



UNIVERSIDAD DON BOSCO

FACULTAD DE INGENIERIA

ESCUELA DE COMPUTACION

Guía de laboratorio Nº 3

Ciclo II

Nombre de la práctica: Estructuras de control: Matrices o arreglos.

Lugar de ejecución: Centro de cómputo ó Virtual.

Materia: Desarrollo de Aplicaciones Web con Software Interpretado en el Cliente.

I. Objetivos

- ☐ Utilizar matrices en la solución de problemas prácticos.
- ☐ Definir matrices unidimensionales y multidimensionales para resolver problemas.
- ☐ Manejar estructuras de control repetitivas para asignar, acceder, eliminar y ordenar los elementos de una matriz.
- ☐ Usar funciones para facilitar el manejo de matrices en JavaScript.

II. Introducción Teórica.

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 #3: Estructuras de control: matrices o arreglos	1
2	Computadora con editor HTML/JavaScript y navegadores instalados	1

IV. Procedimiento.

Ejercicio #1: 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 href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet"
integrity="sha384-QWTKZyjpPEjISv5WaRU90FeRpok6YctnYmDr5pNlyT2bRjXh0JMhY6hW+ALEwIH" crossorigin="anonymous">
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"
integrity="sha384-YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6jIeHh"
crossorigin="anonymous"></script>
  <style>
    .Off {
      background-color: #f8f9fa;
      text-align: center;
      padding: 10px;
    }
    .On {
      background-color: #007bff;
      color: white;
      text-align: center;
      padding: 10px;
    }
  </style>
</head>
<body class="container">
  <div id="resultado" class="mt-5"></div>
</body>
  <script type="text/javascript" src="js/burbuja.js"></script>
</html>
```

Guión 2: burbuja.js

```
// Inicialización de variables
var numeros = [];
var i, j, max, temp;
// 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), ""));
}

// Crear el contenido HTML
var contenido = "<h1>Números ingresados</h1>";
contenido += "<hr>";
contenido += "<table class='\"table table-bordered table-hover\"'><tr>";
for (i = 0; i < max; i++) {
    contenido += "<td class='\"Off\"' onmouseover='\"this.className='On\"'\" onmouseout='\"this.className='Off\"'\">" + numeros[i] + "</td>";
}
contenido += "</tr></table>";

// 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 class='\"mt-5\"'>Números ordenados ascendentemente</h1>";
contenido += "<hr>";
contenido += "<table class='\"table table-bordered table-hover\"'><tr>";
for (i = 0; i < max; i++) {
    contenido += "<td class='\"Off\"' onmouseover='\"this.className='On\"'\" onmouseout='\"this.className='Off\"'\">" + numeros[i] + "</td>";
}
contenido += "</tr></table>";

// Insertar el contenido en el div#resultado
document.getElementById("resultado").innerHTML = contenido;

```

Números ingresados

345	87	3	21
-----	----	---	----

Números ordenados ascendentemente

3	21	87	345
---	----	----	-----

Ejercicio #2: Tabla de colores rgb válidos en HTML, generada a partir de tres arreglos a matrices. Cada una de las matrices representa cada una de las partes del modelo RGB (Red, Green, Blue) en valores hexadecimales seleccionados.

Guión 1: colores.html

```

<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Tabla de Colores Hexadecimales</title>
    <link href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" rel="stylesheet">
    <style>
        .Off {
            background-color: #f8f9fa;
            text-align: center;

```

```

        padding: 10px;
    }
    .On {
        background-color: #007bff;
        color: white;
        text-align: center;
        padding: 10px;
    }
</style>
</head>
<body class="container">
    <div id="resultado" class="mt-5"></div>
</body>
<script src="js/tablaColores.js"></script>
</html>

```

Guión 2: tablaColores.js

```

// Crear tabla de colores hexadecimales
var contenido = "<h1>Tabla de colores hexadecimales</h1>";
contenido += "<hr>";
contenido += "<table class=\"table table-bordered table-hover\" align='center'><tr>";

// Creamos tres arreglos con valores de cadena que representan valores hexadecimales
var ncel = 0;
var r = ["00", "11", "22", "33", "44", "55", "66", "77", "88", "99", "AA", "BB", "CC", "DD", "EE",
"FF"];
var g = ["00", "11", "22", "33", "44", "55", "66", "77", "88", "99", "AA", "BB", "CC", "DD", "EE",
"FF"];
var b = ["00", "11", "22", "33", "44", "55", "66", "77", "88", "99", "AA", "BB", "CC", "DD", "EE",
"FF"];

// Creamos tres bucles anidando uno dentro de otro
for (var i = 0; i < r.length; i++) {
    for (var j = 0; j < g.length; j++) {
        for (var k = 0; k < b.length; k++) {
            // Se crea el color
            var nuevocol = "#" + r[i] + g[j] + b[k];
            if (ncel % 7 == 0) {
                contenido += "<tr>";
            }
            // Se imprime el color
            contenido += "<td style='text-align:center; background-color:" + nuevocol + "'"> " +
nuevocol + "</td>";
            ncel++;
            if (ncel % 7 == 0) {
                contenido += "</tr>";
            }
        }
    }
}
contenido += "</tr></table>";

// Insertar el contenido en el div#resultado
document.getElementById("resultado").innerHTML = contenido;

```

Resultado:

Tabla de colores hexadecimales

#000000	#000011	#000022	#000033	#000044	#000055	#000066
#000077	#000088	#000099	#0000AA	#0000BB	#0000CC	#0000DD
#0000EE	#0000FF	#001100	#001111	#001122	#001133	#001144
#001155	#001166	#001177	#001188	#001199	#0011AA	#0011BB
#0011CC	#0011DD	#0011EE	#0011FF	#002200	#002211	#002222
#002233	#002244	#002255	#002266	#002277	#002288	#002299
#0022AA	#0022BB	#0022CC	#0022DD	#0022EE	#0022FF	#003300
#003311	#003322	#003333	#003344	#003355	#003366	#003377
#003388	#003399	#0033AA	#0033BB	#0033CC	#0033DD	#0033EE
#0033FF	#004400	#004411	#004422	#004433	#004444	#004455
#004466	#004477	#004488	#004499	#0044AA	#0044BB	#0044CC
#0044DD	#0044EE	#0044FF	#005500	#005511	#005522	#005533

Ejemplo #3: Cálculo del promedio de notas de un alumno usando una matriz para almacenar la cantidad de notas que se solicita ingresar junto con el nombre del alumno. Se utiliza una estructura repetitiva para sumar todas las notas y al resultado, se le divide entre la cantidad de notas ingresadas.

Guión 1: promedionotas.html

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Promedio notas</title>
  <meta charset="utf-8" />
  <!-- Incluimos Bootstrap CSS desde un CDN -->
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet"
  integrity="sha384-QWTKZyjpPEjISv5WaRU90FeRpok6YctnYmDr5pNlyT2bRjXh0JMhJY6hW+ALEwIH" crossorigin="anonymous">
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"
  integrity="sha384-YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6jIeHz"
  crossorigin="anonymous"></script>
  <script type="text/javascript" src="js/promedio.js"></script>
  <style>
    /* Estilos adicionales personalizados si es necesario */
    .nota {
      text-align: center;
    }
  </style>
</head>
<body>
</body>
</html>
```

Guión 2: promedio.js

```
// Declarando e inicializando variables
var i, promedio = 0.0;
var alumno, tabla = "";
var notas = [];
var n = parseInt(prompt("¿Cuántas notas va a ingresar?", ""));
alumno = prompt("Ingrese el nombre del alumno ", "");

// Ciclo para capturar las notas
for(i=0; i<n; i++){
  notas[i] = parseFloat(prompt("Ingrese la nota " + (i+1), ""));
}
```



```

tabla += "<div class=\"container\">\n";
tabla += "<table class=\"table table-bordered\">\n";
tabla += "<caption>Evaluaciones y promedio de " + alumno + "</caption>"
tabla += "<thead class=\"thead-dark\">\n<tr>\n<th scope=\"col\">Evaluaciones</th>\n<th scope=\"col\">Notas</th>\n</tr>\n</thead>\n<tbody>\n";
// Obteniendo el promedio y construyendo la tabla
for(i=0; i<notas.length; i++){
    promedio += notas[i];
    tabla += "<tr>\n<td>Evaluación " + (i+1) + "</td>\n<td class=\"nota\">" + notas[i] + "</td>\n</tr>\n";
}
promedio /= notas.length;
promedio = Math.round(promedio * Math.pow(10,2)) / Math.pow(10,2);
tabla += "<tr>\n<th>Promedio</th>\n<td class=\"nota\">" + promedio + "</td>\n</tr>\n";
tabla += "</tbody>\n</table>\n</div>";

// Mostramos la tabla en el documento
document.writeln(tabla);

```

Resultado:

Evaluaciones	Notas
Evaluación 1	10
Evaluación 2	5
Promedio	7.5

Evaluaciones y promedio de Juanito

Ejercicio #4: 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 href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
rel="stylesheet">
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"></script>
    <script src="js/ciudades.js"></script>

```

```

</head>

<body class="bg-light">
  <div class="container-fluid">
    <div class="row">
      <h1 class="display-1 text-center text-dark">Selector de destino de vacaciones</h1>
    </div>
  </div>
  <div class="container">
    <div class="row position-relative bg-white text-center rounded">
      <!-- Title -->
      <p class="display-3">País destino</p>
      <!-- Intro text -->
      <p class="text-dark">
        Seleccione <b class="text-warning">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="form-select m-2">
              <option selected="selected" class="disabled">Italia</option>
              <option>Francia</option>
              <option>España</option>
              <option>Estados Unidos</option>
            </select>
          </div>
          <!-- Campo de selección de la ciudad destino -->
          <div class="field">
            <select name="city" id="city" class="form-select m-2" 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="btn btn-outline-primary m-2" />
        </form>
        <!-- / Form fields -->
      </div>
    </div>
  </div>
</body>

</html>

```

Guión 2: 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"];

  selcountry.onchange = function(){
    addOptions(cities[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);
}

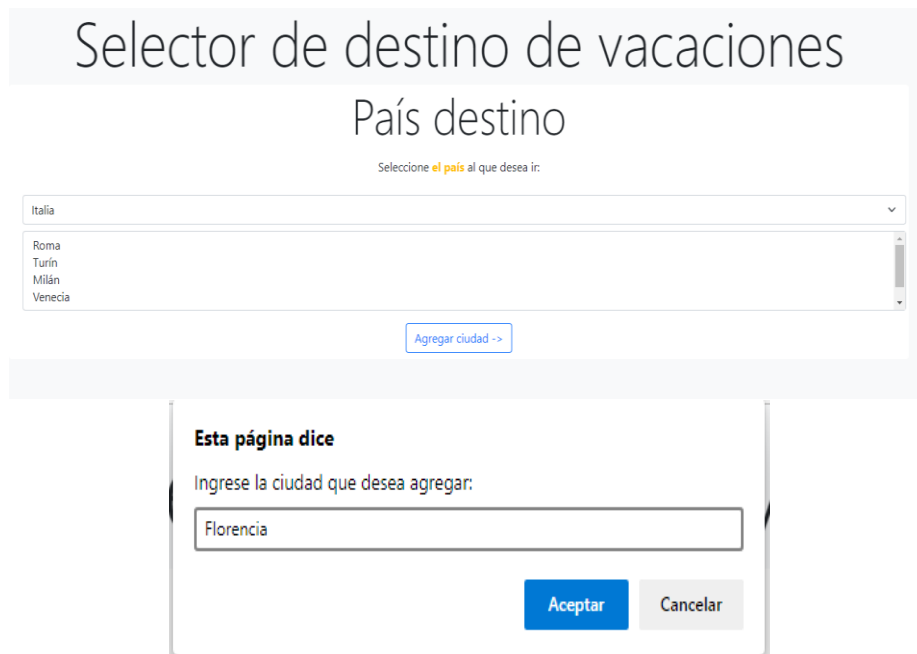
window.onload = iniciar;

```

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:



Selector de destino de vacaciones

País destino

Seleccione el país al que desea ir:

Italia

Islas
Venecia
Verona
Brescia

Agregue ciudad ->

V. Discusión de resultados.

1. 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.
2. Cree una lista de tareas (tarea, descripción y prioridad – prioridad va de 1 a 3), agregue elementos a la lista, utilizando el método prompt de javascript, cada elemento debe de ser ingresado en una matriz de tipo bidimensional y asociativo, al terminar de ingresar las tareas, ordene la matriz por prioridad y luego muestre el resultado.
3. Cree un script que permita implementar el 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, permitiendo que el usuario decida si desea un ordenamiento 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. Bibliografía.

- 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.