se ilustre su funcionamiento

### VII. BIBLIOGRAFIA

- Flanagan, David. JavaScript La Guía Definitiva. 1ª Edición. Editorial ANAYA Multimedia / O'Reilly. 2007. Madrid, España.
- Terry McNavage. JavaScript Edición 2012. 1ª Edición. Editorial ANAYA Multimedia / Apress. Octubre 2011. Madrid, España.
- Tom Negrino / Dori Smith. JavaScript & AJAX Para Diseño Web. 6ª Edición. Editorial Pearson – Prentice Hall. 2007. Madrid España.
- Powell, Thomas / Schneider, Fritz. JavaScript Manual de Referencia. 1ra Edición. Editorial McGraw-Hill. 2002. Madrid, España.
- McFedries, Paul. JavaScript Edición Especial. 1ra Edición. Editorial Prentice Hall. 2002. Madrid, España.