

JavaScript. Nivel I

enero, 2020



Objetivos del nivel

- Conocer las características fundamentales de JavaScript.
- Crear scripts que interactúen con el usuario.
- Aprender a usar funciones y eventos.
- Aprender a trabajar con arreglos

Prerrequisitos del nivel

- HTML 5 y CSS 3 Nivel I
- Lógica de Programación Nivel II

Acerca de este manual

Este manual pertenece al Centro de Asesoramiento y Desarrollo Informático C.A. (CADI F1). Para obtener más información sobre este u otros cursos visite nuestro sitio Web www.cadif1.com, escribanos a la dirección de correo cadi@cadif1.com o visítenos en nuestra sede ubicada en la Av. Pedro León Torres con calle 59, Centro Comercial Sotavento, piso 2 oficina 27, Barquisimeto estado Lara, Venezuela. Tlf. 0251-7179247, 0251-4410268.

Las marcas mencionadas en este manual son propiedad de sus respectivos dueños.
Copyright 2020. Todos los derechos reservados.



Contenido del nivel

Capítulo 1. Introducción

- 1.1.- Qué es JavaScript.
- 1.2.- Ubicación Del Código JavaScript.
- 1.3.- Ejecución de Los Scripts.

Capítulo 2. Herramientas de Desarrollo

- 2.1.- La Consola.
- 2.2.- Salidas de Consola.
- 2.3.- El Depurador.

Capítulo 3. Variables y Tipos de Datos

- 3.1.- Declaración de Variables.
- 3.2.- Constantes.
- 3.3.- Operadores Aritméticos.

Capítulo 4. Funciones Matemáticas

- 4.1.- La Clase Number.
- 4.2.- La Clase Math.
- 4.3.- Generación de Números Aleatorios.

Capítulo 5. Instrucciones Condicionales. Parte 1

- 5.1.- Operadores Relacionales.
- 5.2.- Condiciones.

Capítulo 6. Instrucciones Condicionales. Parte 2

- 6.1.- Operadores Lógicos.
- 6.2.- El Operador ?.
- 6.3.- Switch.

Capítulo 7. Interacción Con el Usuario

- 7.1.- Mensajes Emergentes.
- 7.2.- Confirm.
- 7.3.- La Ventana Prompt.

Capítulo 8. Manipulación de Strings

- 8.1.- La Clase String.
- 8.2.- Métodos de la Clase String.
- 8.3.- Conversión de String.

Capítulo 9. Ciclos

- 9.1.- For.
- 9.2.- While.
- 9.3.- Do While.

Capítulo 10. Funciones. Parte 1

- 10.1.- Definición de Funciones.
- 10.2.- Llamado de Funciones.
- 10.3.- Ámbito Local vs Global.

Capítulo 11. Eventos. Parte 1

- 11.1.- Eventos.
- 11.2.- Manejadores de Eventos.
- 11.3.- Eventos y Funciones.

Capítulo 12. Funciones. Parte 2

- 12.1.- Parámetros.
- 12.2.- Valor de Retorno.

Capítulo 13. Arreglos. Parte 1

- 13.1.- Definición de Arreglos.
- 13.2.- Acceso a Los Elementos.
- 13.3.- Recorrido Del Arreglo.

Capítulo 14. Arreglos. Parte 2

- 14.1.- Funciones de Arreglos.
- 14.2.- Funciones Para Añadir Elementos o Concatenar Array.
- 14.3.- Funciones Para Extraer Elementos.



Capítulo 1. INTRODUCCIÓN

1.1.- Qué es JavaScript

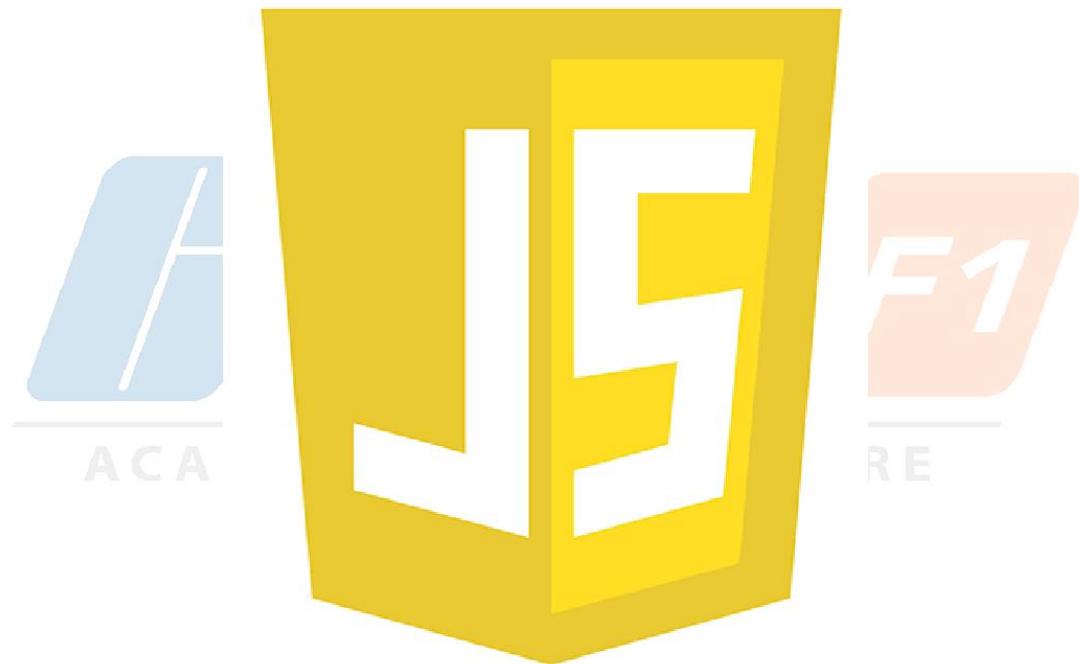
JavaScript (abreviado comúnmente "JS") es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. Su desarrollo comenzó en 1996, propuesto por Netscape Corporation, la empresa propietaria del navegador web Netscape.

JavaScript es un lenguaje de programación utilizado para crear programas encargados de realizar acciones dentro del ámbito de una página web, por lo tanto, es el navegador el encargado de interpretar y ejecutar las instrucciones JavaScript. Lo único que se necesita para programar con JavaScript es un navegador.



JavaScript es un lenguaje de programación muy parecido a Lenguaje C, es por esto que tiene casi las mismas características y reglas de éste, entre las cuales destacan:

- Es sensitivo a mayúsculas y minúsculas, es decir, si se define un identificador con el nombre “Edad” es diferente a otro con el nombre “edad”.
- Las instrucciones terminan con un “;” (es opcional, aunque es buena práctica).
- Los bloques de código se encierran entre las llaves “{” y “}”, que indican el inicio y fin de un bloque.
- Existen dos tipos de comentarios: comentarios de línea (//) y comentarios de bloque (/* */).



El estándar ECMAScript ha evolucionado en el tiempo. En la siguiente tabla se muestra como ha progresado desde su creación:

Edición	Fecha de publicación	Cambios desde la edición anterior
1	Junio de 1997	Primera edición
2	Junio de 1999	Cambios editoriales para mantener la especificación completa alineada con el estándar internacional ISO/IEC 16262
3	Diciembre de 1999	Se agregaron expresiones regulares, mejor manejo de strings, nuevo control de declaraciones, manejo de excepciones con try/catch, definición más estricta de errores formato para la salida numérica y otras mejoras.
4	Abandonado	La cuarta edición fue abandonada debido a diferencias políticas respecto a la complejidad del lenguaje.
5	Diciembre de 2009	Agrega el modo estricto ("strict mode"), un subconjunto destinado a proporcionar una mejor comprobación de errores y evitar constructores propensos a errores. Aísla varias ambigüedades de la tercera edición, y define el comportamiento de las implementaciones del "mundo real". Esta edición 5.1 de ECMAScript Standard está completamente alineada con la tercera edición del estándar internacional ISO/IEC 16262:2011.
5.1	Junio de 2011	
6	Junio de 2015	La sexta edición agrega cambios significativos en la sintaxis para escribir aplicaciones complejas, incluyendo clases y módulos, definiéndolos sencillamente en los mismos términos del modo estricto de la edición ECMAScript 5. Otras nuevas características incluyen iteradores, generadores, control de efectos y librerías/herramientas habilitadas desde ES6. Nuevas características propuestas incluyen promesas/concurrentia.
7	Junio de 2016	La séptima edición está en una etapa muy temprana de desarrollo, pero está orientada a continuar con la reforma del lenguaje, aislamiento de código, control de efectos y librerías/herramientas habilitadas desde ES6. Nuevas características propuestas incluyen iteradores, generadores, control de efectos y librerías/herramientas habilitadas desde ES6. Nuevas características propuestas incluyen promesas/concurrentia.
8	Junio de 2017	La 8 ^a edición, oficialmente conocida como ECMAScript 2017, fue finalizada en Junio de 2017. Incluye constructores <code>async/await</code> , los cuales funcionan usando generadores y promesas.
9	Junio de 2018	La 9 ^a edición, oficialmente conocida como ECMAScript 2018, incluye operadores <code>rest/spread</code> para variables (tres puntos: <code>...identificador</code>), iteración asíncrona <code>Promises.prototype.finally()</code> .

Entre las acciones típicas que se pueden realizar en JavaScript están:

* Por un lado los efectos especiales sobre páginas web, para crear contenidos dinámicos y elementos de la página que tengan movimiento, cambien de color o cualquier otro dinamismo.

* Por el otro, permite ejecutar instrucciones como respuesta a las acciones del usuario, con lo que se pueden crear páginas interactivas con programas como calculadoras, agendas, o tablas de cálculo.

1.2.- Ubicación Del Código JavaScript

El código JavaScript se puede usar de 2 formas:

- empotrado o incrustado en el HTML.
- en un archivo externo.

La primera forma es una de las más comunes. Es muy sencillo, pero tiene una desventaja: se mezclan los dos lenguajes. Para que estos dos lenguajes se puedan mezclar sin problemas se han de incluir unos delimitadores que separan las etiquetas HTML de las instrucciones JavaScript.

La siguiente imagen muestra un ejemplo de la inserción del código JavaScript en un archivo HTML:

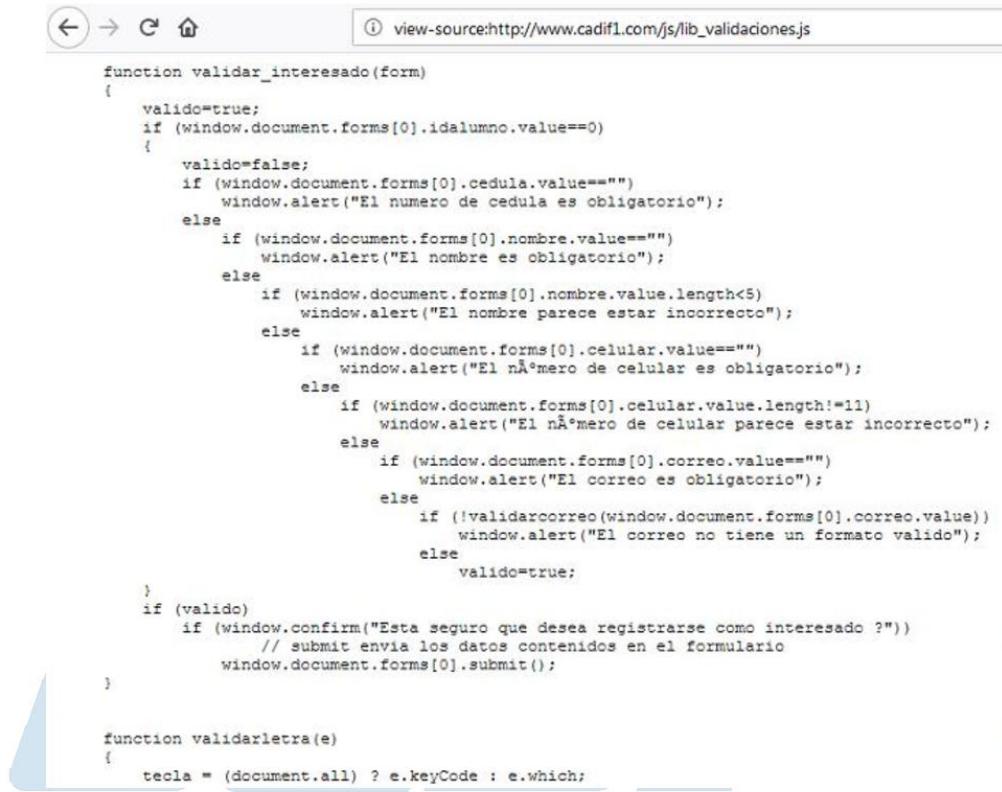
```

<link rel="stylesheet" type="text/css" href="/cPanel_magic_revision_1560220882/frontend/paper_lantern.css" />
<style type="text/css">
    /* Embedding: /usr/local/cpanel/base/frontend/paper_lantern/h
       .cpanel_body(max-width:none).panel-widget{border-radius:3px;box-shadow:0 0 3px rgba(0,0,0,0.1);
       .localytics-chosen.loading+.chosen-container-multi .chosen-choices{background-image:url('ima
    <!-- custom style css file (styles.css) should go here
    NOTE: Do not move below the header JS or you delay the -->
<link rel="stylesheet" type="text/css" href="/styled/current_style/sprites/icon_spriteemap.css?1560427650"/>
<link rel="stylesheet" type="text/css" href="/styled/current_style/styles.css?1560427650" />
<script>
    (function(){
        window.PAGE = {};
        window.MASTER = {};
        window.NVData = {};
        window.thisTheme = "paper_lantern";
        window.thisStyle = "basic";
    })();
</script>
<script type="text/javascript">
var interfaceConfigs=[];function register_interfacecfg_nvdata(nvname){interfaceConfigs.push(nvname)}
</script>
<script type="text/javascript">
    var DEFAULT_BOX_ORDER = ["files","jetbackup","databases","domains","email","metrics","security"];
    PAGE.isRTL = document.getElementsByTagName("HTML")[0].getAttribute("dir") === "rtl";
    PAGE.currentTheme = "paper_lantern";
    PAGE.securityToken = "\/cpanels569633492";
    PAGE.userName = "kpiaadmin";
    PAGE.domain = "kpimetro.com";
</script>

```

Estos delimitadores son las etiquetas `<SCRIPT>` y `</SCRIPT>`. Todo el código JavaScript que se coloque en la página ha de ser introducido entre estas dos etiquetas. Se pueden colocar la cantidad de bloques de script que hagan falta y donde sea necesario. No hay un bloque específico del HTML donde se deba colocar los bloques de script, aunque en algunas situaciones, la ubicación del script puede ser determinante para la correcta ejecución del mismo.

La otra manera de incluir scripts en un archivo HTML, implementada a partir de JavaScript 1.1, es crear archivos externos y colocar el código JavaScript ahí. La siguiente imagen muestra un ejemplo de un archivo externo de JavaScript:



The screenshot shows a browser window with the URL http://www.cadif1.com/js/lib_validaciones.js. The page content is the source code of a JavaScript file. The code defines two functions: `validar_interesado` and `validarletra`. The `validar_interesado` function performs various validations on form fields (idalumno, cedula, nombre, celular, correo) and submits the form if valid. The `validarletra` function handles key presses.

```

function validar_interesado(form)
{
    valido=true;
    if (window.document.forms[0].idalumno.value==0)
    {
        valido=false;
        if (window.document.forms[0].cedula.value=="")
            window.alert("El numero de cedula es obligatorio");
        else
            if (window.document.forms[0].nombre.value=="")
                window.alert("El nombre es obligatorio");
            else
                if (window.document.forms[0].nombre.value.length<5)
                    window.alert("El nombre parece estar incorrecto");
                else
                    if (window.document.forms[0].celular.value=="")
                        window.alert("El nÃºmero de celular es obligatorio");
                    else
                        if (window.document.forms[0].celular.value.length!=11)
                            window.alert("El nÃºmero de celular parece estar incorrecto");
                        else
                            if (window.document.forms[0].correo.value=="")
                                window.alert("El correo es obligatorio");
                            else
                                if (!validarcorreo(window.document.forms[0].correo.value))
                                    window.alert("El correo no tiene un formato valido");
                                else
                                    valido=true;
    }
    if (valido)
        if (window.confirm("Esta seguro que desea registrarse como interesado ?"))
            // submit envia los datos contenidos en el formulario
            window.document.forms[0].submit();
}

function validarletra(e)
{
    tecla = (document.all) ? e.keyCode : e.which;
}

```

Para hacer uso del código que se encuentra en el archivo externo, se debe incluir en el archivo HTML usando la misma etiqueta `<script>` usando la propiedad "src". En el siguiente ejemplo se muestra cómo incluir 2 archivos externos:

```

<!doctype html>
<html lang="es">
<head>
    <meta charset="utf-8">
    <title>Interesado en un curso</title>
    <!--[if lt IE 9]><script src="http://html5shiv.googlecode.com/svn/trunk/html5.js"></script>
<link rel="stylesheet" href="http://fonts.googleapis.com/css?family=Open+Sans:400,300,600,700,800" />
<link rel="shortcut icon" href="http://www.cadif1.com/img/iconos/logo.ico" />
<link rel="stylesheet" href="http://www.cadif1.com/css/jquery.ui.html4.css" />
<link rel="stylesheet" href="http://www.cadif1.com/css/main2.css" />
<script language="javascript" src="http://www.cadif1.com/js/jquery.min.1.7.js" /></script>
<script language="javascript" src="http://www.cadif1.com/js/bootstrap.js" /></script>
<script src="http://www.cadif1.com/js/accordion.js"></script>
<script src="http://www.cadif1.com/js/lib_validaciones.js"></script>
<script src="https://www.google.com/recaptcha/api.js"></script>
</head>
<body>
    <header>
        <div class="contenedor container-fluid">
            <div class="row">
                <figure>
                    <a href="http://www.cadif1.com/">
                        
                    </a>
                </figure>
            </div>
        </div>
    </header>

```

Esta práctica se realiza generalmente cuando hay mucho código JavaScript y no se quiere mezclar con el código HTML, o cuando hay código JavaScript que se desea reutilizar en varios archivos HTML. Algunas consideraciones al usar archivos externos son JavaScript son:

- Los archivos deben tener extensión .js
- No hace falta colocar las etiquetas <script>.
- Es una buena práctica que todos los archivos JS estén en una misma carpeta.
- Generalmente se incluyen en el head de la página, aunque se pueden incluir en cualquier lugar de la página.
- Una particularidad es que no se puede colocar código JavaScript dentro de un par de etiquetas que están incluyendo un archivo externo, debido a que ese código no se ejecutará.

Se deben colocar tantas etiquetas script como archivos se desean incluir (no hay un límite específico en cuanto a la cantidad de archivos que pueden incluirse).

El orden en que se incluyan los archivos externos de scripts puede ser irrelevante o no, dependerá de si un archivo utiliza lo que está declarado en el otro. Por ejemplo, existe una librería de Bootstrap que se basa en JQuery, en este caso, primero debe incluirse el archivo de jquery.js y luego el archivo bootstrap.js para que se pueda ejecutar correctamente la librería de Bootstrap.

Es una práctica recomendada incluir los archivos externos JavaScript al final del HTML para no retardar la descarga de la página, con la desventaja de que las funcionalidades que dependan del script no funcionaran hasta que se descargue completamente el archivo Js.

1.3.- Ejecución de Los Scripts

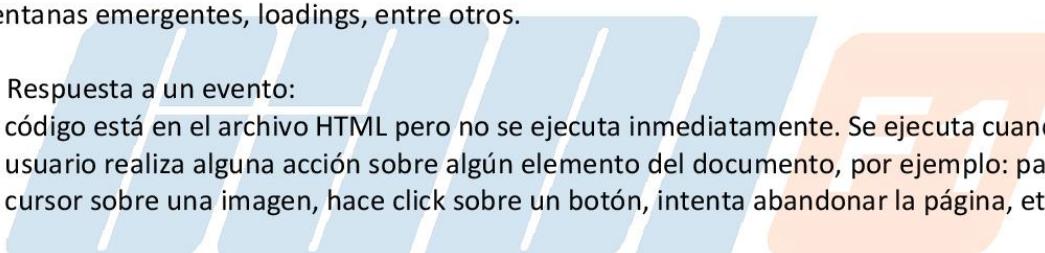
Existen dos formas de ejecutar código JavaScript en una página:

a. Ejecución directa:

Es el método más básico de ejecutar scripts. En este caso, cuando el navegador lee la página y encuentra un script, va interpretando las líneas de código y las va ejecutando inmediatamente una después de otra. Ejemplos de este tipo de ejecución son: sliders, ventanas emergentes, loadings, entre otros.

b. Respuesta a un evento:

El código está en el archivo HTML pero no se ejecuta inmediatamente. Se ejecuta cuando el usuario realiza alguna acción sobre algún elemento del documento, por ejemplo: pasa el cursor sobre una imagen, hace click sobre un botón, intenta abandonar la página, etc.

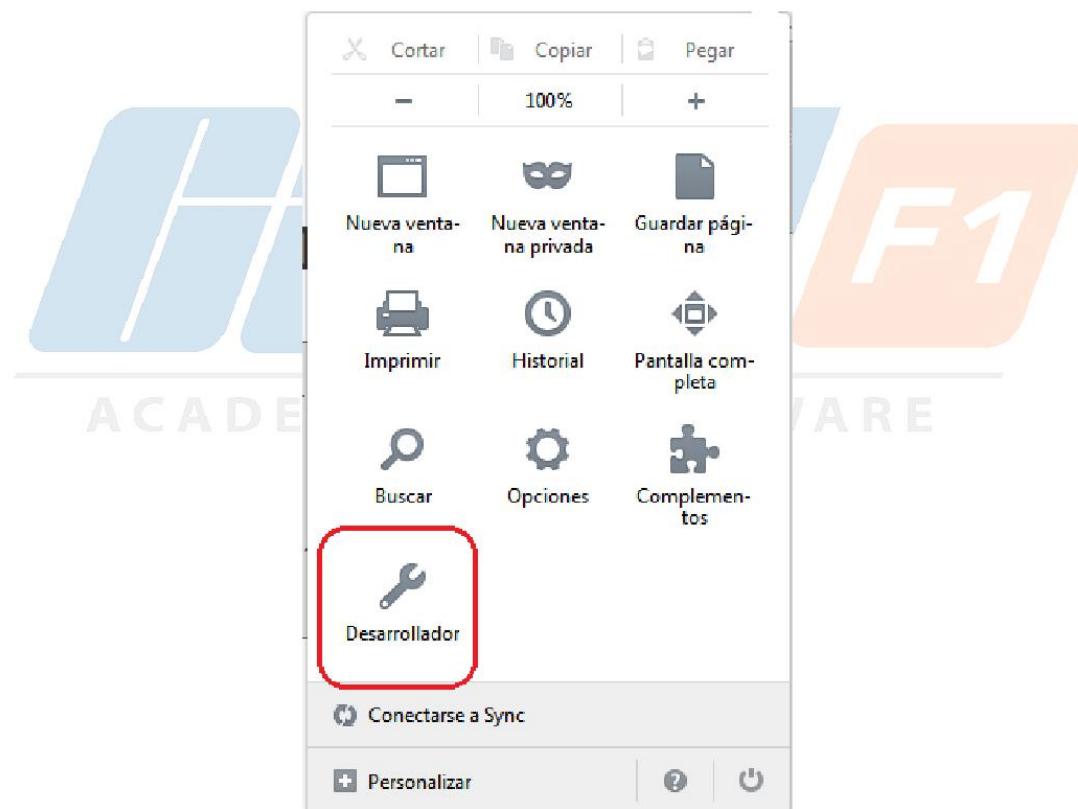


ACADEMIA DE SOFTWARE

Capítulo 2. HERRAMIENTAS DE DESARROLLO

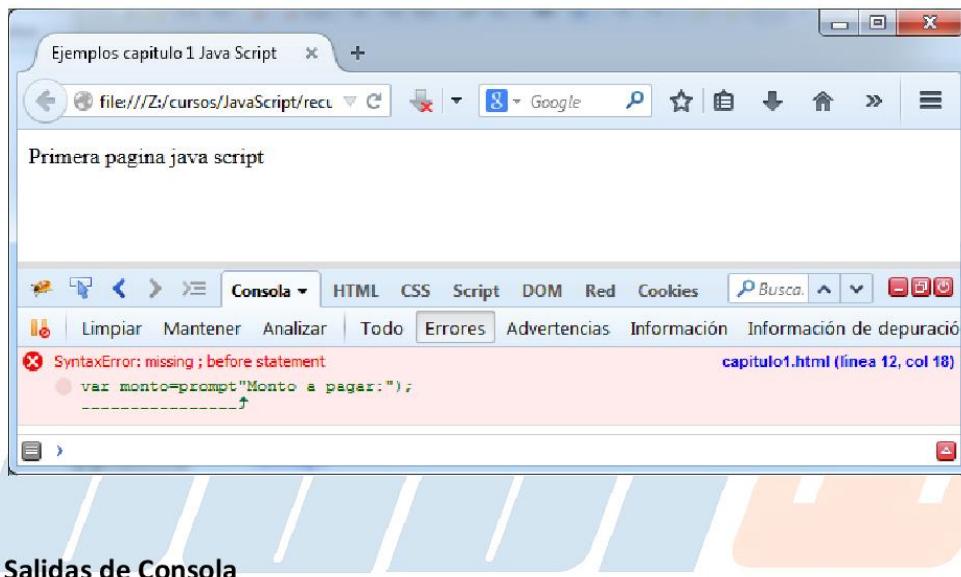
2.1.- La Consola

Como el encargado de ejecutar el código JavaScript es el navegador, es éste quien se encarga de ofrecer al programador herramientas de apoyo para facilitar su trabajo. A estas herramientas se les llama "Herramientas de Desarrollador" (aunque el nombre puede variar de un navegador a otro). Si se está usando Firefox, en el menú de herramientas aparece la opción "Desarrollador", como se muestra en la imagen:



También se puede iniciar presionando la tecla F12 en Firefox. Esta herramienta tiene muchos usos para el programador, entre ellas: el inspector, la consola, el depurador, el

monitor de red (network), visualizador de datos almacenados localmente. Entre las funciones de la consola están: visualizar errores de ejecución, evaluar el estado de una variable, probar instrucciones en el vuelo, visualizar salidas intencionales de las aplicaciones, entre otras.



2.2.- Salidas de Consola

Al ejecutar el código JavaScript se pueden generar salidas que no serán visibles para el usuario final, sino que serán de utilidad para el programador a través de la consola. Estas salidas se logran ejecutando la instrucción:

```
console.log("mensaje de salida...")
```

2.3.- El Depurador

Otras de las herramientas que posee el navegador que es de gran utilidad para los programadores es el depurador. Este permite visualizar el código JavaScript que contiene una página, colocar puntos de control en éste y ejecutar el código paso a paso. En la siguiente imagen se muestran todos los archivos JavaScript que contiene una página y el código de uno de ellos:

```
//let api_url="http://localhost/api_cadif1/";
let dominio="http://localhost/cadif1/";

let api_url="https://api.cadif1.com/";
//let dominio="http://web.cadif1.com/";
let filtrado=false;

$(document).ready(function(){
  $("#lista-cursos").hide();
  params=new URLSearchParams(window.location.search);
  idArea = params.get("id");
  nombreArea = "";

  $.get(api_url+"areadeestudio","",function(data, status){
    $("#loading-areas").hide();
    if (status=="success"){
      let areas=data.areas;
      
```

Para colocar un punto de control sólo debe ubicarse la línea de código en la cual se desea que la ejecución del script se detenga y hacer click con el botón izquierdo del ratón sobre el número que la identifica. Se mostrará un marcador en el número. Cuando el navegador ejecute esa línea se detendrá y esperará instrucciones del programador por medio de los botones que se encuentran resaltados en la parte inferior del depurador:

```
function filtrarCursos(idArea,nombre){
  // si se esta llenando la lista de cursos filtrada por un
  if (idArea!=0) {
    // si NO esta filtrado por un area
    if (!filtrado) {
      filtrado=true;
      // se elimina la etiqueta de Cursos Disponibles
      $("#bread-content > :last-child").remove();
      // se agrega el mismo texto pero con un hipervínculo ;
      $("#bread-content").append("<a href="#" onclick='desfiltra('" + nombre + "');''>" + nombre + "</a>");
    }else
  }
}
```

Las opciones son:

- Reanudar la ejecución (F8): continua con la ejecución sin detenerse a menos que encuentre otro punto de control.
- Pasar sobre la instrucción (F10): si la instrucción es el llamado a una función no entrará a ejecutar cada instrucción.
- Entrar en la instrucción (F11): si la instrucción es el llamado a una función entrará a ejecutar cada instrucción de la función.

Para quitar el punto de control solo debe hacer click nuevamente sobre el punto de control existente.



Capítulo 3. VARIABLES Y TIPOS DE DATOS

3.1.- Declaración de Variables

JavaScript es un lenguaje débilmente tipado, es decir, que no es estricto en cuanto a los tipos de datos que pueden manejar las variables. Aun así, las variables tendrán un tipo de dato de entre los siguientes:

- Numérico
- Booleano
- Cadenas
- Objetos
- Arreglos

Los nombres de las variables han de construirse con caracteres alfanuméricos y/o el carácter subrayado (_). Aparte de esta, hay una serie de reglas adicionales para construir nombres para variables. La más importante es que tienen que comenzar por una letra o el subrayado. No se puede utilizar caracteres especiales ni operadores aritméticos (+, ?, *, -). El símbolo \$ está permitido. Algunos ejemplos de nombres admitidos para las variables podrían ser:

Edad
paisDeNacimiento
_nombre

Tampoco se pueden utilizar palabras reservadas del lenguaje.

Es una costumbre habitual en los lenguajes de programación definir las variables que se van a usar en los programas. En JavaScript no es obligatorio declarar las variables, aunque es recomendable hacerlo. La forma general de declarar una variable es:

```
var operando1 ;
```

También se puede asignar un valor a la variable cuando se está declarando:

```
var operando1 = valor;
```

O simplemente se le puede asignar un valor a un identificador y si aún no ha sido declarada la variable, se declarará en ese momento:

```
operando1 = valor;
```

Cuando una variable se declara, pero no se le ha asignado valor, tiene por defecto el valor "undefined". A diferencia de otros lenguajes de programación, JavaScript no define diferentes tipos de datos para los valores numéricos, tales como: int, float, short, etc. (como Java, C, pascal, etc.). En JavaScript todos los números son siempre almacenados como números de coma flotante de doble precisión.

3.2.- Constantes

Una constante es básicamente una variable cuyo valor asignado no puede ser cambiado durante la ejecución del script. En JavaScript una constante se declara usando la palabra reservada "const", por ejemplo:

```
const MAX = 5000;
```

Las constantes pueden ser numéricas o alfanuméricas.

3.3.- Operadores Aritméticos

Los operadores de JavaScript son similares a los de lenguaje C. Como en todos los lenguajes de programación, JavaScript posee operadores de asignación, aritméticos, lógicos y relacionales. La siguiente tabla muestra un resumen de los operadores aritméticos:

Operador	Significado	Ejemplo
+	suma	números y cadenas
-	resta	
*	producto	
/	división	
%	módulo (resto)	20 % 10 (= 0)
++	suma tipográfica	variable++; ++variable; (variable = variable + 1)
-- (dos guiones)	resta tipográfica	variable--; --variable; (variable = variable - 1)

También existen operadores aritméticos resumidos, que asignan valor a una variable luego de una operación aritmética. Entre ellos están:

Operador	Ejemplo	Equivalente a
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

Capítulo 4. FUNCIONES MATEMÁTICAS

4.1.- La Clase Number

JavaScript posee una clase para manejar las operaciones relacionadas con los números. La clase se llama Number y se utiliza de forma implícita al declarar una variable de tipo numérico. Contiene varios métodos de gran utilidad, tales como:

- `toString`: lleva un número a string.
- `toFixed`: retorna un string con el número escrito con el número especificado de decimales.
- `toPrecision`: retorna un string con un número con una longitud específica.

Por ejemplo:

```
var x = 9.656;  
  
x.toFixed(0);      // returns 10  
x.toFixed(2);      // returns 9.66  
x.toFixed(4);      // returns 9.6560  
x.toFixed(6);      // returns 9.656000  
  
x.toPrecision();    // returns 9.656  
x.toPrecision(2);  // returns 9.7  
x.toPrecision(4);  // returns 9.656  
x.toPrecision(6);  // returns 9.65600
```

4.2.- La Clase Math

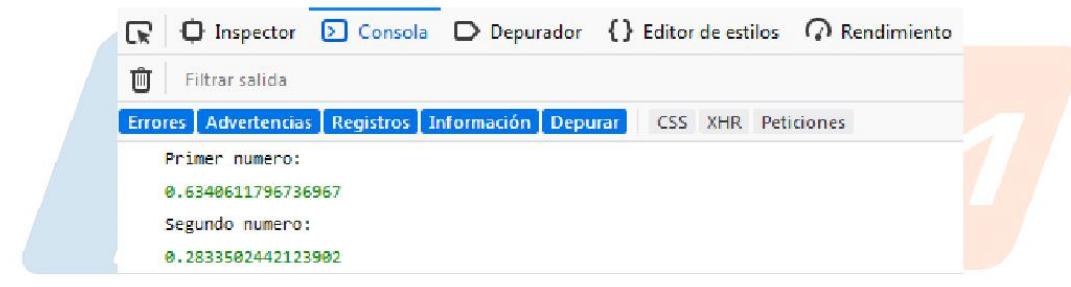
La clase Math contiene métodos de mucha utilidad para realizar operaciones aritméticas. Los métodos son estáticos, por lo tanto, se pueden ejecutar sin necesidad de instanciar la clase. Los métodos más usados son:

- `pow(x,y)`: retorna el valor "X" elevado a la potencia de "Y".
- `sqrt(x)`: retorna la raíz cuadrada del valor "X".

- `round(x)`: retorna el valor de "X" redondeado al entero más cercano utilizando las reglas de redondeo tradicionales.
- `ceil(x)`: retorna el valor "X" redondeado al entero más alto.
- `floor(x)`: retorna el valor "X" redondeado al entero más bajo.
- `random(x)`: retorna un número aleatorio entre 0 (inclusive) y 1 (excluyendo).

4.3.- Generación de Números Aleatorios

Para generar números aleatorios se utiliza la función `Math.random`. La función `Math.random()` retorna un número punto flotante pseudo-aleatorio dentro del rango [0, 1). Esto es, desde el 0 (incluido) hasta el 1 pero sin incluirlo (excluido), el cual se puede escalar hasta el rango deseado. Por ejemplo:



```

A
var n = Math.random();
console.log("Primer numero: ");
console.log(n);

n = Math.random();
console.log("Segundo numero: ");
console.log(n);

```

The screenshot shows the browser's developer tools with the 'Console' tab selected. It displays two lines of output from the console:

```

Primer numero:
0.6340611796736967
Segundo numero:
0.2833502442123902

```

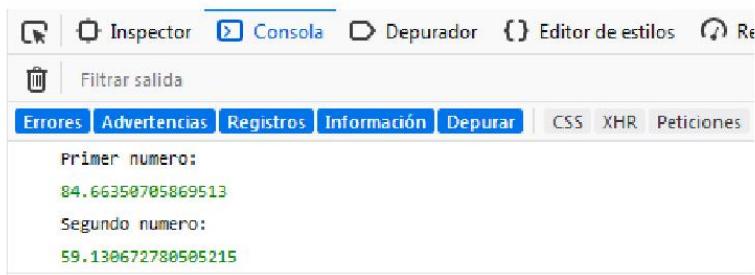
Se puede generar un número aleatorio en un rango de 2 valores enteros, un mínimo y un máximo, por ejemplo, entre 50 y 100. Esto se logra usando la siguiente expresión:

```
Math.random() * (100 - 50) + 50;
```

De forma general:

```
Math.random() * (max - min) + min;
```

Donde "min" es el valor mínimo del rango y "max" es el valor máximo del rango.



The screenshot shows a browser's developer tools with the 'Console' tab selected. It displays the following output:

```
Primer numero:  
84.66350785869513  
Segundo numero:  
59.130672780505215
```

Below the console, the corresponding JavaScript code is shown:

```
var min = 50;  
var max = 100;  
  
var n = Math.random()*(max -min)+ min;  
console.log("Primer numero: ");  
console.log(n);  
  
var n = Math.random()*(max -min)+ min;  
console.log("Segundo numero: ");  
console.log(n);
```

Los valores generados contienen decimales. Se pueden truncar o redondear utilizando las funciones toFixed o toPrecision de la clase Number o las funciones round, ceil o floor de la clase Math. Por ejemplo:

```
var min = 50;
var max = 100;

var n = Math.random()*(max -min)+ min;
console.log("Número original: ");
console.log(n);

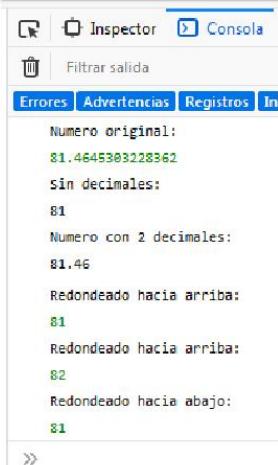
console.log("Sin decimales: ");
console.log(n.toFixed(0));

console.log("Número con 2 decimales: ");
console.log(n.toFixed(2));

console.log("Redondeado : ");
console.log(Math.round(n));

console.log("Redondeado hacia arriba: ");
console.log(Math.ceil(n));

console.log("Redondeado hacia abajo: ");
console.log(Math.floor(n));
```



Filtrar salida

Errores | Advertencias | Registros | In

Número original:
81.4645383228362

Sin decimales:
81

Número con 2 decimales:
81.46

Redondeado hacia arriba:
81

Redondeado hacia abajo:
82

Redondeado hacia abajo:
81

»



Capítulo 5. INSTRUCCIONES CONDICIONALES. PARTE 1

5.1.- Operadores Relacionales

Antes de hacer una condición deben conocerse los operadores relacionales que permiten comparar una variable con otra o con un valor literal. Los operadores relacionales se muestran en la siguiente tabla:

Operador	Significado	Ejemplo
<code>==</code>	es igual a	<code>5 == 8</code> es falso
<code>!=</code>	no es igual a	<code>5 != 1</code> es verdad
<code>></code>	es mayor que	<code>5 > 1</code> es verdad
<code><</code>	es menor que	<code>5 < 8</code> es verdad
<code>>=</code>	es mayor o igual que	<code>5 >= 8</code> es falso
<code><=</code>	es menor o igual que	<code>5 <= 1</code> es falso

JavaScript permite comparar valores de diferentes tipos de datos y aun así concluir que son 2 valores iguales, por ejemplo, al comparar:

```
console.log(100 == "100")
```

Se obtendrá true. Si es necesario determinar si 2 valores son iguales y también son del mismo tipo, se debe utilizar el operador "`==`". La evaluación

```
console.log(100 === "100")
```

retornara false.

5.2.- Condiciones

IF es una estructura de control utilizada para tomar decisiones. Es un condicional que realiza unas u otras operaciones en función de una expresión. Funciona de la siguiente

manera, primero se evalúa una expresión, si el resultado es verdadero, se realizan algunas instrucciones. La sintaxis de la estructura IF es la siguiente:

```
if (expresión) {
    acciones a realizar en caso positivo
}
```

La siguiente imagen muestra un ejemplo de cómo usar el if:

The screenshot shows a browser's developer tools console tab. The code is as follows:

```
var min = -50;
var max = 50;

var n = Math.random()*(max-min)+ min;
console.log("Número original: ");
console.log(n);

if (n > 0) console.log("Es positivo");
```

The output in the console is:

Filtrar salida

Erros Advertencias Registros Información

Número original:
37.32941155947452
Es positivo

Las llaves son opcionales. Se utilizan cuando más de una instrucción estará condicionada por la expresión lógica, por ejemplo:

The screenshot shows a browser's developer tools console tab. The code is as follows:

```
var min = -50;
var max = 50;

var n = Math.random()*(max-min)+ min;
console.log("Número original: ");
console.log(n);

if (n > 0) {
    console.log("Es positivo");
    // se quitan los decimales
    n = n.toFixed(0);
    console.log("Número sin decimales: ");
    console.log(n);
}
```

The output in the console is:

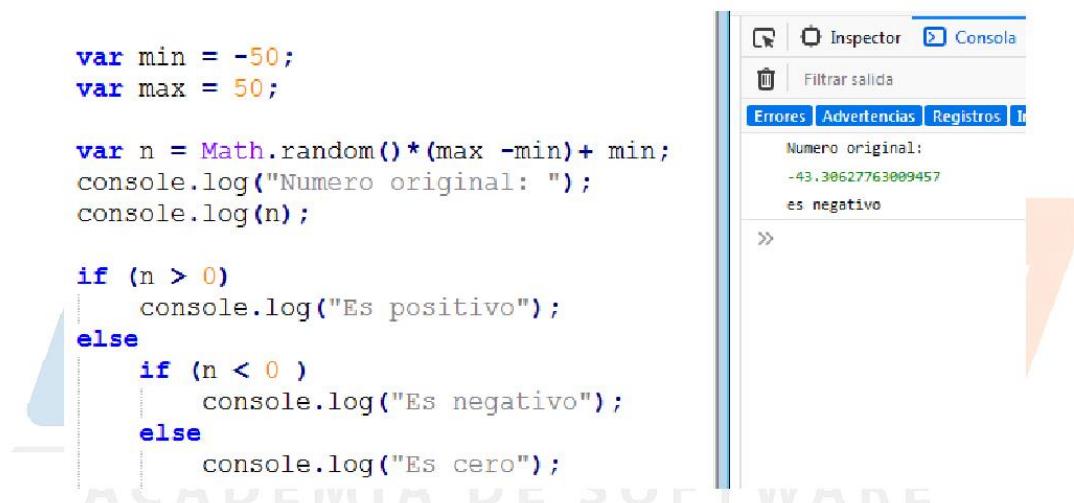
Filtrar salida

Erros Advertencias Registros

Número original:
21.723458379161244
Es positivo
Número sin decimales:
22

Opcionalmente se pueden indicar acciones a realizar en caso de que la evaluación de la sentencia sea falsa, incluso haciendo una secuencia de condiciones anidadas:

```
if (expresión) {
    acciones a realizar en caso positivo
}else {
    acciones a realizar en caso negativo
}
```



The screenshot shows a browser's developer tools console tab. The code executed is:

```
var min = -50;
var max = 50;

var n = Math.random()*(max -min)+ min;
console.log("Número original: ");
console.log(n);

if (n > 0)
    console.log("Es positivo");
else
    if (n < 0 )
        console.log("Es negativo");
    else
        console.log("Es cero");
```

The output in the console is:

Número original:
-43.38627763009457
es negativo

Capítulo 6. INSTRUCCIONES CONDICIONALES. PARTE 2

6.1.- Operadores Lógicos

Los operadores lógicos ("and" y "or") permiten crear expresiones lógicas más complejas. En la siguiente tabla se listan los operadores lógicos

Operador	Significado	Ejemplo
&&	Y	<code>1 == 1 && 2 < 1</code> es falso
	O	<code>1 == 2 15 > 2</code> es verdad
!	NO	<code>!(1 > 2)</code> es verdad

6.2.- El Operador ?

Para hacer condiciones en circunstancias simples, se puede utilizar un operador condicional muy singular: el "?". Se le denomina operador ternario y su uso general es el siguiente:

`expresionLogica ? operacionA : operacionB`

Se evalúa una expresión lógica, si la expresión lógica es true, se ejecuta "operacion A", en caso contrario, se ejecuta "operacion B". Por ejemplo:

`a = n>0 ? n*2 : n/2;`

En el ejemplo anterior, se evalúa si n es mayor a 0, en caso de que sea true esa premisa, la variable "a" tomará el valor de la expresión "n*2", en caso contrario, tomara el valor de la expresión "n/2". Esta instrucción es equivalente a hacer el siguiente "if":

```
if (n>0 )
    a = n*2;
else
    a = n /2;
```

6.3.- Switch

Cuando es necesario evaluar si una variable puede tomar ciertos valores puntuales, en vez de utilizar un anidamiento de "if", se utiliza la instrucción condicional "switch". La estructura general del switch es la siguiente:

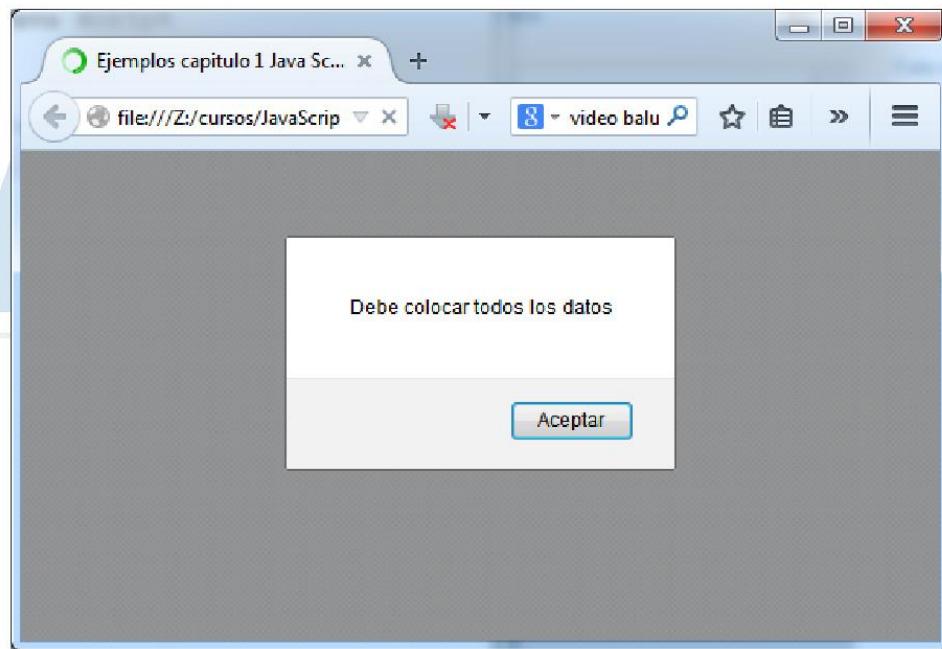
```
switch( variable )
{
    case valor X:
        expresionX1;
        expresionX2;
        break;
    case valor Y:
        expresionY1;
        expresionY2;
        break;
    default:
        expresion Default1;
}
```

ACADEMIA DE SOFTWARE

Capítulo 7. INTERACCIÓN CON EL USUARIO

7.1.- Mensajes Emergentes

Una de las tareas que más comúnmente se realizan con JavaScript es notificarle al usuario final la ocurrencia de algún evento. Console.log es utilizado para el desarrollador, pero no para el usuario final (aunque en teoría lo puede visualizar si sabe dónde encontrarlo). Lo más usual, es mostrarle la información con una ventana emergente. En la siguiente imagen se muestra un ejemplo de como aparecen las ventanas emergentes (la forma de la ventana puede variar de un navegador a otro):



Estas ventanas emergentes deben usarse con cuidado, debido a que en algún momento se utilizaron de forma negativa, por tal razón, algunas nuevas versiones de los navegadores bloquearon por defecto estas ventanas. La instrucción para mostrar mensajes emergentes se llama "alert". La siguiente imagen muestra un ejemplo de como se usa la función alert:

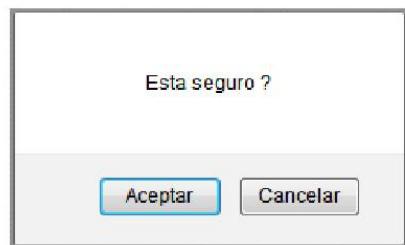
```
<html>
  <head>
    <title>
      Ejemplos capitulo 1 Java Script
    </title>
  </head>
  <script language="javascript">
    // codigo de ejecucion directa
    alert("Debe colocar todos los datos");
  </script>
  <body>
    Primera pagina java script
  </body>
</html>
```

7.2.- Confirm

JavaScript posee una función que permite mostrar al usuario una confirmación, de modo que pueda determinarse si desea realizar una acción o no. Muestra una ventana como "alert" pero con los botones "Aceptar" y "Cancelar". Si el usuario hace click en el botón "Aceptar" retorna true, en caso contrario, retorna false. Cómo implica una decisión, generalmente va acompañado de un condicional. En la siguiente imagen se muestra un ejemplo de cómo usar el confirm:

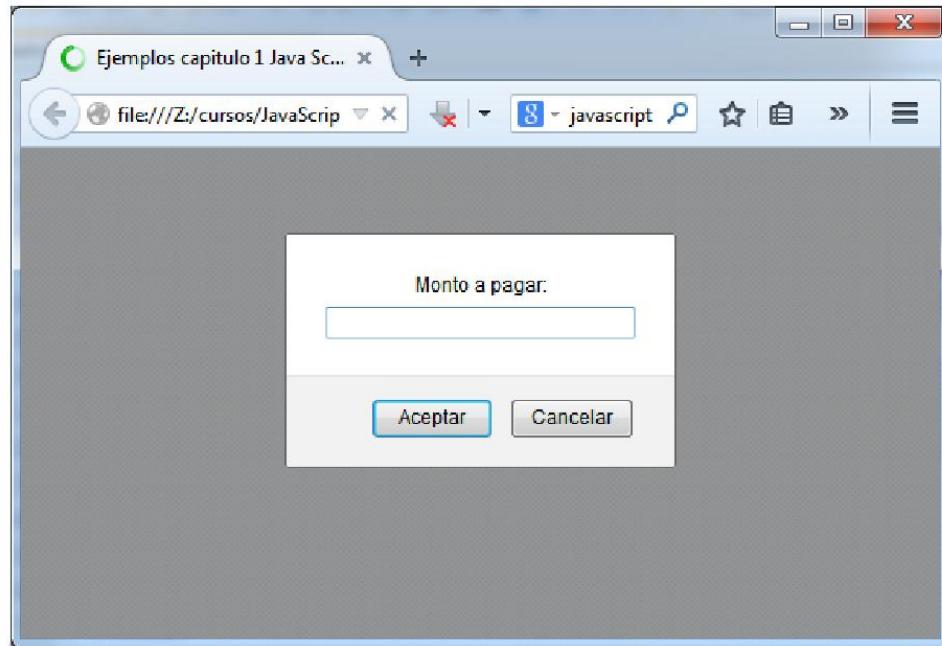
```
<script language="javascript">
    // código de ejecución directa

    if (confirm("Esta seguro ?"))
        alert("El usuario hizo click en aceptar");
    else
        alert("El usuario hizo click en cancelar");
</script>
```



7.3.- La Ventana Prompt

Adicionalmente, en ocasiones se necesita solicitar algún dato al usuario. Para esto JavaScript posee una función llamada "prompt", que muestra una ventana con un mensaje y un cuadro de texto para que el usuario introduzca información. La siguiente imagen muestra como aparece la ventana solicitando información al usuario:



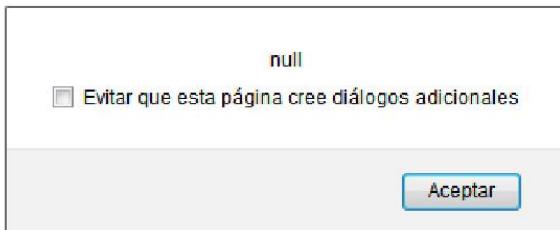
El dato que el usuario escribe puede almacenarse en una variable para luego ser utilizada. Es importante resaltar que el dato que retorna "prompt" es de tipo String, por lo tanto, debe tener en cuenta que si el dato solicitado se quiere manipular como número, debe convertirse. En la imagen se muestra un ejemplo de cómo usar la función "prompt":

```
<script language="javascript">
    // codigo de ejecucion directa
    //alert("hola mundo");
    var monto=prompt("Monto a pagar:");
    var nro_horas=prompt("Numero de horas:");
</script>
<body>
    Primera pagina java script
</body>
```

Si el usuario hace click en el botón "Aceptar" sin escribir nada, "prompt" retornará una cadena vacía. Si el usuario hace click en el botón "Cancelar", la función prompt retornará el valor "null":

```
<script language="javascript">
    // código de ejecución directa
    var monto=prompt("Monto a pagar:");

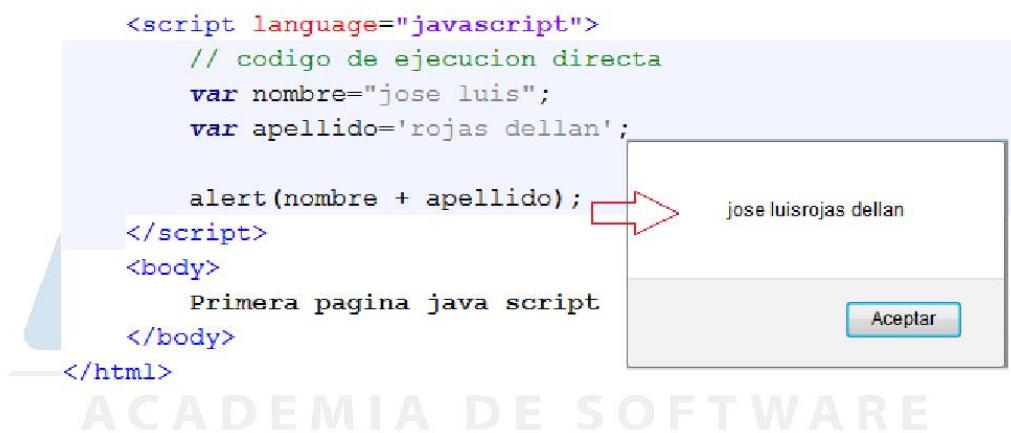
    alert(monto);
</script>
```



Capítulo 8. MANIPULACIÓN DE STRINGS

8.1.- La Clase String

JavaScript posee una clase implícita llamada String que es utilizada para manipular las cadenas de caracteres. Una cadena de caracteres va encerrada entre comillas dobles o simples. En la siguiente imagen se muestra un ejemplo de cómo declarar variables que contienen valores String. Uno de las operaciones más comunes con los strings es concatenarlos (unir 2 cadenas o más). Esto se logra con el operador "+":



```

<script language="javascript">
    // codigo de ejecucion directa
    var nombre="jose luis";
    var apellido='rojas dellan';

    alert(nombre + apellido);
</script>
<body>
    Primera pagina java script
</body>
</html>

```

ACADEMIA DE SOFTWARE

Se pueden concatenar varias cadenas a la vez:



```

<script language="javascript">
    // codigo de ejecucion directa
    //alert("hola mundo");
    var monto=prompt("Monto a pagar:");
    var nro_horas=prompt("Numero de horas:");

    alert("El monto es "+monto+" y el numero de horas es "+nro_horas);
</script>

```

Cuando se hacen operaciones entre números y strings JavaScript siempre tratará de convertir los strings en números, por ejemplo:

```
var x = 5 + 5;  
var y = "5" + 5;  
var w= "5" * 5;  
var z = "Hello" + 5;
```

Resultará:

10
55
25
Hello5

8.2.- Métodos de la Clase String

Otra de las operaciones muy comunes con los string es convertirlos a mayúsculas. Para esto se utiliza el método `toUpperCase()` de la clase `String`. El método `toLowerCase()` convierte a minúsculas. También es muy común necesitar conocer la cantidad de caracteres que tiene una cadena. Esto se logra con la propiedad `"length"`. En la siguiente imagen se muestra un ejemplo de cómo usar estas funciones:

```
<script language="javascript">
    // código de ejecución directa
    var nombre="jose luis";
    var apellido='rojas dellan';
    var nombrecompleto=nombre + " " + apellido;
    alert("mi nombre es "+nombrecompleto.toUpperCase()+
        " y tiene "+nombrecompleto.length+" caracteres");
</script>
```



mi nombre es JOSE LUIS ROJAS DELLAN y tiene 22 caracteres

Aceptar

8.3.- Conversión de String

También es muy común necesitar convertir datos string a números, ya sean enteros o reales. Para esto JavaScript tiene 2 funciones:

- parseInt: convierte de string a un número entero.
- parseFloat: convierte de string a un número real.

El siguiente ejemplo muestra el uso de ambas funciones:

```

<script language="javascript">
    // código de ejecución directa
    var dato1=prompt("Monto a pagar:");
    var dato2=prompt("Número de horas:");

    monto=parseFloat(dato1);
    nro_horas=parseInt(dato2);
    bono=monto*0.1;
    horas_extras=nro_horas-10;

    alert("El bono es "+bono+
        " y el número de horas extras "+nro_horas);
</script>

```

Si la cadena que se está intentando convertir a número no es un número, las funciones de conversión retornan el valor NaN (Not a Number). Esto indica que la conversión no pudo realizarse exitosamente. Al realizar cualquier operación aritmética con un valor NaN el resultado será un NaN. Entonces se necesita verificar si la variable contiene un valor numérico válido (NaN) usando una función llamada "isNaN". Esta función devuelve "true" si la variable evaluada es NaN (no es un número válido) y devuelve "false" si la variable contiene un número válido. Un ejemplo de como usar la función "isNaN" es el siguiente:

ACADEMIA DE SOFTWARE

```

<script language="javascript">
    // código de ejecución directa
    var edad= 10;

    if (isNaN(edad))
        alert("No es un número");
    else
        alert("Es un número válido ");
</script>

```

Esta función es usada especialmente cuando los valores numéricos son entradas introducidas por el usuario, debido a que el usuario puede cometer un error y colocar un valor inválido. Si el valor de la variable es leído con la función "prompt", se puede evaluar

si al convertir a número el string leído se convierte exitosamente, en cuyo caso, se obtiene el valor numérico, en caso contrario, se obtiene un NaN. El siguiente es un ejemplo de como combinar "isNaN" con prompt:

```
<script language="javascript">
    // código de ejecución directa
    var monto=prompt("Monto a pagar:");

    if (isNaN(monto))
        alert("El usuario introdujo un valor inválido");
    else
        alert("El monto es "+monto);
</script>
```



Capítulo 9. CICLOS

9.1.- For

Java Script como cualquier otro lenguaje de programación posee instrucciones para realizar ciclos repetitivos. Uno de ellos es el ciclo for. El ciclo for se utiliza para repetir más instrucciones un determinado número de veces. De entre todos los bucles, el FOR se suele utilizar cuando sabemos con seguridad el número de veces que queremos que se ejecute la sentencia. La sintaxis del bucle se muestra a continuación:

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

En la imagen se muestra un ejemplo de cómo usar el ciclo for:

```
<script language="javascript">  
// código de ejecución directa  
var nro_alumnos=prompt("Introduzca el numero de alumnos:");  
var total_nota=0;  
  
for (i=1;i<=nro_alumnos;i++)  
{  
    var nota=parseFloat(prompt("Introduzca la nota del alumno "+i));  
    total_nota+=nota;  
}  
promedio=total_nota/nro_alumnos;  
alert("El promedio de la sección:"+promedio);  
  
</script>
```

9.2.- While

Otro de los tipos de ciclos es el WHILE. Este tipo de ciclo se utilizan cuando se quiere repetir la ejecución de unas sentencias un número indefinido de veces, siempre que se cumpla una condición. Es más sencillo de comprender que el bucle FOR, pues no incorpora en la misma línea la inicialización de las variables su condición para seguir

ejecutándose y su actualización. Sólo se indica la condición que se tiene que cumplir para que se realice una iteración. La sintaxis general del while es la siguiente:

```
while ( expresión lógica )
{
    sentencias a ejecutar
}
```

En la siguiente imagen se muestra un ejemplo del uso del while:

```
<script language="javascript">
    var total_nota=0;
    var nro=0;
    while (confirm("Desea registrar una nota?"))
    {
        var nota=parseInt(prompt("Introduzca la nota del alumno "+(nro+1)));
        if (isNaN(nota))
            alert("No introdujo una nota válida");
        else
        {
            total_nota+=parseFloat(nota);
            nro_alumnos++;
        }
    }
    if (nro_alumnos>0)
    {
        promedio=total_nota/nro_alumnos;
        alert("Se registraron "+nro_alumnos+". El promedio es:"+promedio);
    }else
        alert("No se registraron notas");
</script>
```

9.3.- Do While

Otro de los ciclos que posee JavaScript es el do..while, que se utiliza cuando se necesita que el ciclo se ejecute al menos una vez. La diferencia con el ciclo "while" es que la condición se evalúa al final. Si la expresión evaluada es verdadera, el ciclo se repite una vez más, en caso contrario, se rompe el ciclo. El uso general del ciclo "do while" es:

```
do
{
.....
}while ( expresión lógica )
```

El mismo ejemplo realizado con el ciclo "while" se puede hacer con "do..while":

```
<script language="javascript">
    var total_nota=0;
    var nro=0;
    do
    {
        var nota=parseInt(prompt("Introduzca la nota del alumno "+(nro+1)));
        if (isNaN(nota))
            alert("No introdujo una nota válida");
        else
        {
            total_nota+=parseFloat(nota);
            nro_alumnos++;
        }
    }while (confirm("Desea registrar otra nota?"));
    if (nro_alumnos>0)
    {
        promedio=total_nota/nro_alumnos;
        alert("Se registraron "+nro_alumnos+". El promedio es:"+promedio);
    }else
        alert("No se registraron notas");
</script>
```

Un ejemplo muy común para usar el ciclo "do..while" es para validar los valores leídos por el teclado con "prompt". Al leer el valor, se evalúa si contiene un valor válido, en caso contrario, se vuelve a leer el valor. En el siguiente ejemplo, se lee el valor y trata de convertir a float (usando parseFloat), si no se logra convertir, la función retornará NaN. Si el valor isNaN, se debe repetir el ciclo para volver a leer:

```
<script type="text/javascript" >
    do
    {
        nota=parseFloat(prompt("Introduzca la nota del alumno "));
        if (isNaN(nota))
            alert("No introdujo una nota válida");
    }while (isNaN(nota));
    alert("La nota leída fue: "+nota);
</script>
```



Capítulo 10. FUNCIONES. PARTE 1

10.1.- Definición de Funciones

En ocasiones se tienen muchas instrucciones y deben agruparse en bloques. También en ocasiones hace un bloque de código debe reutilizarse. Para ellos se utilizan las funciones. Una función se debe definir con una sintaxis especial que se muestra a continuación:

```
function nomrefuncion (){
    instrucciones de la función
    ...
}
```

Las funciones se pueden colocar en cualquier parte de la página, siempre entre etiquetas <SCRIPT>, o en archivos externos.

La siguiente imagen muestra un ejemplo de la definición de una función:

The diagram illustrates the definition of a function within an HTML document structure. It features a blue header bar with the text 'F1' in white. Below it, a light blue background contains the HTML code. A grey rectangular box highlights the script block, which contains the function definition. The code is as follows:

```
<body>
    <h1>Funciones</h1>
    <script>
        function leerEdad()
        {
            if (prompt("Edad:") == null)
                alert("Debe escribir la edad");
        }
    </script>
</body>
```

Es una práctica común y muy recomendada definir las funciones en un archivo externo, para así poder reutilizarlas en varios archivos HTML. Se pueden tener funciones en ambos lugares, en el HTML o en el archivo externo. Por ejemplo:

```
1  <html>
2   <head>
3   </head>
4   <body>
5     <h1>Funciones</h1>
6     <script>
7       function leerEdad()
8       {
9         if (prompt("Edad:") == null)
10           alert("Debe escribir la edad")
11       }
12     </script>
13   </body>
14   <script src="validaciones.js"></script>
15 </html>
```



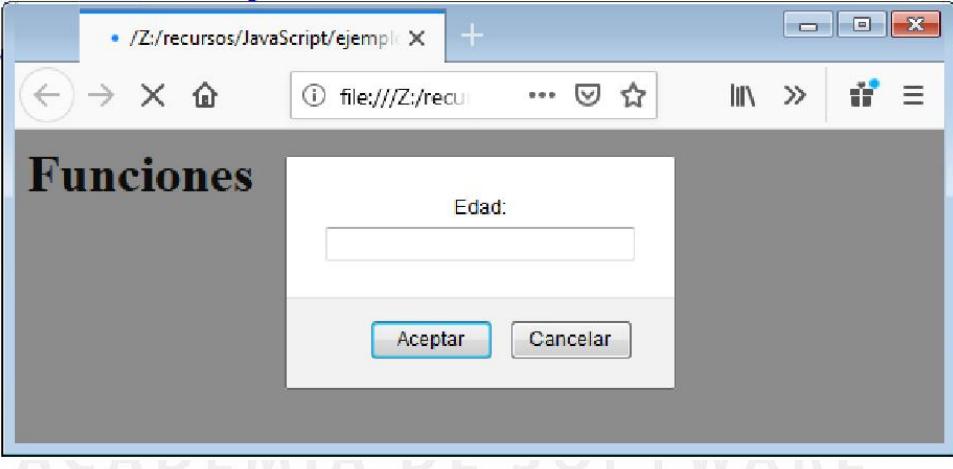
```
1  function leerNombre()
2  {
3    ....
4  }
5
6
7  function leerCorreo()
8  {
9    ....
10 }
11
12 function leerNumero()
13 {
14   ....
15 }
```

10.2.- Llamado de Funciones

El código que está adentro de una función no se ejecutará a menos que en algún lugar la función sea llamada. Para llamar o ejecutar una función, sólo se debe colocar su nombre, seguido de los paréntesis. El siguiente ejemplo muestra como se realiza el llamado de una función:

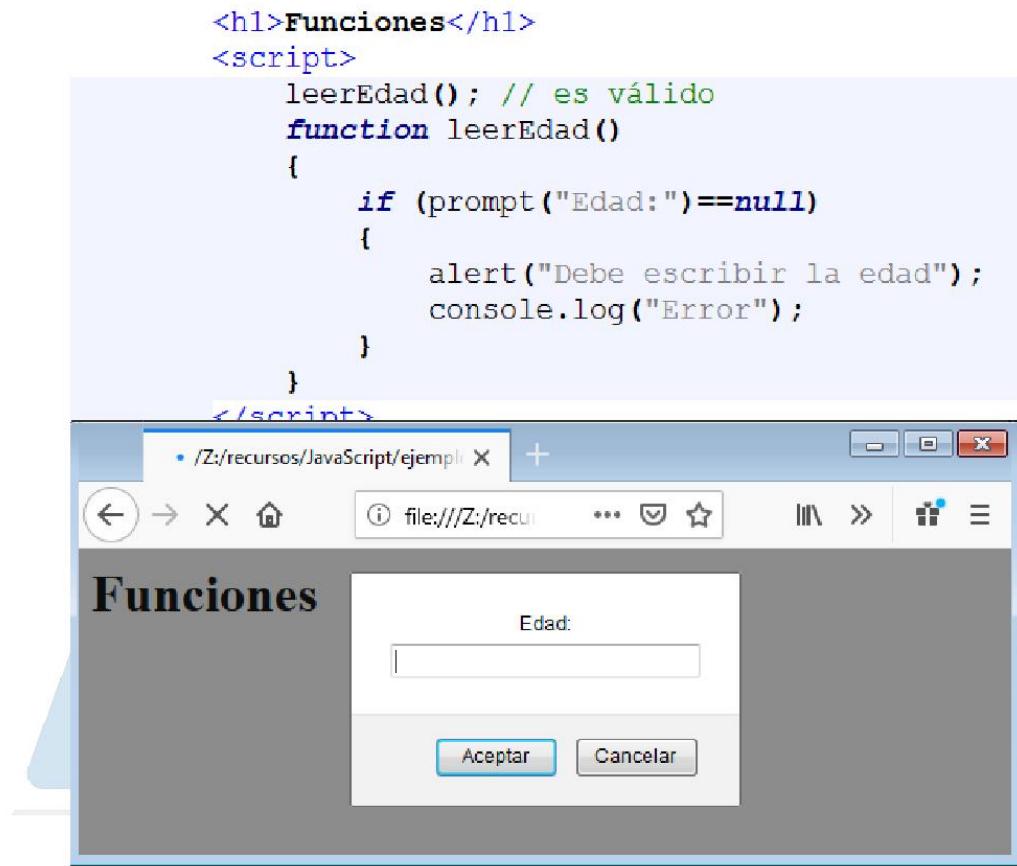
ACADEMIA DE SOFTWARE

```
<script>
    function leerEdad()
    {
        if (prompt("Edad:") == null)
        {
            alert("Debe escribir la edad");
            console.log("Error");
        }
    }
    leerEdad(); // se llama la función
</script>
```



The screenshot shows a web browser window with the title bar reading '/Z:/recursos/JavaScript/ejemplo'. The main content area displays the word 'Funciones'. A modal dialog box is overlaid on the page, titled 'Edad:' with a text input field below it. At the bottom of the dialog are two buttons: 'Aceptar' (Accept) and 'Cancelar' (Cancel). The browser interface includes standard navigation buttons (back, forward, search, etc.) and a toolbar.

Una función puede llamarse siempre y cuando sea conocida por el navegador, por lo tanto, existe una limitación a la hora de definir una función con relación a los lugares desde donde se la llame. Si una función es llamada antes de definirse, pero la definición está en el mismo bloque de script donde es llamada, podrá ejecutarse sin problemas:



Pero si la función es llamada desde un bloque de script que es leído por el navegador antes del bloque de script que contiene la definición de la función, se conseguirá un error:

The screenshot shows a browser window with developer tools open. The code in the script tag is:

```
<script>
    leerEdad(); // error
</script>
<script>
    function leerEdad()
    {
        if (prompt("Edad:") == null)
        {
            alert("Debe escribir la edad");
            console.log("Error");
        }
    }
</script>
<script>
    leerEdad(); // es valido
</script>
```

A red arrow points from the error message in the console to the line `leerEdad(); // error` in the code. A red circle highlights the error message in the console:

ReferenceError: leerEdad is not defined [saber más]

De igual forma, cuando las funciones están en un archivo externo, si se intenta llamar una función contenida en un archivo externo desde un script que esté antes del script que incluye el archivo, ocurrirá un error:

```

<head>
    <meta charset="utf-8" />
</head>
<body>
    <h1>Funciones</h1>
    <script>
        leerNombre(); // error
    </script>
    <script src="validaciones.js"></script>
    <script>
        leerCorreo(); // es valido
    </script>
</body>
</html>

```

```

2   function leerNombre()
3   {
4       ....
5   }
6
7   function leerCorreo()
8   {
9       ....
10  }
11
12  function leerNumero()
13  {
14      ....
15  }

```

En el ejemplo anterior, se intenta llamar a la función "leerNombre", pero se produce un error porque la función aún no existe. Luego, se incluye el archivo externo que contiene varias funciones (entre ellas, la función "leerNombre") y por último, se llama a la función "leerCorreo" que si puede ser ejecutada porque la función está definida en el archivo externo.

10.3.- Ámbito Local vs Global

Las variables que son declaradas afuera de las funciones tienen un ámbito global, es decir, pueden usarse en cualquier script posterior a donde son declaradas, inclusive, adentro de una función. Por ejemplo:

ACADEMIA DE SOFTWARE

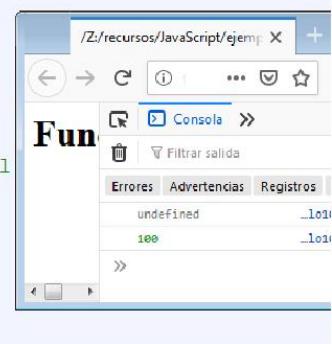
```

<h1>Funciones</h1>
<script>
    var edad; // edad es una variable global

    console.log(edad);
    leerEdad();
    console.log(edad);

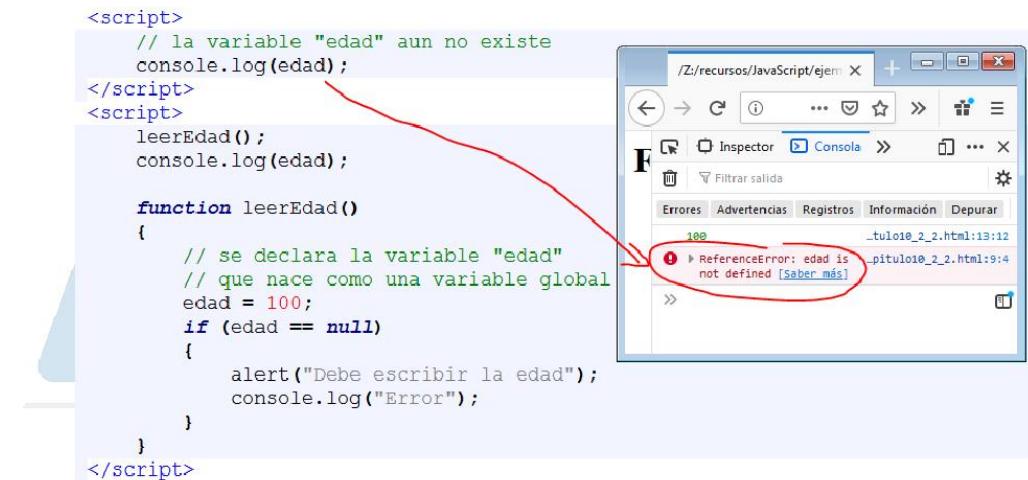
    function leerEdad()
    {
        // se hace referencia a la variable global
        edad = 100;
        if (edad == null)
        {
            alert("Debe escribir la edad");
            console.log("Error");
        }
    }
</script>

```



En el ejemplo anterior, se declara globalmente una variable con el nombre "edad" y se muestra en la consola. El valor por defecto de la variable es "undefined". Luego se llama a la función "leerEdad" que modifica a la variable global "edad", asignándole el valor "100" (no se crea una variable nueva porque ya existe globalmente). Luego del llamado de la función, se vuelve a mostrar en la consola el valor de la variable global "edad", que ahora contiene el valor "100".

Si se asigna valor a una variable que no ha sido declarada previamente, la variable se crea globalmente. Por ejemplo:



The screenshot shows a browser's developer tools console window. On the left, there is a code editor with the following JavaScript code:

```

<script>
    // La variable "edad" aun no existe
    console.log(edad);
</script>
<script>
    leerEdad();
    console.log(edad);

    function leerEdad()
    {
        // se declara la variable "edad"
        // que nace como una variable global
        edad = 100;
        if (edad == null)
        {
            alert("Debe escribir la edad");
            console.log("Error");
        }
    }
</script>

```

A red arrow points from the line `console.log(edad);` in the first script block to the error message in the console. Another red arrow points from the line `edad = 100;` in the function body to the same error message. The error message in the console is:

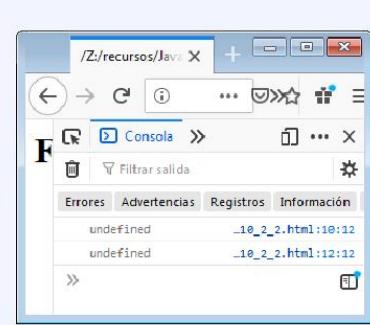
ReferenceError: edad is not defined [Saber más]

Pero si la variable se declara adentro de una función usando la palabra reservada "var", esta variable toma un contexto local y sólo podrá ser usada adentro de la función:

```
<h1>Funciones</h1>
<script>
    var edad; // edad es una variable global

    console.log(edad);
    leerEdad();
    console.log(edad);

    function leerEdad()
    {
        // se crea una nueva variable que solo
        // existirá adentro de la función
        var edad = 100;
        if (edad == null)
        {
            alert("Debe escribir la edad");
            console.log("Error");
        }
    }
</script>
```



En el ejemplo anterior, se declara globalmente una variable con el nombre "edad" y se muestra en la consola. El valor por defecto de la variable es "undefined". Luego se llama a la función "leerEdad" que define una variable local "edad", asignándole el valor "100". Luego del llamado de la función, se vuelve a mostrar en la consola el valor de la variable global "edad", que sigue conteniendo el valor "undefined", porque el valor "100" lo tenía la variable local "edad".

Una variable se puede declarar en un contexto más local aún: el contexto de un bloque (block scope), por ejemplo, el bloque de un "if" o de un ciclo "for". Para lograrlo, se incluyó a partir de ECMAScript 6 - ECMAScript 2015 la palabra reservada "let". El siguiente ejemplo muestra su uso:

```
<script>
    function leerEdad()
    {
        // se crea una nueva variable local
        // pero global en la función
        var edad = 100;
        if (edad == null)
        {
            // se crea una variable local al "if"
            let porc = edad * 0.5;
            // monto es global a la función
            var monto = 1000*porc;
            alert("Debe escribir la edad");
            console.log("Error");
        }
        // se puede acceder al valor de monto
        // siempre y cuando entre al "if"
        console.log(monto);
        // error, porc no existirá porque
        // solo existe en el bloque del "if"
        console.log(porc);
    }
</script>
```

ACADEMIA DE SOFTWARE

Las variables declaradas por let tienen por alcance el bloque en el que se han definido, así mismo, como en cualquier bloque interno. De esta manera, let trabaja muy parecido a var. La más notable diferencia es que el alcance de una variable var es la función contenedora.

Capítulo 11. EVENTOS. PARTE 1

11.1.- Eventos

En el primer capítulo se mencionó que el código JavaScript se puede ejecutar directo o como respuesta a un evento. Hasta ahora se ha visto el código de ejecución directa. Ahora se explicará el código de ejecución como respuesta a eventos.

Los eventos son la manera que se tiene para controlar las acciones de los visitantes y definir un comportamiento de la página cuando se produzcan. Cuando un usuario visita una página web e interactúa con ella se producen los eventos y con JavaScript se puede definir qué se necesita que ocurra cuando se produzcan.

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

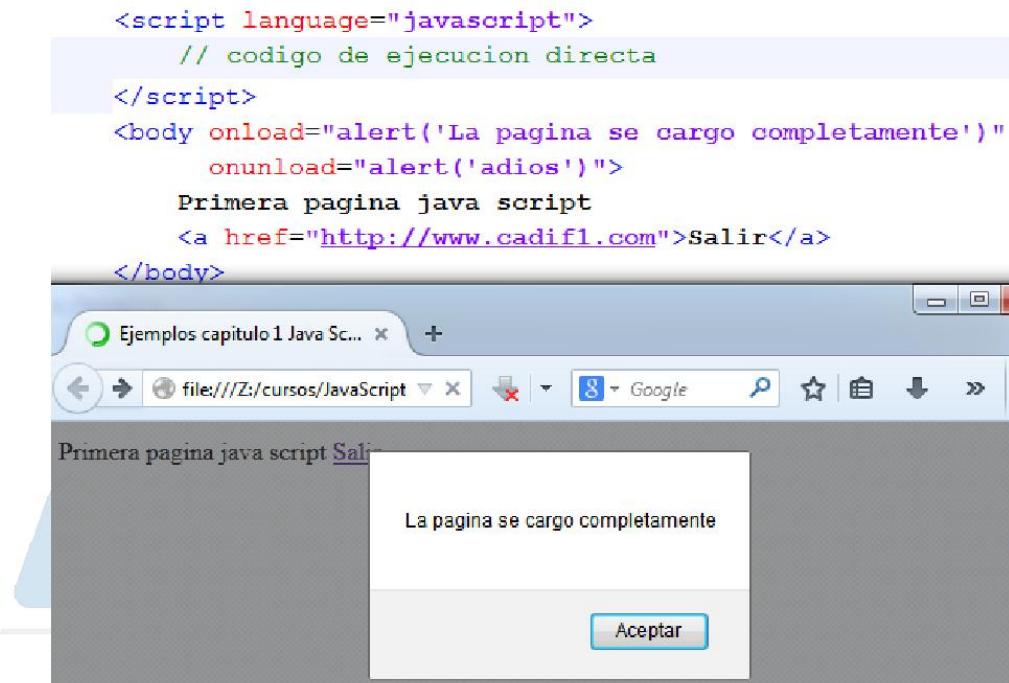
Casi todas las etiquetas HTML pueden responder a eventos. Algunas pueden responder a algunos eventos y otros no. Se pueden programar varios eventos al mismo elemento HTML. Por ejemplo, a la etiqueta body se le puede programar el evento "onload" y "onunload". El evento "onload" se dispara cuando la página termina de cargarse en el navegador. El evento "onunload" se dispara cuando el visitante abandona la página.

Existen muchos otros eventos. A continuación, se describen algunos de ellos y las etiquetas que pueden responder a estos:

- onfocus: ocurre cuando un objeto toma el foco.
- onblur: ocurre cuando un objeto pierde el foco.
- onkeypress: ocurre cuando se presiona una tecla.
- onmousemove: ocurre cuando el cursor del ratón se mueve sobre un objeto.
- onmouseover: ocurre cuando el cursor del ratón se coloca sobre un objeto.
- onsubmit: cuando se intenta enviar un formulario.
- onscroll: cuando se hace scroll en la ventana del navegador.
- onchange: cuando el contenido de un elemento cambia.

11.2.- Manejadores de Eventos

La siguiente imagen muestra un ejemplo de como programar estos eventos:

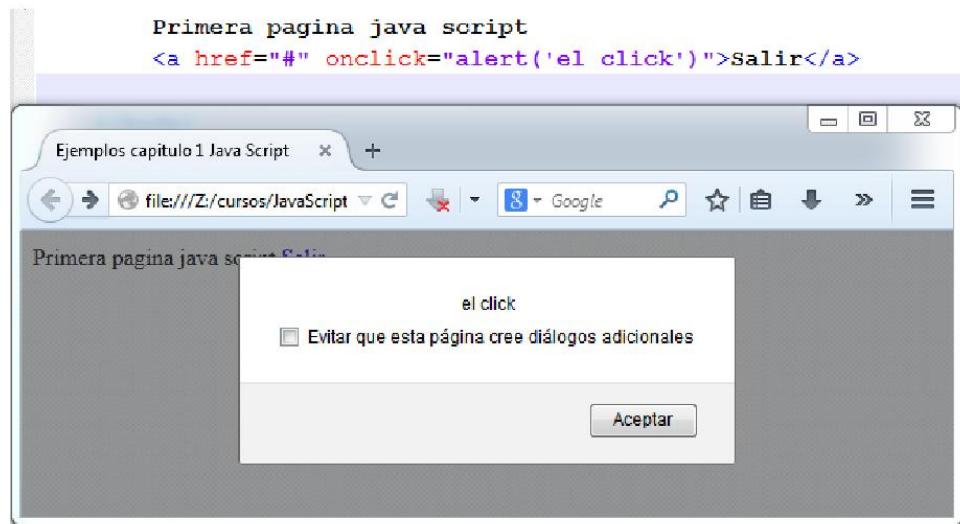


The screenshot shows a web browser window titled "Ejemplos capítulo 1 Java Sc...". The address bar shows "file:///Z:/cursos/JavaScript". The page content is a simple HTML file with the following code:

```
<script language="javascript">
    // codigo de ejecucion directa
</script>
<body onload="alert('La pagina se cargo completamente')"
      onunload="alert('adios')">
    Primera pagina java script
    <a href="http://www.cadif1.com">Salir</a>
</body>
```

The browser displays the text "Primera pagina java script Salir". A JavaScript alert dialog box is open in the center of the page, containing the message "La pagina se cargo completamente" and a single button labeled "Aceptar".

Dentro de los manejadores de eventos se pueden colocar tantas instrucciones como hagan falta, separadas por punto y coma. Lo habitual es colocar una sola instrucción. Otro de los eventos que se programa muy frecuentemente es el "onclick". La etiqueta "href" (etiqueta para hipervínculos) puede responder a este evento. La siguiente imagen muestra un ejemplo de como programar el evento onclick:

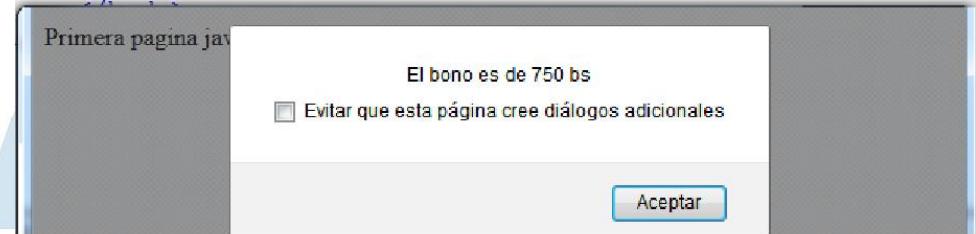


11.3.- Eventos y Funciones

Colocar muchas instrucciones separadas con punto y coma (;) directamente en la etiqueta HTML para manejar un evento no es lo más recomendable. La mejor práctica es colocar en el HTML la llamada a una función. Por esto, combinar eventos y funciones es muy habitual. Veamos un ejemplo de como crear una ventana con una página nueva a partir puro código JavaScript:

```
<script language="javascript">
    function calcular_bono()
    {
        var nro=prompt("Número:");
        var monto=prompt("Monto:");
        if (nro>0)
            bono=monto*0.5;
        else
            bono=monto*0.1;

        alert("El bono es de "+bono+" bs");
    }
</script>
<body>
    Primera pagina java script
    <a href="#" onclick="calcular_bono()">Calcular</a>
</body>
```



Cualquiera de los eventos puede llamar a una función. El siguiente es un ejemplo de cómo llamar a una función desde el evento onload y del evento keypress de un input, el cual tiene un parámetro llamado "event" que le envía la información de la tecla presionada:

```
<script language="javascript">
    function calcular_bono()
    {
        var nro=prompt("Número:");
        var monto=prompt("Monto:");
        if (nro>0)
            bono=monto*0.5;
        else
            bono=monto*0.1;

        alert("El bono es de "+bono+" bs");
    }
    function mostrar_tecla(tecla)
    {
        alert("presion la tecla:"+tecla.key);
    }
</script>
<body onload="calcular_bono()">
    Primera pagina java script
    <input type="text" onkeypress="mostrar_tecla(event)" />
</body>
```

ACADEMIA DE SOFTWARE

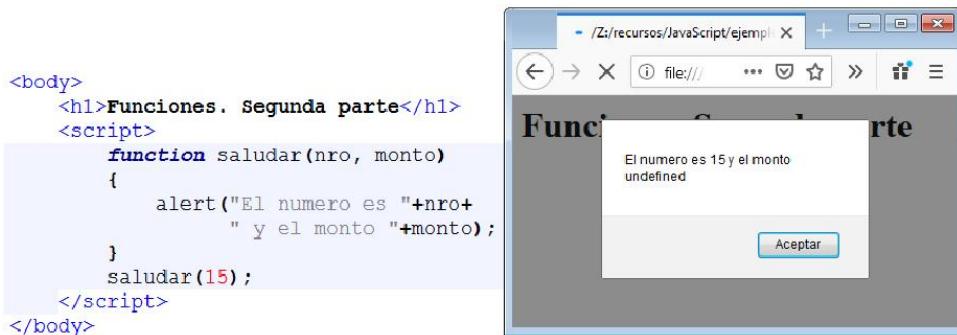
Capítulo 12. FUNCIONES. PARTE 2

12.1.- Parámetros

Las funciones pueden tener parámetros, los cuales se colocan entre los paréntesis en la declaración de la función. JavaScript no maneja los tipos de datos explícitamente, por lo tanto, no es necesario indicar el tipo de dato de los parámetros. En la siguiente imagen se muestra un ejemplo del uso de los parámetros y de cómo debe llamarse la función pasándole parámetros a la misma:

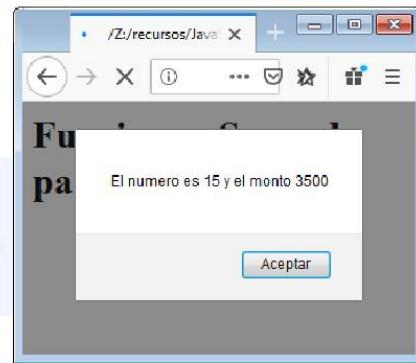


Si una función recibe parámetros y éstos no son pasados en el llamado de la función, el valor de los parámetros no recibidos será "undefined". Por ejemplo:



Si se pasan parámetros de más, no habrá ningún error, aunque los datos de los parámetros adicionales no serán usados:

```
<head>
    <meta charset="utf-8" />
</head>
<body>
    <h1>Funciones. Segunda parte</h1>
    <script>
        function saludar(nro, monto)
        {
            alert("El numero es "+nro+
                  " y el monto "+monto);
        }
        saludar(15,3500,"Jose");
    </script>
</body>
```

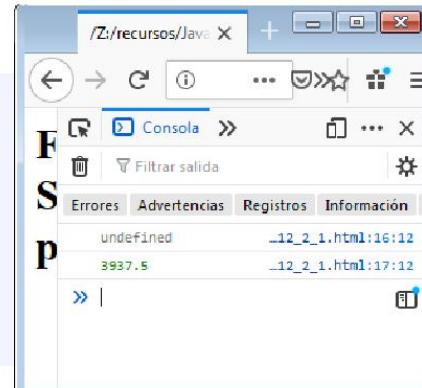


12.2.- Valor de Retorno

Algunas funciones pueden tener un valor de retorno, algunas no. Para que una función retorne un valor debe contener en el bloque de código la instrucción "return" seguido del valor que retornará. Si una función no tiene instrucción "return" el valor de retorno es "undefined".

ACADEMIA DE SOFTWARE

```
<h1>Funciones. Segunda parte</h1>
<script>
    function saludar(monto)
    {
        alert("El monto "+monto);
    }
    function calcular(monto)
    {
        return monto*0.25;
    }
    console.log(saludar(15750));
    console.log(calcular(15750));
</script>
```



El valor que las funciones retornan puede almacenarse en una variable para luego ser utilizada para compararse o mostrarse. El siguiente es un ejemplo de almacenar el valor de retorno de una función en una variable para luego mostrarlo en un alert:

```
<script language="javascript">
    function calcular_bono(nro, monto)
    {
        if (nro>0)
            return monto*0.5;
        else
            return monto*0.1;
    }
    var bono= calcular_bono(5,1500);
    alert("El bono es de "+bono+" bs");
</script>
```



ACADEMIA DE SOFTWARE

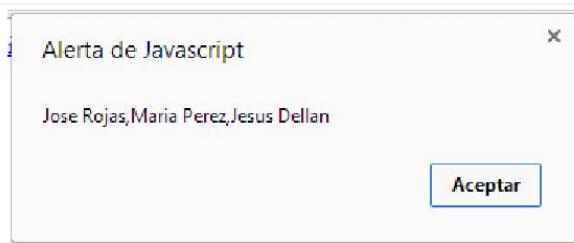
Capítulo 13. ARREGLOS. PARTE 1

13.1.- Definición de Arreglos

Un arreglo es una variable que contiene varios elementos, los cuales están organizados uno después del otro y pueden accederse con un identificador. En JavaScript, los arreglos se declaran usando los corchetes []. A continuación, se muestran unos ejemplos:

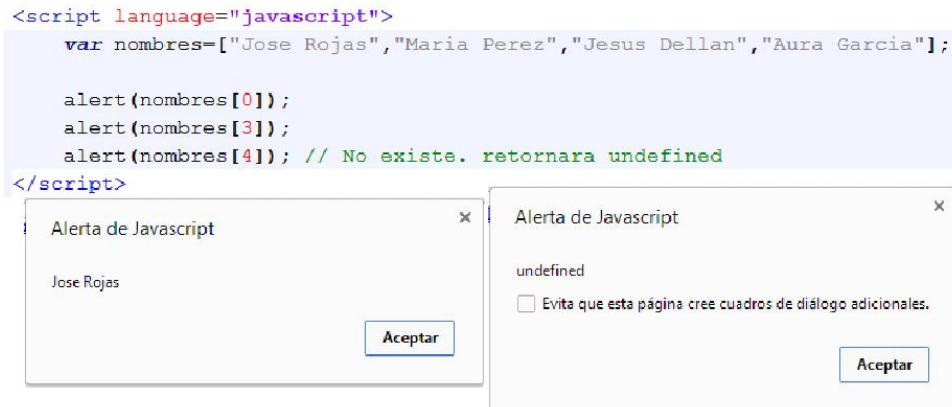
```
<script language="javascript">
    var nombres=["Jose Rojas","Maria Perez","Jesus Dellan"];
    var edades=[32,25,41];
    var aprobados=[true,false,false];

    alert(nombres);
</script>
```



13.2.- Acceso a Los Elementos

Cada uno de los elementos del arreglo pueden ser accedidos haciendo uso de los corchetes y la posición donde está ubicado. La posición del primer elemento es la 0 y la posición del último elemento es la cantidad de elementos del arreglo menos 1 (n-1). A continuación, se muestran un ejemplo de cómo acceder los elementos de un arreglo:



13.3.- Recorrido Del Arreglo

Es muy frecuente la necesidad de recorrer todos los elementos de un arreglo. Teniendo claro del punto anterior de cuales son las posiciones que ocupan los elementos, es necesario realizar un ciclo, que generalmente es el "for". A continuación, se muestra un ejemplo:

```
<script language="javascript">
    var nombres=["Jose Rojas","Maria Perez","Jesus Dellan","Aura Garcia"];

    for (i=0;i<4;i++)
        alert("El nombre "+i+" es "+nombres[i]);

</script>
```

Capítulo 14. ARREGLOS. PARTE 2

14.1.- Funciones de Arreglos

Un arreglo en JavaScript es un objeto, el cual automáticamente obtiene unos métodos (funciones) que permiten manipularlos más fácilmente. Algunas de estas funciones son las siguientes:

- length: indica el tamaño del arreglo. Es una propiedad, por lo tanto, no lleva paréntesis al llamarlo.
- concat(): concatena 2 arreglos en un tercer arreglo.
- join(): retorna todos los elementos del arreglo como una cadena con los valores separados entre sí con un separador especificado.
- pop(): extrae del arreglo el último elemento.
- push(): agrega un elemento al final del arreglo.
- reverse(): invierte los elementos de un arreglo.
- sort(): ordena los elementos del arreglo en forma lexicográfica (alfabéticamente) de menor a mayor.
- indexOf(x): Busca x dentro del array y devuelve la posición de la primera ocurrencia.
- lastIndexOf(): Busca x dentro del array empezando por el final y devuelve la posición de la primera ocurrencia.

En la siguiente imagen se muestra un ejemplo de cómo usar algunos métodos de la clase arreglos:

```
<script language="javascript">
    var nombres=["Jose","Maria","Jesus","Aura"];

    nombres.push("Blanca");

    alert("en el arreglo hay "+nombres.length+" elementos");
    alert(nombres.join(","));

    nombres.sort();
    alert(nombres.join("#"));
    nombres.pop();
    alert(nombres.join("-"));

</script>
```

14.2.- Funciones Para Añadir Elementos o Concatenar Array

Las siguientes funciones permiten agregar elementos a un array, así como unir o concatenar varios arreglos:

`concat(it1, it2, ... , itN)`: Devuelve la concatenación de it1, it2, ..., itN con el array sobre el que se invoca.

`push(x)`: Añade x al final del array como nuevo (o nuevos) elemento, y devuelve la nueva longitud del array.

```
var nombre1 = ["Cecilia", "Luis"];
var nombre2 = ["Emilio", "Tobias", "Jose"];
var todos = nombre1.concat(nombre2);

/*
el valor de todos quedaría así:
["Cecilia", "Luis", "Emilio", "Tobias", "Jose"]
*/

var nombre = "Carlos";
todos.push(nombre);

/*
el valor de todos ahora es:
["Cecilia", "Luis", "Emilio", "Tobias", "Jose", "Carlos"]
*/
```

`unshift(x)`: Añade x al principio del array como nuevo (o nuevos) elementos.

`splice (ind, 0, it1, it2, ... , itN)`: Modifica el array añadiendo los elementos it1, it2, ..., itN, que son insertados en la posición ind (desplazando a los existentes).

```

/*
el valor de todos es:
["Cecilia", "Luis","Emilio", "Tobias", "Jose", "Carlos"]
*/
todos.unshift("Maria","Karla");
/*
el valor de todos ahora es:
["Maria","Karla","Cecilia","Luis","Emilio","Tobias","Jose","Carlos"]
*/
todos.splice(2,0,"Pedro","Jorge");
/*
el valor de todos despues de splice es:
["Maria","Karla","Pedro","Jorge","Cecilia","Luis","Emilio","Tobias","Jose","Carlos"]
*/

```

14.3.- Funciones Para Extraer Elementos

Las siguientes funciones son utilizadas para extraer elementos o partes de un array con eliminación

pop(), Elimina el último elemento del array .

shift(), Elimina el primer elemento del array.

```

/*
el valor de todos es:
["Maria","Karla","Pedro","Jorge","Cecilia","Luis","Emilio","Tobias","Jose","Carlos"]
*/
todos.pop();
/*
luego de pop() el valor de todos es:
["Maria","Karla","Pedro","Jorge","Cecilia","Luis","Emilio","Tobias","Jose"]
*/
todos.shift();
/*
luego de shift() el valor de todos ahora es:
["Karla","Pedro","Jorge","Cecilia","Luis","Emilio","Tobias","Jose"]
*/

```

splice (ind, cuant),Modifica el array borrando cuant elementos a partir del índice ind.

splice (ind, cuant, it1, it2, ... , itN), Modifica el array eliminando cuant elementos e insertando it1, it2, ..., itN,

desde el índice ind.

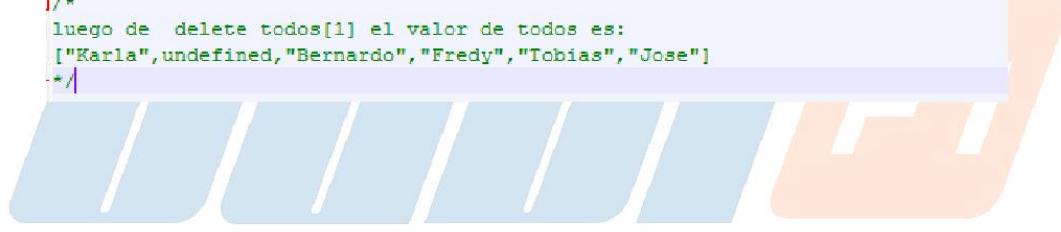
delete A[ind]: Elimina el elemento con índice ind del array , ahora A[ind] es undefined

```
todos.splice(3,2);
/*
luego de splice(3,2) el valor de todos es:
["Karla","Pedro","Jorge","Emilio","Tobias","Jose"]
*/

todos.splice(2,2,"Bernardo","Freddy");
/*
luego de splice(4,2,"Bernardo","Freddy") el valor de todos es:
["Karla","Pedro","Bernardo","Freddy","Tobias","Jose"]
*/

delete todos[1];

/*
luego de delete todos[1] el valor de todos es:
["Karla",undefined,"Bernardo","Freddy","Tobias","Jose"]
*/
```



ACADEMIA DE SOFTWARE