APLICACIONES WEB

TEMA 3: PROGRAMACIÓN EN EL LADO DEL CLIENTE

Objetivos

- ✓ Introducir lenguaje de programación JavaScript (JS)
- ✓ Mostrar el Modelo de Objetos del Documento (DOM)
- ✓ Mostrar el modelo de eventos de JS
- ✓ Mostrar la utilización de JS para la validación de formularios

Índice: Temas 3.1 y 3.2

Índice: Temas 3.1 y 3.2

Motivación

- ✓ HTML ofrece una funcionalidad básica para la construcción de páginas web.
- ✓ Para dotar de un mayor dinamismo a las páginas web se puede complementar HTML con otros lenguajes: **los lenguajes de Script**.
- ✓ Los lenguajes de Scripts son lenguajes de programación interpretados con los que se pueden crear secuencias de instrucciones (*scripts*) para insertarlos entre el código HTML de las páginas web.
- ✓ El código de los *scripts* se ejecuta a medida que se encuentra durante la construcción de la página web.
- ✓ Con ese código se pueden dinamizar las páginas, mostrar controles o conectar con bases de datos.
- ✓ Hay varios lenguajes de script: VBScript, JavaScript, PHP, etc…

Ejecución de Scripts

- Las secuencias de órdenes que constituyen un script se pueden ejecutar en el cliente o en el servidor:

- **Programación en el lado del cliente**
  - La secuencia de órdenes se ejecuta en la máquina cliente, aquella en la que se está usando el navegador.
  - Se usa para añadir comportamiento a la página web en la que se encuentra el script.

- **Programación en el lado del servidor**
  - La secuencia de órdenes se ejecuta en el servidor antes de enviar la página web al navegador cliente.
  - Se usa para construir dinámicamente la página con posibles accesos a bases de datos que determinan el contenido de la página.

El lenguaje JavaScript

- ✓ Aquí nos vamos a ocupar de la programación en el lado del cliente utilizando el lenguaje ***JavaScript***.

- ✓ Con JavaScript **se puede**, por ejemplo:
  - Dinamizar las páginas con efectos: texto que aparece y desaparece, animaciones, etc…
  - Controlar el navegador: crear ventanas emergentes, mostrar mensajes al usuario, etc…
  - Validar la información suministrada por el usuario en un formulario.
  - Responder a los sucesos generados al interactuar el usuario con el navegador, etc…

- ✓ Con JavaScript **no se puede**, por ejemplo:
  - Leer o modificar las preferencias del navegador.
  - Acceder a los archivos del ordenador del usuario.
  - Comunicarse con recursos que no pertenezcan al mismo dominio del script.

escribir un archivo

base de datos...

Características (1)

- ✓ Sintaxis basada en el lenguaje Java (¡y ahí acaban las similitudes!).
- ✓ Es un lenguaje interpretado: no se compila. el navegador tiene un interprete que lee el codig
- ✓ Es un lenguaje débilmente tipado: no se define el tipo de las variables, sino que se deduce por el contexto.
- ✓ Distingue las mayúsculas y minúsculas.
- ✓ No se tienen en cuenta los espacios en blanco y las nuevas líneas, por lo que podemos sangrar nuestros scripts para que sean más legibles.
- ✓ No es necesario terminar cada sentencia con el carácter de punto y coma (;), aunque si es recomendable.
- ✓ Se pueden incluir comentarios al estilo C/C++ o Java.

Características (2)

- ✓ Tiene algunas características de orientación a **objetos**: iiiii permite programar funci
  - Elementos que se pueden manipular: ventanas, formularios, elementos de formularios, …
  - Cada objeto es de una clase (tipo de objeto): un objeto es un ejemplar de su clase con nombre único, que tiene propiedades y métodos.
  - **Propiedades (atributos)**: Características de los objetos descritas en sus clases y que conforman el contenido de los objetos, siendo valores sencillos u objetos de otras clases.
  - **Métodos**: Acciones que los objetos pueden realizar y que están descritas en las clases en forma de funciones. Para que un objeto realice una acción, se le pasa un mensaje al objeto con el nombre del método deseado
  - Ya están predefinidas muchas clases de **objetos** con sus propiedades y métodos específicos, cuyo acceso se hace uso de la sintaxis de punto: `window.status` , `window.close()`.

Características (3)

- ✓ También se pueden definir **funciones** propias:
  - Bloques de código que se definen como una unidad independiente con nombre, y que se invocan colocando su nombre seguido de paréntesis.
  - Pueden recibir valores u objetos en forma de argumentos y pueden devolver valores u objetos como resultado de su ejecución.
- ✓ Se puede responder a **eventos**:
  - Sucesos que se producen en el navegador debidos a la interacción del usuario, tales como la pulsación o movimiento del ratón, el envío de los datos de un formulario, etc…
  - A cada evento posible se le asocia un manejador de evento, función con el código a ejecutar cuando se produce el evento: `onClick`, `onMouseover`, `onSubmit`, `...`
  - No todos los eventos han de tener asociado un manejador, sino tan sólo aquellos a los que se quiera responder.

Índice: Temas 3.1 y 3.2

- **¿Dónde colocar scripts en una página HTML?**
  - ✓ En la cabecera de la página (`<head>`): Se ejecutan al abrir la página.
  - ✓ En el cuerpo de la página (`<body>`): Se ejecutan a medida que se encuentran durante la construcción de la página web.
  - ✓ Puede haber múltiples scripts en un archivo HTML.

```
.js
<head>
        <script>
                codigo      alert("Mensaje
               </script>
head>
<body>
        </body>
```

- **¿Cómo se invocan los scripts en una página HTML?**
  - ✓ Usando la etiqueta `<script>`:
    - Permite incluir código de script directamente en la página.
    - Permite incluir una referencia que apunte a un fichero externo con el código.
  - ✓ Usando los atributos de etiquetas HTML que soportan scripts: Manejadores de eventos, enlaces, etc…

Incluyendo código directamente mediante la etiqueta SCRIPT (1)

- El código con las instrucciones JS se puede incluir directamente en cualquier parte del documento HTML encerrándolo entre la etiqueta **<script>**.
- Método utilizado para incluir pequeños bloques de código en la cabecera.
- El intérprete de HTML ignora el contenido del elemento **<script>** y deja que se ocupe de él el intérprete de scripts.
- El navegador sabe qué lenguaje de scripts se utiliza al indicarlo explícitamente en el atributo **language** del elemento **<script>** (normalmente se detecta):

```
language="JavaScript"
```

- Además, para que la página XHTML resultante sea válida, es necesario añadir el atributo **type** a la etiqueta **<script>**:

```
type="text/javascript"
```

Incluyendo código directamente mediante la etiqueta SCRIPT (2)

- ✓ Algunos navegadores no disponen de soporte completo de JS:
  - Se trata de un navegador antiguo.
  - El usuario los ha deshabilitado parcial o completamente.
- ✓ En estos casos, para indicar al usuario de la necesidad de la utilización de JS para el correcto funcionamiento de la página, es posible incluir un aviso mediante la etiqueta **<noscript>**:
  - Normalmente va al comienzo del cuerpo del documento.
  - Su contenido se muestra únicamente en navegadores que no interpreten o tengan JS deshabilitado.

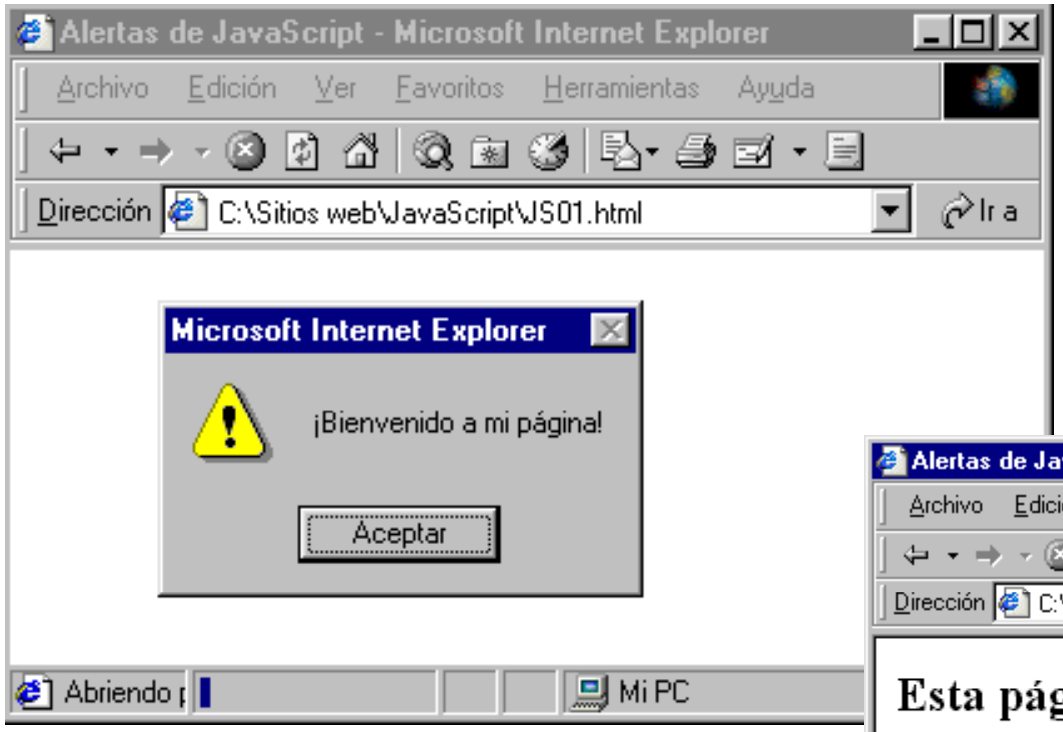
```
<noscript>
   <p>Esta página requiere JavaScript</p>
</noscript>
```

Incluyendo código directamente mediante la etiqueta SCRIPT (3)

- ✓ Ejemplo de inclusión de código JS en un documento HTML: Ventana con alerta.

```
<html>
    <head>
        <title>Ejercicio 1</title>
        <script>
            alert("¡Bienvenido a mi página!")
        </script>
    </head>
    <body>
        <noscript>
            <h1>Esta página requiere JavaScript</h1>
        </noscript>
    </body>
</html>
```

➢ Incluyendo código directamente mediante la etiqueta SCRIPT (4)



Navegador antiguo o con JS deshabilitado

Incluyendo código mediante un fichero externo

- El código con las instrucciones JS se puede incluir en un archivo externo (con extensión .js).
- Los documentos XHTML enlazan con dicho archivo mediante el atributo **src** de la etiqueta **<script>**.
- Se pueden crear todos los archivos JS que sean necesarios, y cada documento XHTML puede enlazar tantos archivos JS como necesite.

```
<head>
  <title>Ejercicio 2</title>
  <script src="js/ejercicio02.js"></script>
</head>
```

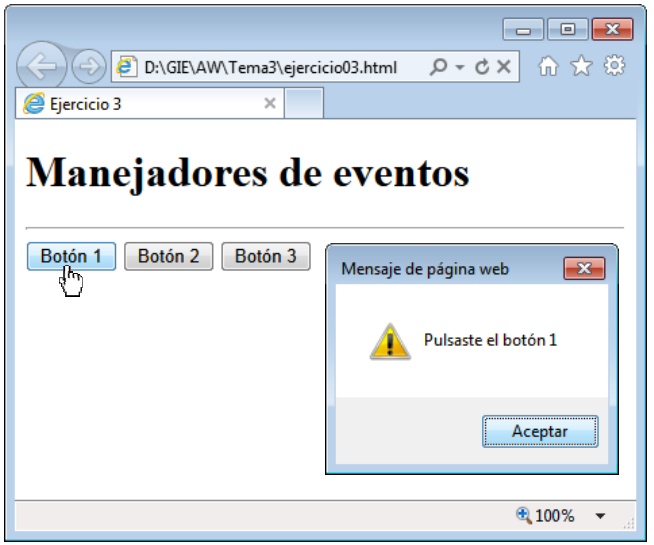
- Archivo con el código en JS: **ejercicio02.js**

```
alert("¡Bienvenido a mi página!")
```

➢ Incluyendo código mediante ciertos atributos (1): Manejador de Eventos

- ✓ El código con las instrucciones JS se puede incluir en un manejador de eventos de determinados elementos HTML.

- ✓ Ejemplo: Evento **onclick** de un botón

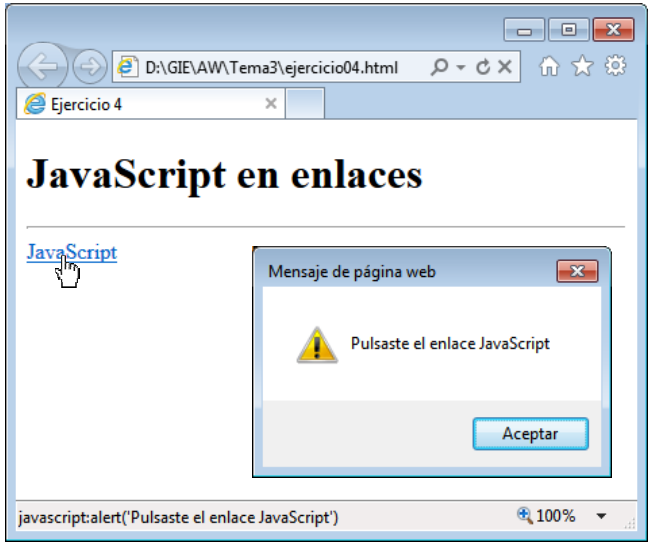



```
<html>
<head>
  <title>Ejercicio 3</TITLE>
</head>
<body>
  <h1>Manejadores de eventos</h1><hr/>
  <form>
     <input type="button" value="Botón 1" onclick="alert('Pulsaste el botón 1')" />
     <input type="button" value="Botón 2" onclick="alert('Pulsaste el botón 2')" />
     <input type="button" value="Botón 3" onclick="alert('Pulsaste el botón 3')" />
  </form>
</body>
</html>
```

➢ Incluyendo código mediante ciertos atributos (2): Enlaces

- ✓ El código con las instrucciones JS también se puede incluir en la URL de un elemento **<a>** precedido por la palabra **javascript**.

- ✓ Ejemplo: Pulsación de un enlace **href**




```
<html>
<head>
  <title>Ejercicio 3</TITLE>
</head>
<body>
  <h1>JavaScript en enlaces</h1><hr/>
  <a href="javascript:alert('Pulsaste el enlace JavaScript')">JavaScript</a>
</body>
</html>
```

Funciones (1)

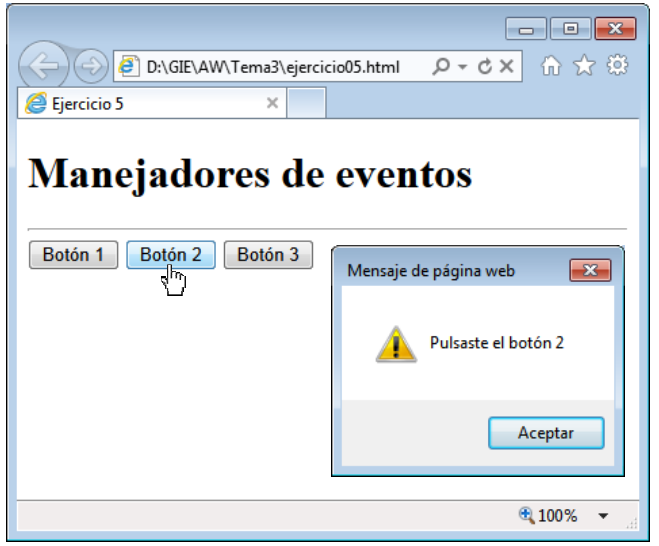
- Las funciones permiten reutilizar código, ejecutándolo tantas veces como se necesite con tan solo invocar a la misma escribiendo su nombre seguido de paréntesis nombre().

```
function nombre() {
  // código de la función
}
```

- El código de la función es una secuencia de instrucciones separadas por punto y coma (;).
- Se pueden proporcionar datos a las funciones en forma de parámetros incluyendo los correspondientes argumentos en la llamada, entre los paréntesis y separados por comas.
- Las funciones también pueden devolver un resultado.

Funciones (2)

- Ejercicio 3 implementado con una función:




```
<html>
<head>
  <title>Ejercicio 3</TITLE>
  <script>
     function myfunc(boton) {
        alert('Pulsaste el botón ' + boton);
     }
  </script>
</head>
<body>
  <h1>Manejadores de eventos</h1><hr/>
  <form>
     <input type="button" value="Botón 1" onclick="myfunc(1)" />
     <input type="button" value="Botón 2" onclick="myfunc(2)" />
     <input type="button" value="Botón 3" onclick="myfunc(3)" />
  </form>
</body>
</html>
```

Índice: Temas 3.1 y 3.2

Variables (1)

- ✓ Se definen mediante la palabra reservada **`var`**: `var nombre;`
- ✓ El nombre de una variable es su identificador. Existen tres normas que hay que cumplir al escoger dicho nombre:
  - Un nombre de variable válido se compone únicamente de una palabra (sin espacios), pudiendo estar formada por letras, números o los símbolos $ (dólar) y _ (guión bajo).
  - El primer carácter del nombre de la variable no puede ser un número.
  - El nombre de la variable no debe coincidir con ninguna de las palabras reservadas.
- ✓ **Ámbito de las variables**: Siempre que se declare una variable local a un ámbito (a una función, por ejemplo), dominará sobre cualquier otra variable que se hubiese declarado fuera de ese ámbito. Además, una variable declarada en una función no puede ser accedida desde otra función. Las variables declaradas fuera de una función son globales y pueden ser accedidas en el interior de la misma.

Variables (2)

- JavaScript es un lenguaje sin tipos (no necesita una declaración del tipo). Los principales tipos de datos que puede almacenar una variable en JS son:

- **Números:**

  - **Enteros**: Cualquier sucesión de dígitos (0 a 9), precedidos por el signo menos (-) si es un número negativo.

```
var iva = 16;            // variable tipo entero
```

  - **En coma flotante**: Se representan de igual forma que los números enteros, pero añaden una parte decimal que se representa con un punto (.) seguido de tantas cifras como compongan la parte decimal del número.

```
var total = 234.65;    // variable tipo decimal
```

Variables (3)

- **Booleanos:** Representan únicamente dos valores, verdadero (true) o falso (false), y se suelen utilizar en sentencias condicionales.

```
var clienteRegistrado = false;
```

- **Cadenas de texto:** Se delimitan por comillas dobles (") o simples (') y pueden contener cualquier combinación de números, letras, espacios y símbolos).

```
var mensaje = "Bienvenido a nuestro sitio web";
```

- **Arrays:** Son secuencias de elementos accesibles por su posición.

```
var dias = ["Lunes", "Martes", "Miércoles", "Jueves"];

var diaSeleccionado = dias[0]; // diaSeleccionado = "Lunes"
```

Operadores

+	Suma o concatenación	/	División	++	Incremento
-	Resta	%	Módulo	--	Decremento
*	Multiplicación	-	Cambio de signo		*Como prefijo o sufijo*

Asignaciones

= Asigna el valor de la derecha a la variable de la izquierda. El valor puede ser el resultado de evaluar una expresión.

Algunas expresiones de asignación se pueden abreviar: `x += y` es igual que `x = x + y`

Comparaciones

==	Igual que (===)	<=	Menor o igual que	&&	And
!=	Distinto de (!==)	>	Mayor que	||	Or
<	Menor que	>=	Mayor o igual que	!	Not

Comentarios

De usa sola línea (`//` ...) o de varias líneas (`/*` ... `*/`), igual que en C++ o Java.

Índice: Temas 3.1 y 3.2

Objetos en JavaScript (1)

- ✓ JavaScript es un lenguaje basado en objetos (≠ orientado a objetos).
- ✓ En JavaScript un **objeto** es un conjunto de valores (de cualquier tipo) con nombre denominados **propiedades** o atributos. Cuando las propiedades contienen un valores de tipo función, estas reciben el nombre de **métodos**.
- ✓ Para acceder a las propiedades o métodos de un objeto se emplea el operador punto (.). Por ejemplo, en el siguiente código se accede a la propiedad `status` y al método `alert()` del objeto `window`:

```
window.status = "Bienvenido a JavaScript";
window.alert("Bienvenido a mi página");
```

- ✓ También se puede acceder mediante la notación de arrays asociativos:

```
window["status"] = "Bienvenido a JavaScript";
window["alert"]("Bienvenido a mi página");
```

Objetos en JavaScript (2)

- ✓ Existen dos formas de **crear objetos**:
  - Inicializadores de objeto:

```
nombreObjeto = {prop1:val1, prop2:val2, ..., propN:valN};
```

  - Funciones constructoras:

```
function ObjConstructor(arg1, arg2, ..., argN) {
   this.prop1 = arg1;
   this.prop2 = arg2;
   ...
   this.propN = argN;
}
objeto = new ObjConstructor(val1, val2, ..., valN);
```

Este método es mejor que el anterior, ya que permite crear un tipo para distintos objetos.

```
objeto1 = new ObjConstructor(val11, val12, ..., val1N);
objeto2 = new ObjConstructor(val21, val22, ..., val2N);
```

Objetos en JavaScript (3)

✓ Métodos:

- Para definir un método de un objeto, simplemente hay que asignar a una propiedad del objeto el nombre de una función. En la función, si se quiere acceder a las propiedades del objeto, se tiene que emplear la palabra reservada **this**.

```
function fun() { ... }

function ObjConstructor(arg, fun) {
   this.prop = arg;
   this.meth = fun;
}
objeto = new ObjConstructor(arg, fun)
objeto.fun();
```

✓ Eliminación de objetos:

- Para eliminar un objeto se emplea el operador **delete**. Este operador devuelve **true** si el borrado es correcto y false en caso contrario: `objeto.delete();`

Objetos en JavaScript (4)

- Además de los tipos de datos simples y objetos definidos por el usuario, JavaScript posee diferentes clases de objetos con sus métodos correspondientes:

- **Objetos del lenguaje:**

`Array`	Para crear colecciones (no es necesario reservar memoria).
`Date`	Relacionado con la fecha y la hora.
`Math`	Valores y funciones matemáticas.
`String`	Para la manipulación de cadenas de caracteres.
`RegExp`	Expresiones regulares.

a var mi=[12357910]

function myfunc(mi){
var miarai2;
miarai.sort()
return miarai. elegir

no entra

- **Objetos del navegador:**

Permiten manejar aspectos de la presentación de las páginas en el navegador.

El objeto Array (1)

- ✓ Los objetos de la clase **Array** se utilizan para trabajar con secuencias de elementos. Se pueden crear de tres formas diferentes:
  - Creación simple:

```
var array1 = new Array();
```

  - Creación con tamaño:

```
var array2 = new Array(10);
```

  - Creación e inicialización:

```
var array3 = new Array("elemento1", "elemento2");
```

- ✓ Se puede acceder a sus elementos mediante su posición (comienza en 0):

```
elemento = array3[1];   // elemento= vale "elemento2"
```

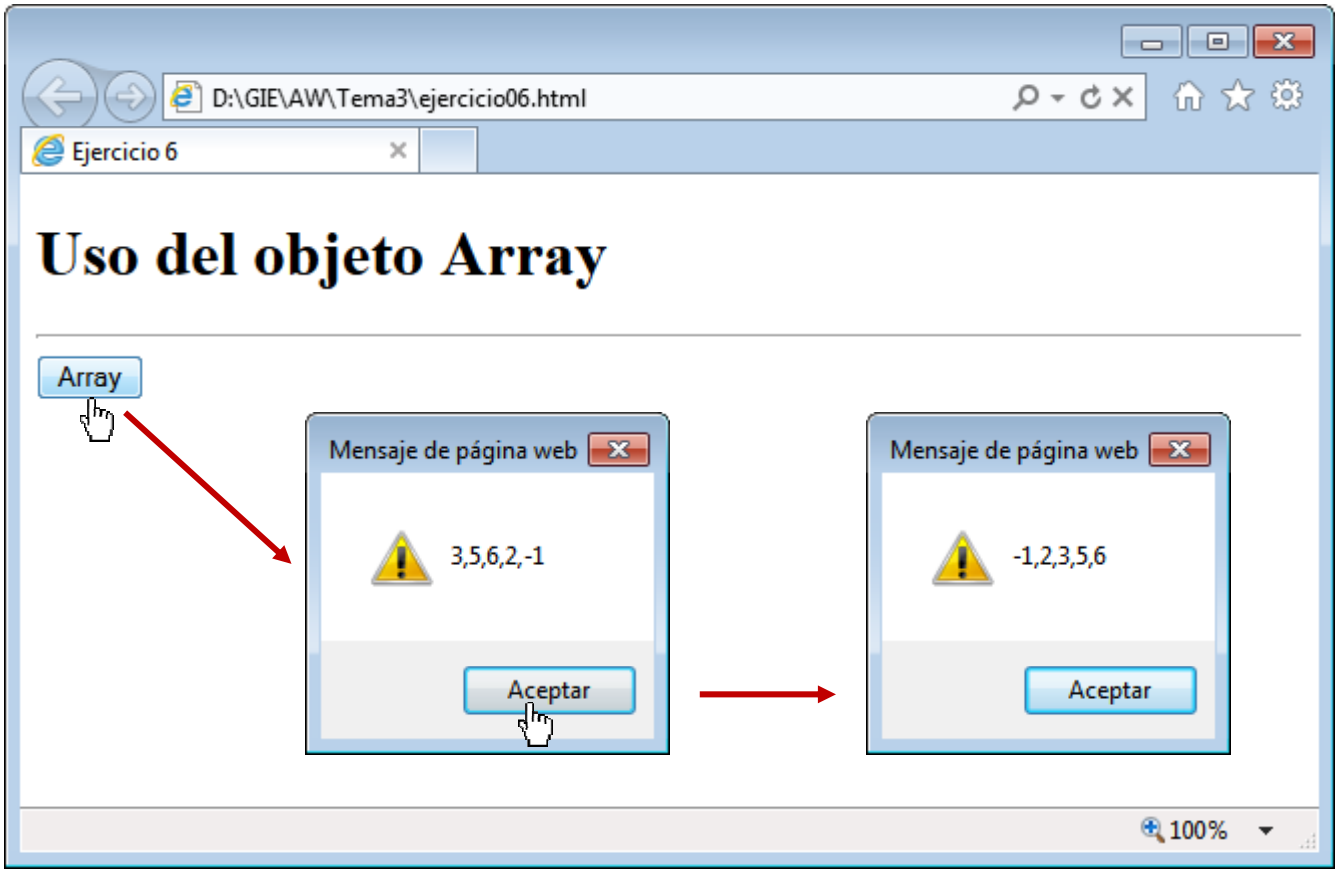
El objeto Array (2)

- Algunos de los principales métodos de los objetos de la clase **Array** son:

`concat(valor1,...)`	Añade nuevos valores array y devuelve el nuevo array (el array sobre el que se realiza la llamada no se modifica).
`join(separador)`	Crea una cadena de texto de los elementos que contiene el array, separándolos con 'separador'.
`pop()`	Elimina el último elemento.
`push(valor1,...)`	Añade nuevos valores al array.
`reverse()`	Invierte el orden de los elementos de un array.
`shift()`	Elimina y devuelve el primer elemento de un array.
`slice(start,end)`	Devuelve un nuevo array con los elementos entre la posición 'start' y 'end'.
`sort()`	Ordena los elementos del array.

El objeto Array (3)

- Utilizar el método `sort()` para ordenar el array [3,5,6,2,-1]:



El objeto Date (1)

- ✓ Los objetos de la clase **Date** se utilizan para trabajar con fechas y horas. Se pueden crear de cuatro formas diferentes:
  - Fecha y hora actuales:

```
var tiempo = new Date();
```

  - Creación como milisegundos desde 01/01/1970:

```
var tiempo = new Date(mili);
```

  - Fecha específica:

```
var tiempo = new Date(año,mes,dia);
```

  - Fecha y hora específicas:

```
var tiempo = new Date(año,mes,dia,hora,minuto,segundo,mili);
```

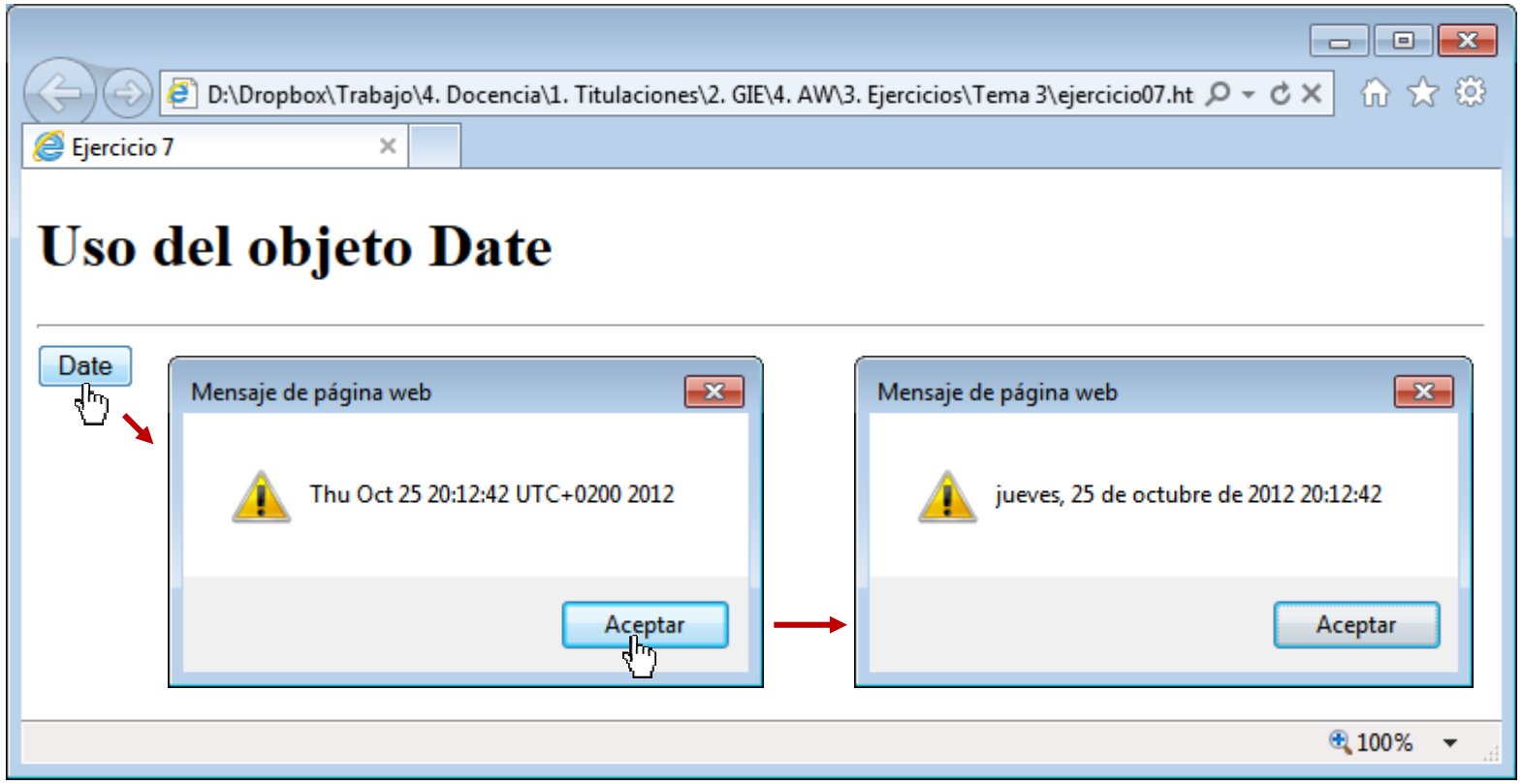
El objeto Date (2)

- Algunos de los principales métodos de los objetos de la clase **Date** son:

`getDate()`	Obtiene el día del mes de la fecha (1-31).
`getDay()`	Obtiene el día de la semana (0=domingo ... 6= sábado).
`getFullYear()`	Obtiene el año (4 dígitos).
`getHours()`	Obtiene la hora (0-23).
`getMonth()`	Obtiene el mes (0-11).
`setDate(día)`	Establece día del mes (1-31).
`setFullYear(año)`	Establece el año (4 dígitos).
`setHours(hora)`	Establece la hora (0-23).
`setMonth(mes)`	Establece el mes (0-11).
`setSeconds(segundos)`	Establece los segundos (0-59).
`toLocaleString()`	Cadena que representa la fecha.

El objeto Date (3)

- Utilizar el método `toLocaleString()` para mostrar la fecha y hora como una cadena de texto con el formato local:



El objeto Math (1)

- ✓ Clase **Math** posee varias funciones para llevar a cabo cálculos matemáticos.
- ✓ No se pueden crear objetos de la clase.
- ✓ Ejemplo: `Math.round(numero)`
- ✓ Algunas propiedades:
  - **E** Es el número e (la constante de Euler). `Math.E`
  - **LN2** Es el logaritmo neperiano (natural) de 2.
  - **LN10** Es el logaritmo neperiano (natural) de 10.
  - **LOG2E** Es el logaritmo en base 2 del número e.
  - **LOG10E** Es el logaritmo en base 10 del número e.
  - **PI** Es el número Pi.

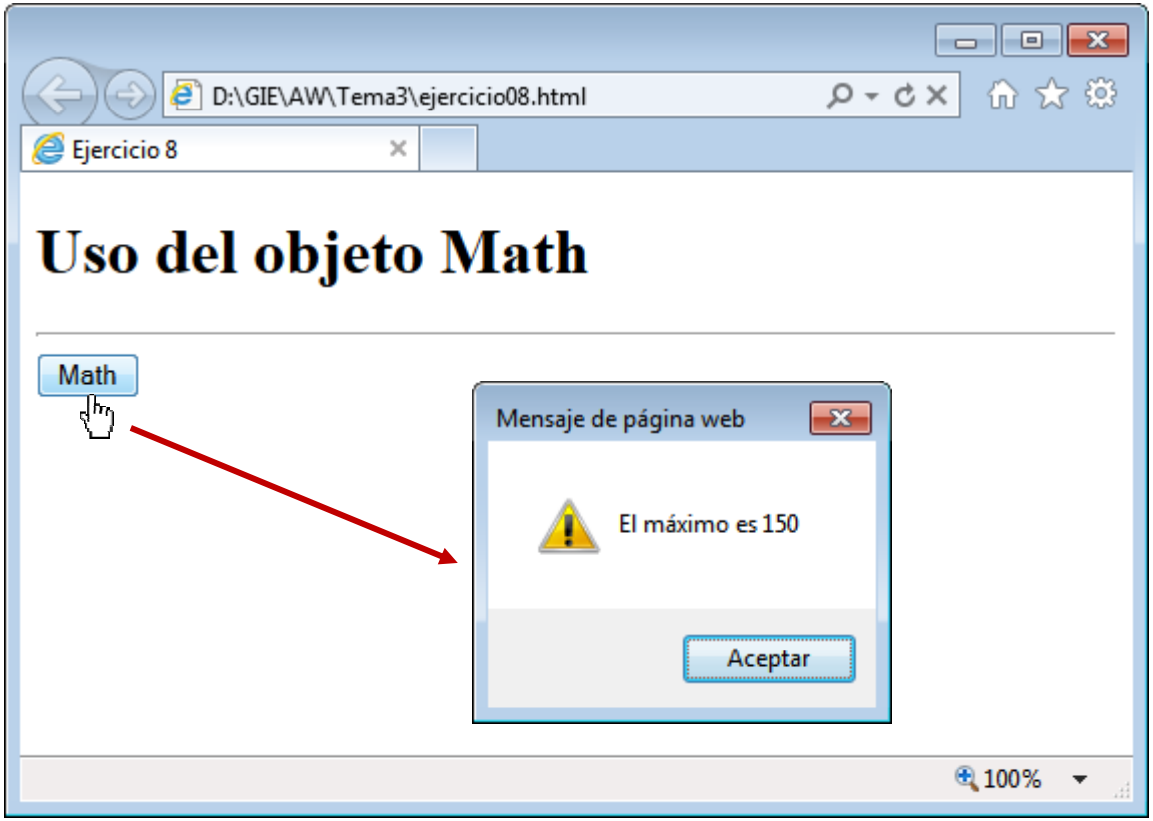
El objeto Math (2)

- Algunas de las principales funciones de la clase **Math** son:

`abs(número)`	Valor absoluto.
`ceil(número)`	Número redondeado al entero superior.
`cos(número)`	Coseno.
`exp(exponente)`	Número e elevado al exponente.
`floor(número)`	Número redondeado al entero inferior.
`log(número)`	Logaritmo neperiano.
`max(número1,número2)`	Mayor de los dos números.
`pow(base,exponente)`	Base elevada al exponente.
`random()`	Un número pseudoaleatorio entre 0 y 1.
`round(número)`	Redondeo al entero más próximo.
`sqrt(número)`	Raíz cuadrada del número.

El objeto Math (3)

- Utilizar la función `max()` para calcular el máximo de los valores [0,150,30,20,38]:



El objeto String (1)

- ✓ Los objetos de la clase **String** se utilizan para al almacenamiento y manipulación de texto. Se pueden crear de tres formas diferentes:
  - Creación como objeto:

```
var cadena = new String();
```

  - Creación como objeto con contenido inicial:

```
var cadena = new String("Contenido inicial");
```

  - Creación como simple cadena:

```
var cadena = "Contenido inicial";
```

- ✓ La longitud de la cadena se encuentra en la propiedad **length**:

```
longitud = cadena.length;        // longitud vale 17
```

El objeto String (2)

- Algunas de las principales funciones de la clase **String** son:

`charAt(pos)`	Carácter de la cadena en la posición *pos.*
`indexOf(cad,[inicio])`	Posición de la subcadena *cad* desde inicio.
`concat(c1,c2, ...)`	Concatena las cadenas *c1*, *c2* …
`slice(p1,p2)`	Subcadena entre las posiciones *p1* y *p2*-1.
`split(separador)`	Separa la cadena en un array de subcadenas.
`toUpperCase()`	Convierte la cadena a mayúsculas.

- Al ser arrays de caracteres, el acceso a sus elementos se puede hacer por posición:

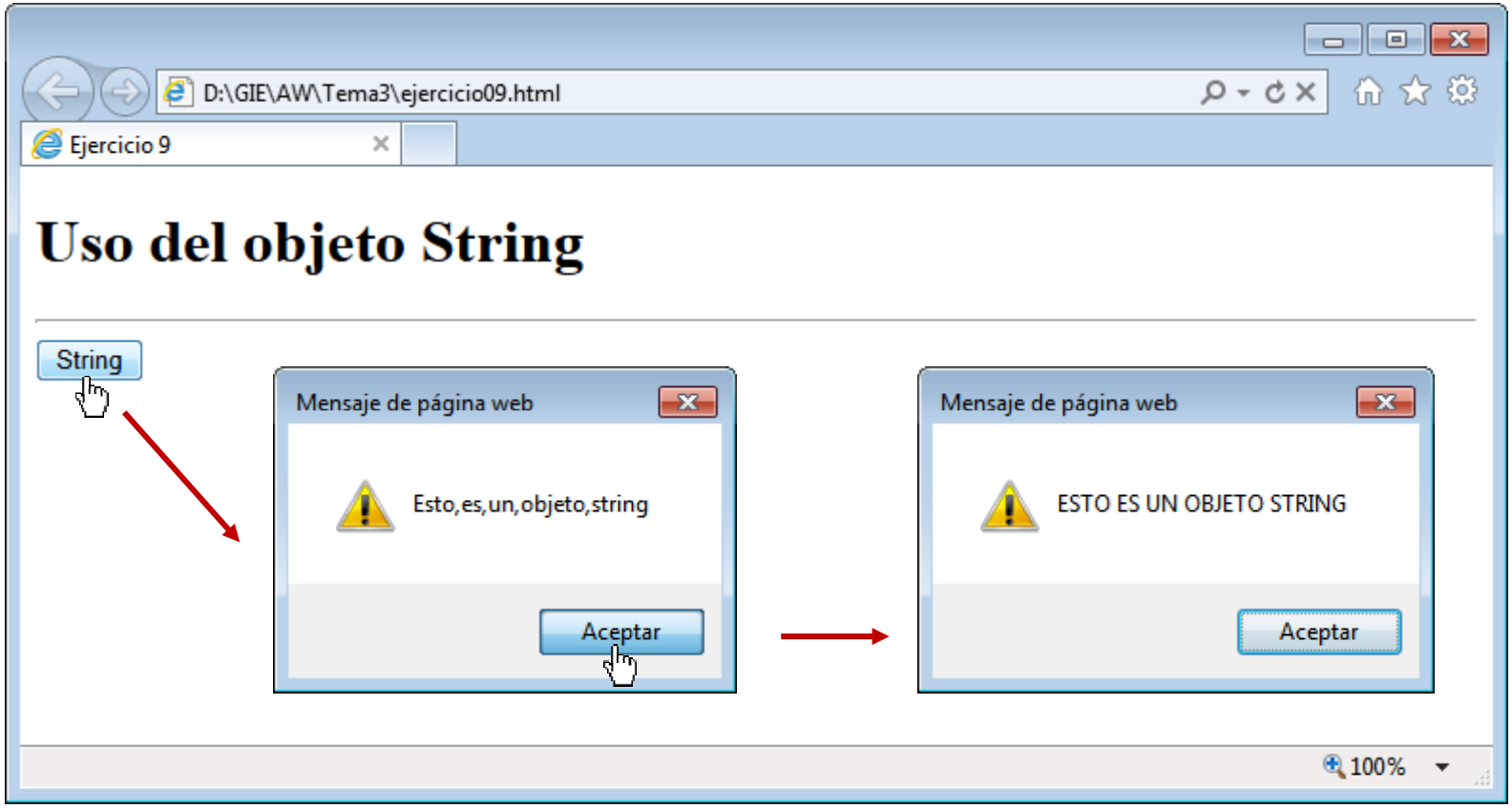
```
caracter = cadena[3];   // carácter vale "t"
```

- Las cadenas se pueden concatenar con el operador `+`:

```
cad3 = cad1 + cad2;
```

El objeto String (3)

- Utilizar los métodos `split()` y `toUpperCase()` para separar y pasar a mayúsculas respectivamente la cadena de texto “Esto es un objeto string”:



El objeto RegExp (1)

- ✓ Los objetos de la clase **RegExp** se utilizan el trabajo con expresiones regulares. Las Expresiones Regulares consisten en patrones que describen conjuntos de cadenas de caracteres. Ejemplo: *.exe representa todos los archivos ejecutables.
- ✓ Se pueden crear de dos formas diferentes:
  - Creación como objeto:

```
var patron= new RegExp("patron","modificadores");
```

  - Creación como simple cadena:

```
var patron = /patron/modificadores;
```

- ✓ El patrón representa el patrón que debe seguir la cadena que se busca y tiene su propia sintaxis: expresiones con cochetes, metacaracteres, cuantificadores, etc…
- ✓ Los modificadores hacen la búsqueda es sensible a mayúsculas, global, etc…

El objeto RegExp (2)

- Por ejemplo, la siguiente expresión regular puede utilizarse para buscar las palabras azul o roja dentro de una frase:

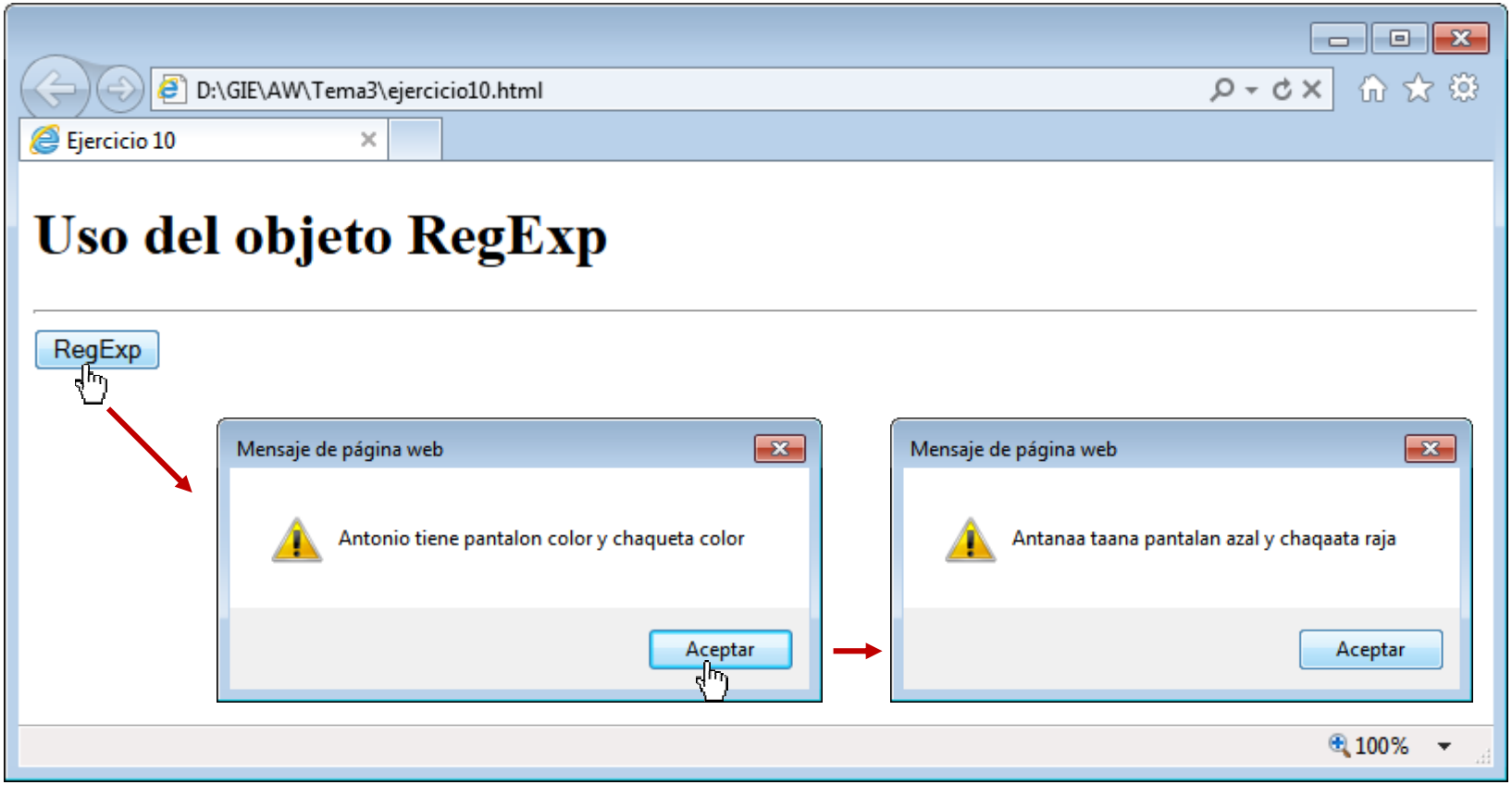
```
var patron= new RegExp("(azul|roja)","g");
var patron = /(azul|roja)/g;
```

- Una vez generada la expresión regular, esta puede ser utilizada en algunos métodos de las cadenas como:

`match(patron)`	Devuelve las subcadenas que cumplen el patrón.
`search(patron)`	Devuelve las posición de la primera coincidencia.
`split(patron)`	Utiliza el patrón como separador en subcadenas.
`replace(patron,cad)`	Reemplaza por *cad* las subcadenas que cumplen el patrón

El objeto RegExp (3)

- Utilizar la función `replace()` para transformar la frase “Antonio tiene pantalon azul y chaqueta roja” en las siguientes frases:



Índice: Temas 3.1 y 3.2

Bloques de código

- Cuando en una rama de ejecución de una instrucción condicional o en un bucle se han de ejecutar varias instrucciones, la secuencia de instrucciones ha de ir encerrada entre llaves, de forma que se cree un bloque de código como una unidad de ejecución.
- Un bloque de código no es más que una secuencia de instrucciones encerrada entre llaves:

```
if(n<100) {
    window.resizeBy(10,10);
    window.moveBy(30,0);
}
```

Bloque de código

Instrucciones de control típicas

- Las principales estructuras para el control del flujo en JavaScript se pueden clasificar, al igual que en otros lenguajes, en los dos grupos siguientes:

- **Instrucciones condicionales o bifurcaciones:**
  - Bifurcación condicional: `if`
  - Ramificación múltiple: `switch`

- **Instrucciones de repetición o bucles:**
  - Bucle definido: `for` (Iterador: `for..in`)
  - Bucles no definidos: `while` / `do..while`

Instrucción IF (1)

- La sintaxis de la sentencia de bifurcación condicional **`if`** tiene varias posibilidades:

```
if(condición) instrucción;
```

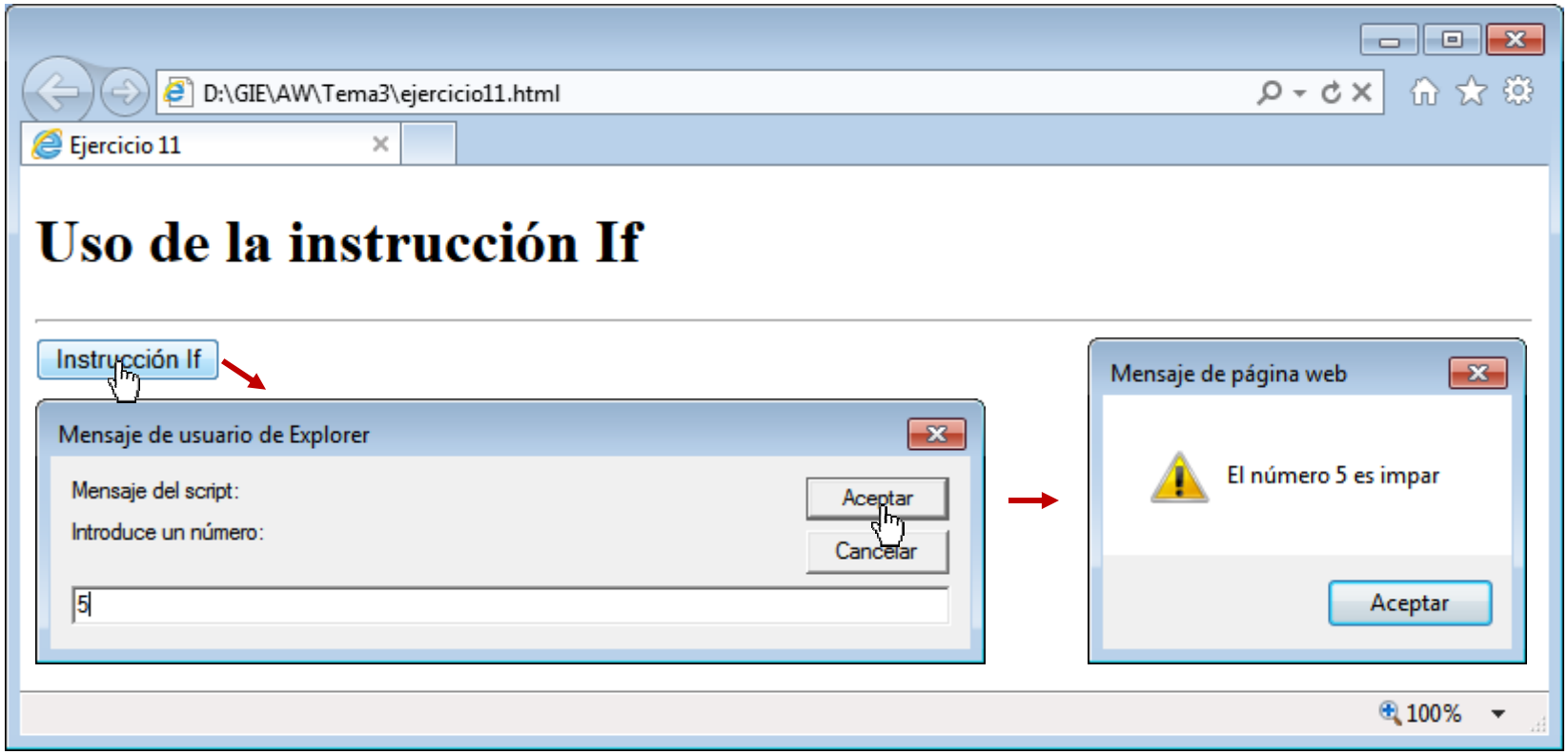
```
if(condición) instrucción-if;
else instrucción-else;
```

```
if(condición) {
   secuencia-de-instrucciones;
}
```

```
if(condición) {
   secuencia-de-instrucciones-if;
}
else {
   secuencia-de-instrucciones-else;
}
```

Instrucción IF (2)

- Hacer un programa que pida un número al usuario y muestre un mensaje diciendo si es par o impar. AYUDA: Mediante la función **`prompt()`** del objeto **`window`** se puede pedir un dato al usuario.



Instrucción SWITCH (1)

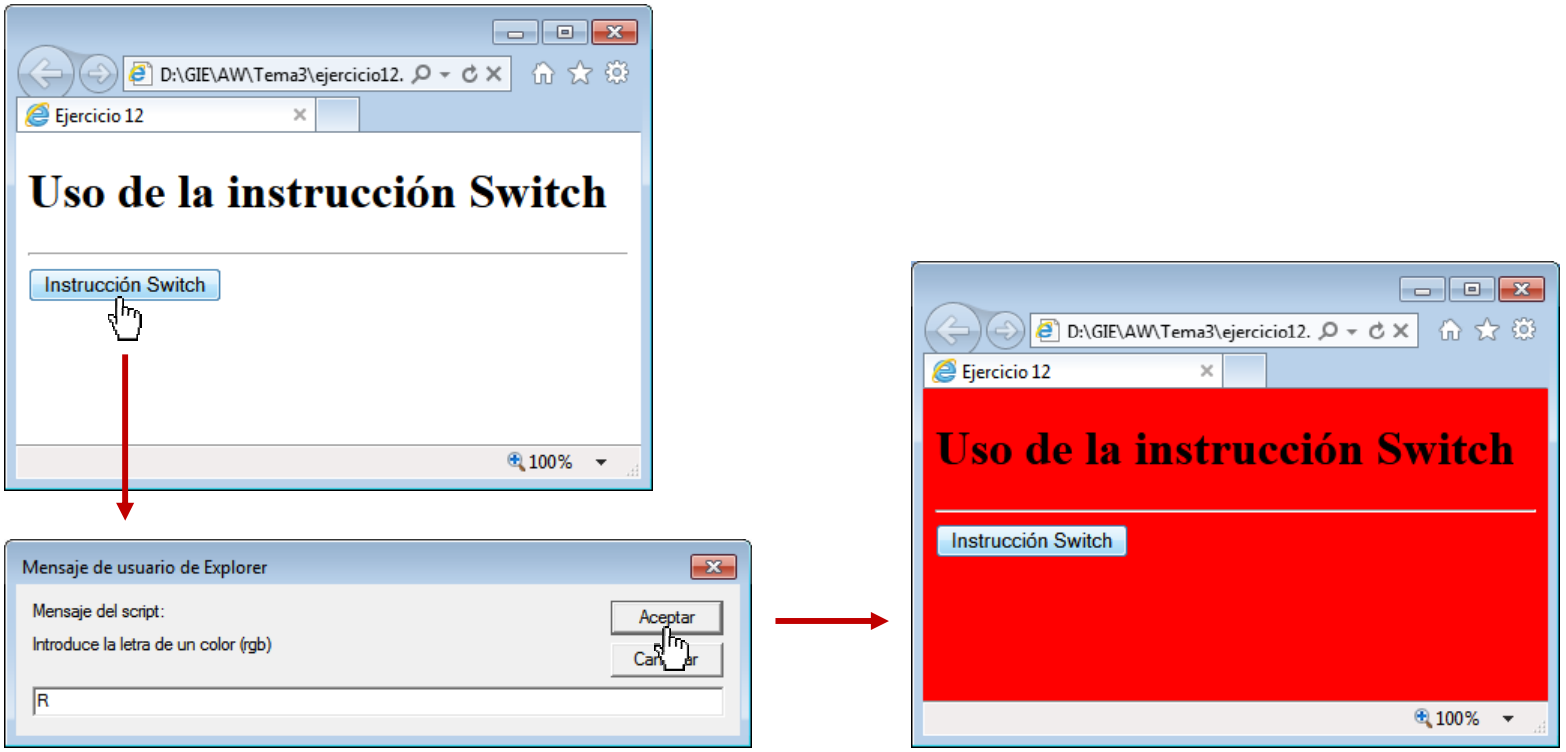
- La sentencia de selección múltiple **switch** posee la siguiente sintaxis:

```
switch(expresión) {
    case valor1:
        instrucciones1
        [break;]
    case valor2:
        instrucciones2
        [break;]
    ...
[   default:
        instrucciones-por-defecto ]
}
```

[...] : opcional

➢ Instrucción SWITCH (2)

✓ Hacer un programa que modifique el color de fondo de la web en función del color solicitado al usuario (blanco por defecto). AYUDA: El color de fondo puede establecerse modificando la propiedad **`bgColor`** del objeto **`document`**.



Instrucción FOR (1)

- La sentencia de repetición **for** posee la siguiente sintaxis:

```
for(inicialización;condición;incremento)
  instrucción;
for(inicialización;condición;incremento) {
  secuencia-de-instrucciones
}
```

- La sección de inicialización sirve para establecer la situación inicial.
- La sección de condición indica si debe proseguir la repetición; mientras sea cierta el bucle continuará.
- La sección de incremento sirve para establecer la nueva situación de la siguiente iteración.

Instrucción FOR (2)

- La sentencia de repetición **`for..in`** sirve para procesar todos los elementos de una colección y posee la siguiente sintaxis:

```
for(variable in colección) instrucción;
for(variable in colección) {
  secuencia-de-instrucciones
}
```

- La colección puede ser un array o el conjunto de propiedades de un objeto:
  - Si se trata de un array, la variable de control es una variable entera que va tomando los valores de posición:

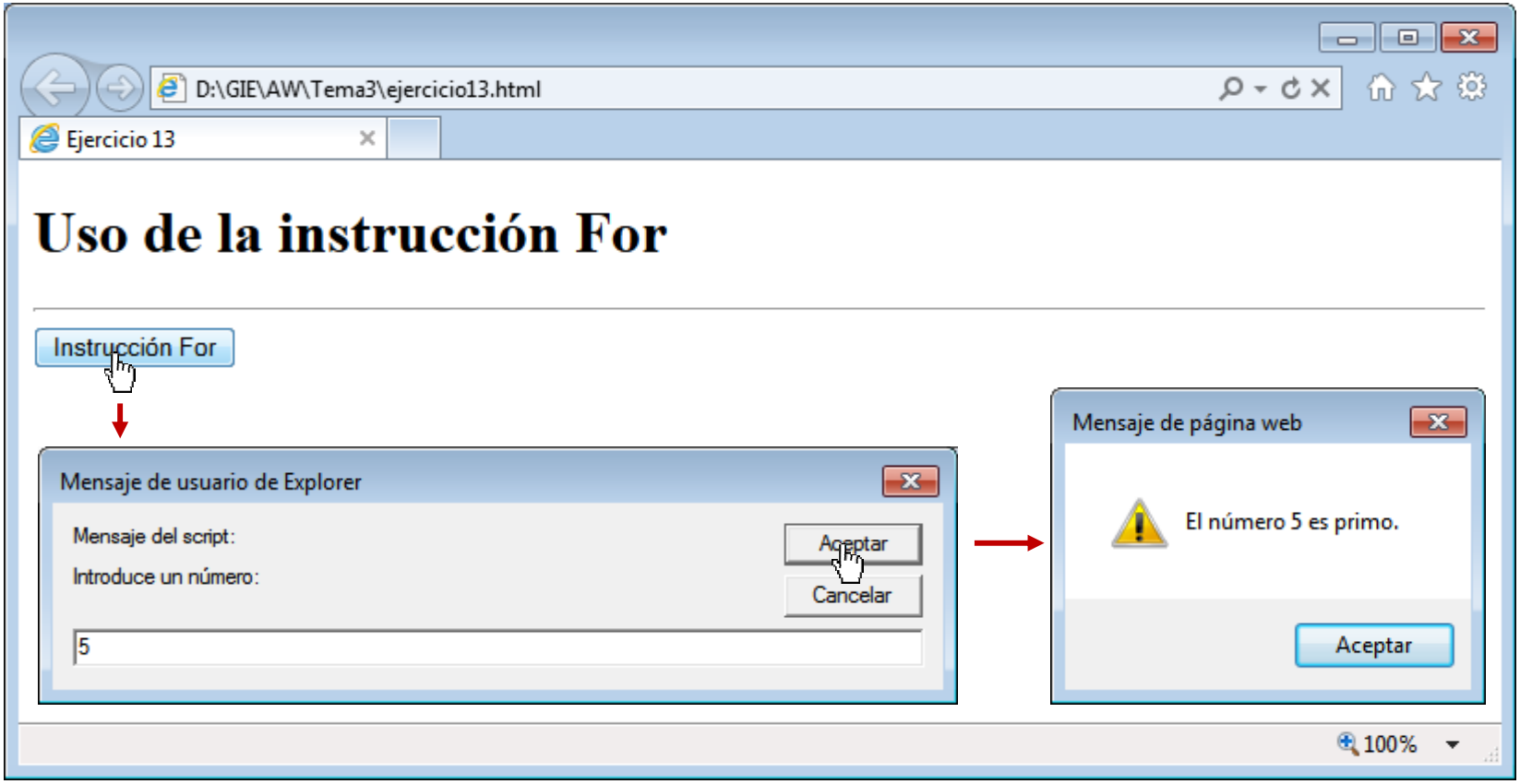
```
for(posicion in lista) lista[posicion];
```

  - Si se trata de un objeto, la variable de control va tomando los nombres de cada una de las propiedades de dicho objeto:

```
for(propiedad in objeto) objeto[propiedad];
```

➢ Instrucción FOR (3)

✓ Hacer un programa que compruebe si un número introducido por el usuario es primo o no:



Instrucción WHILE (1)

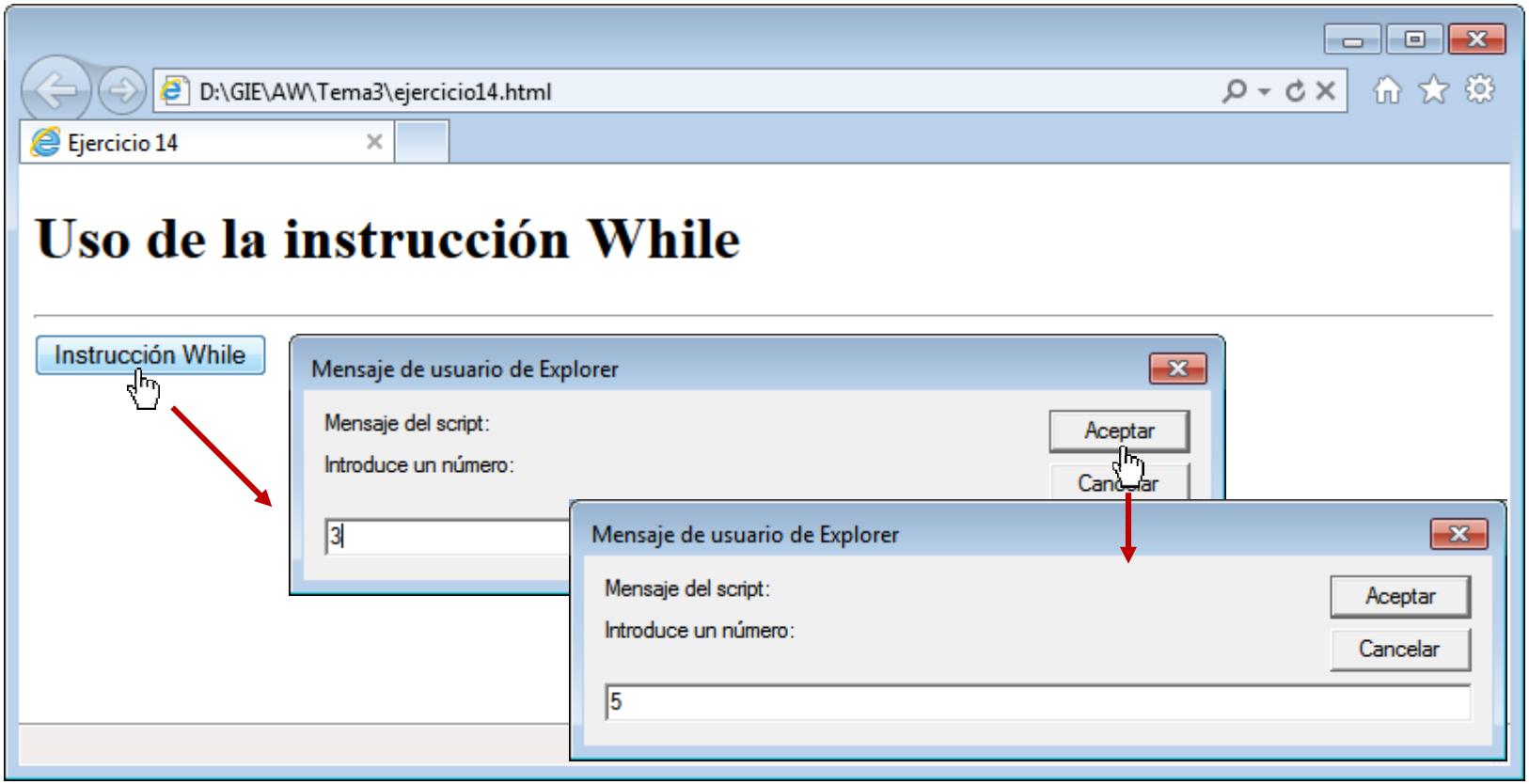
- La sintaxis de la sentencia repetición **while** tiene varias posibilidades:

```
while(condición) instrucción;

while(condición) {
  secuencia-de-instrucciones
}

do {
  secuencia-de-instrucciones
} while(condición)
```

- Mientras que la condición sea cierta, se sigue ejecutando la instrucción o bloque.
- A diferencia del bucle **while**, en el bucle **do..while** siempre se ejecuta al menos una vez el código.

Instrucción WHILE (2)

- ✓ Hacer un programa que continúe pidiendo números al usuario mientras no se introduzca el número 5:



Índice: Temas 3.1 y 3.2

Características generales

- Para definir una función se usa la siguiente sintaxis (aunque no haya parámetros, debe haber paréntesis):

```
function nombre(parámetros) {
  cuerpo de la función
}
```

- Para devolver un valor en una función se usa la instrucción **return**:

```
function miFuncion(par1, par2, par3) {
  var local1, local2;
  var resultado;
  //...
  return resultado;
}
```

- En las últimas versiones de JS las funciones se pueden anidar:

Ejemplo de función anidada

```
function hipotenusa(a, b) {
  function cuadrado(x) { return x*x; }
  return Math.sqrt(cuadrado(a) + cuadrado(b));
}
```

Paso de parámetros

- ✓ Por valor (o copia) si se trata de datos simples como números o cadenas de caracteres.
- ✓ Por referencia si se trata de objetos (realmente es un paso por valor del puntero).
- ✓ Normalmente se proporcionan tantos argumentos como parámetros definidos en la función y en el orden en el que están definidos.
- ✓ Sin embargo, también se puede llamar a una función con menos parámetros que los definidos en la declaración de la misma:

```
function hola(nombre) {
  if(typeof nombre == "undefined")   alert("Hola quienquiera que seas");
  else                               alert("Hola " + nombre);
}

hola();              // Muestra el mensaje "Hola quienquiera que seas"
```

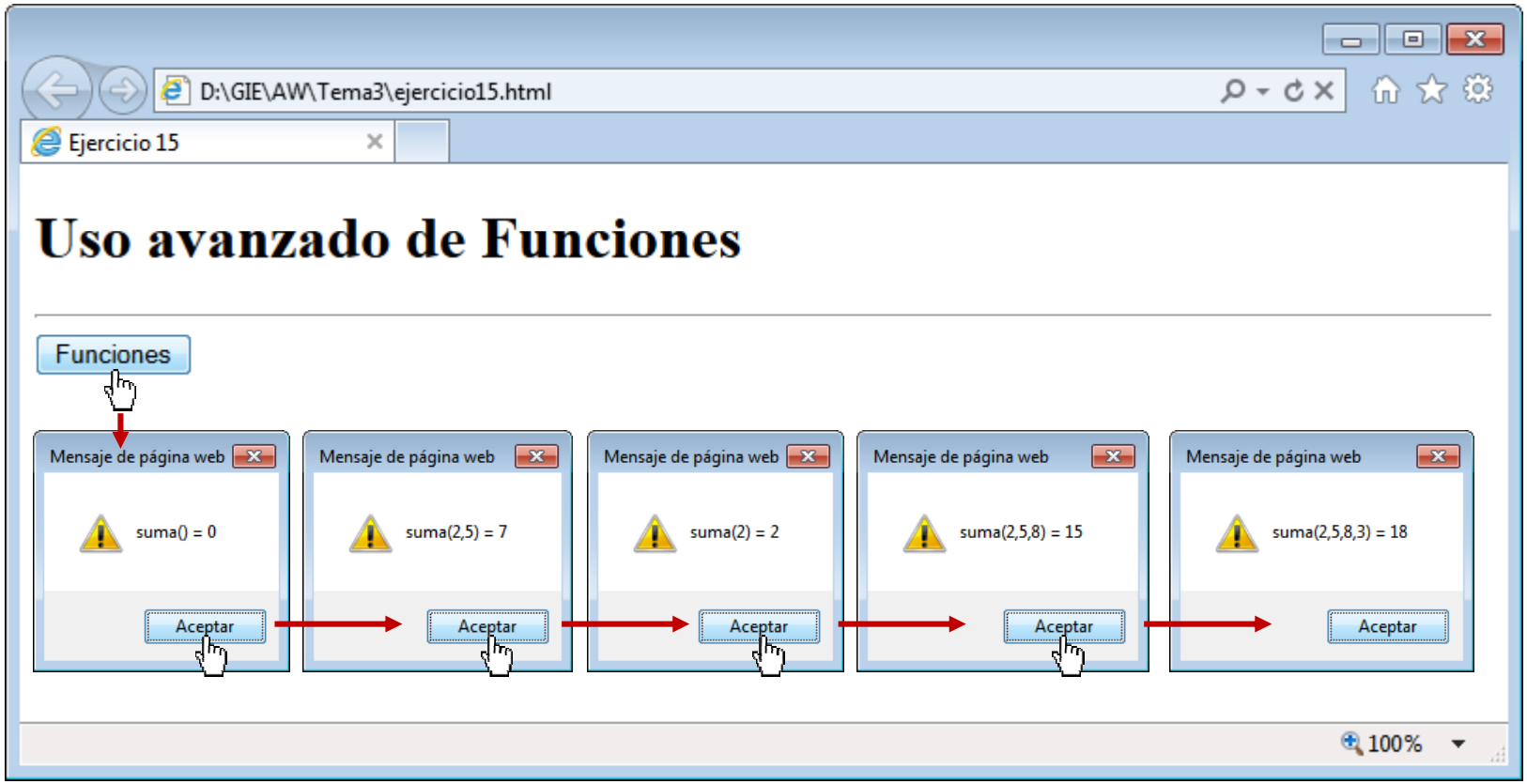
Número de argumentos variable (1)

- ✓ Aunque el número de argumentos de una función en JavaScript es fijo, podemos pasarle cualquier número de parámetros al invocarla.
- ✓ En estos casos, la función se declara sin argumentos, y estos son accesibles desde el objeto **`arguments`** en el interior de la misma:
  - El número de argumentos se accede mediante la propiedad **`length`**.
  - Los argumentos utilizando la sintaxis **`arguments[]`**.
- ✓ Ejemplo: Función máximo

```
function max(){
  var m = Number.NEGATIVE_INFINITY;
  for(var i = 0; i < arguments.length; i++)
     if (arguments[i] > m) m = arguments[i];
  return m;
}
```

➢ Número de argumentos variable (2)

✓ Diseñar una función capaz de sumar todos los números introducidos como argumentos de la misma:



El constructor Function()

- ✓ Utilizado junto al operador **`new()`**, sirve para crear funciones dinámicas.
- ✓ **`Function()`** recibe como argumentos un número de cadenas, donde la última de ellas es el cuerpo (instrucciones JavaScript) de la función:

Declaración dinámica de función y uso

```
var f = new Function("x", "y", "return x*y;");
alert(f(10,2));
```

Declaración estática equivalente

```
function f(x, y) { return x*y; }
```

Funciones como datos

- Las funciones pueden utilizarse como si fueran valores de variables:

Uso de función como dato

```
function square(x) { return x*x; }
var a = square(4);  // a contiene el número 16
var b = square;     // b referencia a la función square
var c = b(5);       // c contiene el número 25
```

- Las funciones también pueden asignarse a propiedades de objetos en vez de a variables globales:

Asignación a propiedades de objetos

```
var o = new Object;
o.square = new Function("x", "return x*x");
y = o.square(16);
```

- Las funciones no necesitan necesariamente nombres:

```
var a = new Array(3);
a[0] = function(x) { return x*x; }  // Función literal
a[1] = 20;
a[2] = a[0](a[1]);
```