Introducción a JavaScript parte I

JUAN CARLOS CONDE RAMÍREZ

WEB-TECHNOLOGIES

Objetivos

- Conocer los orígenes de JavaScript y sus variaciones iniciales.
- Entender qué es JavaScript y su importancia para las aplicaciones Web actuales.
- Conocer los elementos sintácticos fundamentales para el uso de JavaScript
- Visualizar su funcionamiento a través de ejemplos simples.

Introducción I

- JavaScript es un lenguaje de scripting que se ejecuta del lado del cliente, en un navegador Web.
- Nótese que también puede utilizar JavaScript del lado del servidor y fuera de un navegador, pero no es el tema de este curso.
- Si el navegador es compatible, JavaScript proporciona acceso a la página actual y permite al script determinar las propiedades del cliente, redirigir al usuario a otra página, acceso a las cookies, etc.

Introducción II

- JavaScript nació en septiembre de 1995, paralelamente al lanzamiento de la versión 2.0 del navegador Netscape, el primero en estar equipado con el lenguaje de scripting.

- En aquel momento el lenguaje se llamaba Moca, luego de su salida LiveScript; Netscape concluyó un acuerdo de comercialización con SUN (creador de Java) y decide cambiar el nombre del lenguaje en diciembre de ese año a JavaScript.

- El concepto inmediatamente tuvo éxito: Microsoft lo integró en las versiones de Internet Explorer 3 y las siguientes.

Introducción III

- Por razones legales, una versión del lenguaje por parte de Microsoft se llama JScript.
- JScript era más o menos compatible con JavaScript, pero comenzaron a incluirle funciones adicionales, específicas de Internet Explorer.
- En 1997, se publico el estándar ECMAScript (CE MA-262); donde JavaScript es la primera implementación.

Introducción IV

- La norma define que el *lenguaje* y no las *funciones de los host* de ambientes (por ejemplo, cómo acceder a la ventana actual del navegador o abrir un nuevo).

- ECMAScript se convierte en un estándar ISO en 1998.

- En las cercanías de 1997 o 1998, la guerra de navegadores entre Netscape y Microsoft alcanza su apogeo; los dos proveedores agregaron una nueva característica incompatible a la versión 5 de su navegador.

Introducción V

- El resto es sólo historia:
  - Netscape abandona la idea de publicar la versión 5 del navegador y decide empezar de nuevo con Netscape 6.
  - Internet Explorer pudo aumentar su cuota de mercado a más del 90%.
- También tomó varios años al proyecto Mozilla. Tengamos en cuenta que el navegador Firefox se basa en Mozilla y comienza entonces a ganar cierta cuota de mercado a Microsoft.

Introducción VI

- Desde el punto de vista de JavaScript, algunas cosas han evolucionado en los últimos años. Firefox 1.5, lanzado a finales de 2005, consideró la nueva versión de JavaScript 1.6, pero los cambios son bastante limitados e Internet Explorer estaba lejos de apoyar.

- Pero con Internet Explorer 7 y Firefox 2.0, la época es más interesante para los desarrolladores Web.

- Los navegadores que actualmente aceptan JavaScript, son los siguientes:

Navegadores Compatibles

- Internet Explorer
- Mozilla y sus derivados (Firefox, Epiphany, Camino, Galeon, etc..)
- Ópera
- Konqueror
- Safari
- Chrome

¿Qué es JavaScript? I

- JavaScript, al igual que Flash, Visual Basic Script, es una de las múltiples maneras que han surgido para extender las capacidades del lenguaje HTML.

- Al ser la más sencilla, es por el momento la más extendida. JavaScript no es un lenguaje de programación propiamente dicho como C, C++, Delphi, etc.

- Es un lenguaje script u orientado a documento, como pueden ser los lenguajes de macros que tienen muchos procesadores de texto y plantillas de cálculo.

¿Qué es JavaScript? II

- No se puede desarrollar un programa con JavaScript que se ejecute fuera de un navegador, aunque en este momento comienza a expandirse a otras áreas como la programación en el servidor con Node.js.

- JavaScript es un lenguaje interpretado que se embebe en una página Web HTML.

- Un lenguaje interpretado significa que a las instrucciones las analiza y procesa el navegador en el momento que deben ser ejecutadas.

“Hola Mundo” en JavaScript I

- Nuestro primer programa será el famoso "Hola Mundo", es decir un programa que muestre en el documento HTML el mensaje "Hola Mundo".

```
<html>
    <head></head>
    <body>
        <script type=“text/javascript”>
                document.write(‘Hola Mundo’);
        </script>
    </body>
</html>
```

- El programa en JavaScript debe ir encerrado entre la marca script e inicializada la propiedad `type` con la cadena `text/javascript`:

```
<script type=“text/javascript”>
</script>
```

“Hola Mundo” en JavaScript II

- Para imprimir caracteres sobre la página debemos llamar al comando ‘`write`’ del objeto `document`.

- La información a imprimirse debe ir entre comillas y encerrada entre paréntesis. Todo lo que indicamos entre comillas aparecerá tal cual dentro de la página HTML.

- Es decir, si pedimos al navegador que ejecute esta página mostrará el texto ‘`Hola Mundo`’.

“Hola Mundo” en JavaScript III

- Cada vez que escribimos una instrucción finalizamos con el carácter punto y coma.

- Es muy importante considera que JavaScript es sensible a mayúsculas y minúsculas; no es lo mismo escribir `document.write` que `DOCUMENT.WRITE` (la primera forma es la correcta, la segunda forma provoca un error de sintaxis).

- Nos acostumbraremos a prestar atención cada vez que escribamos en minúsculas o mayúsculas para no cometer errores sintácticos. Ya veremos más adelante, que los nombres de funciones llevan letras en mayúsculas

Variables I

- Una variable es un depósito donde hay un valor. Consta de un nombre y pertenece a un tipo (numérico, cadena de caracteres, etc.).

- Tipos de variable:
  - Una variable puede almacenar:
  - Valores Enteros (100, 260, etc.)
  - Valores Reales (1.24, 2.90, 5.01, etc.)
  - Cadenas de caracteres ('Juanito', 'Compras', 'Listado', etc.)
  - Valores lógicos (true, false)

- Existen otros tipos de variables que veremos más adelante.

Variables II

- Las variables son nombres que ponemos a los lugares donde almacenamos la información.

- En JavaScript, deben comenzar por una letra o un subrayado (_), pudiendo haber además dígitos entre los demás caracteres. Una variable no puede tener el mismo nombre de una palabra clave del lenguaje.

- Una variable se define anteponiéndole la palabra clave `var`:
  - `var dia;`

Variables III

- se pueden declarar varias variables en una misma línea:
  - `var dia, mes, anio;`

- a una variable se la puede definir e inmediatamente inicializarla con un valor:
  - `var edad = 20;`

- o en su defecto en dos pasos:
  - `var edad;`
  - `edad = 20;`

Variables IV

- Elección del nombre de una variable:
  - Debemos elegir nombres de variables representativos. En el ejemplo anterior los nombres `dia`, `mes`, `anio` son lo suficientemente claros para darnos una idea acabada sobre su contenido, una mala elección de nombres hubiera sido llamarlas `a`, `b` y `c`.
  - Podemos darle otros buenos nombres. Otros no son tan representativos, por ejemplo `d`, `m`, `a`.
  - Posiblemente cuando estemos resolviendo un problema dicho nombre nos recuerde que almacenamos el `dia`, pero pasado un tiempo lo olvidaríamos.

Variables V

```
<html>
    <head></head>
    <body>
        <script type = “text/javascript”>
            var nombre = ‘Juan’;
            var edad = 10;
            var altura = 1.92;
            var casado = false;
            document.write( nombre );
            document.write( ‘<br>’ );
            document.write( edad );
            document.write( ‘<br>’ );
            document.write( altura );
            document.write( ‘<br>’ );
            document.write( casado );
        </script>
    </body>
</html>
```

Variables VI

- Cuando imprimimos una variable, no la debemos disponer entre simples comillas (en caso de hacer esto, aparecerá el nombre de la variable y no su contenido).

- Los valores de las variables que almacenan nombres (es decir, son cadenas de caracteres) deben ir encerradas entre comillas simples o dobles.

- Los valores de las variables enteras (en este ejemplo la variable edad) y reales no deben ir encerradas entre comillas. Cada instrucción finaliza con un punto y coma.

Variables VII

- Las variables de tipo *boolean* pueden almacenar solo dos valores: `true` o `false`.

- El resultado al visualizar la página debe ser 4 líneas similares a éstas:
  - Juan
  - 10
  - 1.92
  - False

- Es decir, que se muestran los contenidos de las 4 variables.

Variables VIII

- Una variable es de un tipo determinado cuando le asignamos un valor:
  - `var edad = 10;` `//Es de tipo entera ya que le asignamos un valor //entero.`
  - `var nombre = 'juan';` `//Es de tipo cadena.`
- Para mostrar el contenido de una variable en una página debemos utilizar la función '`write`' que pertenece al objeto `document`.

Variables IX

- Recordemos que el lenguaje JavaScript es sensible a mayúsculas y minúsculas y no será lo mismo si *tipeamos*:
  - `Document.Write(nombre);`
- Esto porque no existe el objeto '`Document`' sino el objeto '`document`' (con d minúscula), lo mismo no existe la función '`Write`' sino '`write`', este es un error muy común cuando comenzamos a programar en JavaScript.

Entrada de datos por teclado I

- Para la entrada de datos por teclado tenemos la función *prompt*.
- Cada vez que necesitamos ingresar un dato con esta función, aparece una ventana donde cargamos el valor.
- Hay otras formas más sofisticadas para la entrada de datos en una página HTML, pero para el aprendizaje de los conceptos básicos de JavaScript nos resultará más práctica esta función.

Entrada de datos por teclado II

- Para ver su funcionamiento analicemos este ejemplo:

```
<html>
    <head></head>
    <body>
        <script type = “text/javascript”>
            var nombre;
            var edad;
            nombre = prompt(‘Ingresa tu nombre:’, ‘’);
            edad = prompt(‘Ingresa tu edad:', ‘’);
            document.write(‘Hola ’);
            document write(nombre);
            document.write(‘ así que tienes ’);
            document.write(edad);
            document.write(‘ años’);
        </script>
    </body>
</html>
```

Entrada de datos por teclado III

- La sintaxis de la función *prompt* es:

```
<variable receptora> = prompt( <mensaje a mostrar>, <valor inicial a mostrar> );
```

- La función *prompt* tiene dos parámetros: uno es el mensaje y el otro el valor inicial a mostrar.

Estructuras secuenciales I

- Cuando en un problema sólo participan operaciones, entradas y salidas se la denomina estructura secuencial.

- El problema anterior, donde se ingresa el nombre de una persona y su edad se trata de una estructura secuencial.

- Ejemplo de otro algoritmo con estructura secuencial:
  - “Capturar dos números por teclado e imprimir su suma y su producto”.

Estructuras secuenciales II

```
<html>
    <head>
        <script type="text/javascript">
            var valor1;
            var valor2;
            valor1 = prompt('Introducir primer número:', '');
            valor2 = prompt('Introducir segundo número', '');
            var suma = parseInt(valor1)+parseInt(valor2);
            var producto = parseInt(valor1)*parseInt(valor2);
            document.write('La suma es ');
            document.write(suma);
            document.write('<br>');
            document.write('El producto es ');
            document.write(producto);
        </script>
    </head>
    <body></body>
</html>
```

Estructuras secuenciales II

- Lo primero que debemos tener en cuenta es que si queremos que el operador '+' sume los contenidos de los valores numéricos introducidos por teclado, debemos llamar a la función `parseInt` y pasar como parámetro las variables `valor1` y `valor2` sucesivamente.

- Con esto logramos que el operador más, sume las variables como enteros y no como cadenas de caracteres.

- Si por ejemplo sumamos `1 + 1` sin utilizar la función `parseInt` el resultado será `11` en lugar de `2`, ya que el operador `+` concatena las dos cadenas.

Estructuras secuenciales II

- En JavaScript, como no podemos indicarle de qué tipo es la variable, requiere mucho más cuidado cuando operamos con sus contenidos.

- Este problema es secuencial ya que ingresamos dos valores por teclado, luego hacemos dos operaciones y por último mostramos los resultados.

Estructuras condicionales simples I

- No todos los problemas pueden resolverse empleando estructuras secuenciales. Cuando hay que tomar una decisión aparecen las estructuras condicionales.

- En nuestra vida diaria se nos presentan situaciones donde debemos decidir. Por supuesto que en un problema se combinan estructuras secuenciales y condicionales.

- Cuando se presenta la elección tenemos la opción de realizar una actividad o no realizarla.

Estructuras condicionales simples II

- En una estructura condicional simple por el camino del verdadero hay actividades y por el camino del falso no hay actividades.

- Por el camino del verdadero pueden existir varias operaciones, entradas y salidas, inclusive ya veremos que puede haber otras estructuras condicionales.

- Ejemplo:
  - Capturar la calificación de un alumno.
  - Mostrar un mensaje que aprobó si tiene una nota mayor o igual a 4.

Estructuras condicionales simples III

```
<html>
    <head></head>
    <body>
        <script type="text/javascript">
        var nombre;
        var nota;
        nombre = prompt('Ingresa tu nombre:', '');
        Nota = prompt('Ingresa tu nota:', '');
        if (nota>=4)
        {
             document.write(nombre+' esta aprobado con un '+nota);
        }
        </script>
    </body>
</html>
```

Estructuras condicionales simples IV

- Aparece la instrucción `if` en el lenguaje JavaScript.

- La condición debe ir entre paréntesis. Si la condición se verifica verdadera se ejecuta todas las instrucciones que se encuentran encerradas entre las llaves de apertura y cerrado seguidas al `if`.

- Para disponer condiciones en un `if` podemos utilizar alguno de los siguientes operadores relacionales.

Estructuras condicionales simples V

operador	comparación
>	mayor
>=	mayor o igual
<	menor
<=	menor o igual
!=	distinto
==	igual

Estructuras condicionales simples VI

- Siempre debemos tener en cuenta que en la condición del `if` deben intervenir una variable un operador relacional y otra variable o valor fijo.

- Otra cosa que hemos incorporado es el operador `+` para cadenas de caracteres: `document.write(nombre+' esta aprobado con un '+nota);`

- Con esto hacemos más corta la cantidad de líneas de nuestro programa, recordemos que veníamos haciéndolo de la siguiente forma:

```
document.write(nombre);
document.write(' esta aprobado con un ');
document.write(nota);
```

Estructuras condicionales compuestas I

- Cuando se presenta la elección tenemos la opción de realizar una actividad u otra.

- Es decir tenemos actividades por el verdadero y por el falso de la condición. Lo más importante que hay que tener en cuenta es que se realizan las actividades de la rama del verdadero o las del falso, NUNCA se realizan las actividades de las dos ramas.

- En una estructura condicional compuesta tenemos entradas, salidas, operaciones, tanto por la rama del verdadero como por la rama del falso.

Estructuras condicionales compuestas II

- Ejemplo:
  - Implementar un programa que lea dos números distintos y muestre el mayor de ellos.

```
<html>
    <head></head>
    <body>
        <script type="text/javascript">
        var num1,num2;
        num1 = prompt('Ingresa el primer número:', '');
        num2 = prompt('Ingresa el segundo número:', '');
        num1 = parseInt(num1);
        num2 = parseInt(num2);
        if (num1>num2){
            document.write('el mayor es '+num1);
        }
        else{
            document.write('el mayor es '+num2);
        }
        </script>
    </body>
</html>
```

Estructuras condicionales compuestas III

- La función `prompt` retorna un *string* por lo que debemos convertirlo a entero cuando queremos saber cual de los dos valores es mayor numéricamente.

- En el lenguaje JavaScript una variable puede ir cambiando el tipo de dato que almacena a lo largo de la ejecución del programa. Más adelante veremos qué sucede cuando preguntamos cuál de dos *string* es mayor.

- Estamos en presencia de una estructura condicional compuesta ya que tenemos actividades por la rama del *verdadero* y del *falso*.

Estructuras condicionales compuestas IV

- La estructura condicional compuesta tiene la siguiente codificación:

```
if (<condición>){
    <Instruccion(es)>
}
else{
    <Instruccion(es)>
}
```

- Es igual que la estructura condicional simple salvo que aparece la palabra clave "`else`" y posteriormente un bloque `{}` con una o varias instrucciones.

- Si la condición del `if` es verdadera se ejecuta el bloque que aparece después de la condición, en caso que la condición resulte falsa se ejecuta la instrucción o bloque de instrucciones que indicamos después del `else`.

Estructuras condicionales anidadas I

- Decimos que una estructura condicional es anidada cuando por la parte del verdadero o el falso de una estructura condicional hay otra estructura condicional.

- Ejemplo: Proponer un programa que solicite por teclado tres calificaciones de un alumno, calcular el promedio e imprimir alguno de estos mensajes:
  - Si el promedio es >=7 mostrar “Aprobado”.
  - Si el promedio es >=4 y <7 mostrar “Regular”.
  - Si el promedio es <4 mostrar “Reprobado”.

Estructuras condicionales anidadas II

```
<html>
    <head></head>
    <body>
        <script type="text/javascript">
            var nota1,nota2,nota3;

            nota1 = prompt('Ingresa 1ra. nota:', '');
            nota2 = prompt('Ingresa 2da. nota:', '');
            nota3 = prompt('Ingresa 3ra. nota:', '');

            //Convertimos los 3 string en enteros
            nota1 = parseInt(nota1);
            nota2 = parseInt(nota2);
            nota3 = parseInt(nota3);

            var pro;
            pro = (nota1+nota2+nota3)/3;
```

Estructuras condicionales anidadas III

```
            if (pro>=7)
            {
                 document.write('aprobado');
            }
            else{
                if (pro>=4)
                {
                     document.write('regular');
                }
                else
                {
                     document.write('reprobado');
                }
            }
        </script>
    </body>
</html>
```

Operadores lógicos I

- El operador &&, traducido se lo lee como "Y". Se emplea cuando en una estructura condicional se disponen dos condiciones.

- Cuando vinculamos dos o más condiciones con el operador "&&" las dos condiciones deben ser *verdaderas* para que el resultado de la condición compuesta de Verdadero y continúe por la parte del verdadero de la estructura condicional.

- El operador ||, se lee como "O". Se emplea cuando en una estructura condicional se disponen dos condiciones.

Operadores lógicos: AND

- Si la condición 1 es Verdadera o la condición 2 es Verdadera, luego ejecutar la rama del Verdadero.

- Recordemos que la condición debe ir entre paréntesis en forma obligatoria

- La utilización de operadores lógicos permiten en muchos casos, plantear algoritmos más cortos y comprensibles.

Estructura `switch`

- La instrucción `switch` es una alternativa para remplazar los `if/else if`.

- De todos modos se puede aplicar en ciertas situaciones donde la condición se verifica si es igual a cierto valor. No podemos preguntar por mayor o menor.

- Con un ejemplo sencillo veremos cual es su sintaxis:

  - Proponer un programa que solicite un valor entre 1 y 5. Luego mostrar en español el valor ingresado. Mostrar un mensaje de error en caso de haber ingresado un valor que no se encuentre en dicho rango.

Ejemplo de `switch`

```
</head>
     <body>
          <script type="text/javascrip">
               var valor;
               valor = prompt('Ingresar un valor comprendido entre 1 y 5:', '' );
               //Convertimos a entero
               valor = parseInt(valor);
               switch (valor)
               {
                    case 1: document.write('uno');
                              break;
                    case 2: document write('dos');
                              break;
                    case 3: document.write('tres');
                              break;
                    case 4: document.write('cuatro');
                              break;
                    case 5: document.write('cinco');
                              break;
                    default:document.write('debe ingresar un valor comprendido entre 1 y 5.');
               }
          </script>
     </body>
</html>
```

Estructura `switch` II

- Debemos tener en cuenta que la variable que analizamos debe ir después de la instrucción `switch` entre paréntesis.

- Cada valor que se analiza debe ir luego de la palabra clave 'case' y seguido de los dos puntos, las instrucciones a ejecutar, en caso de verificar dicho valor la variable que analiza el `switch`.

- Es importante disponer la palabra clave `'break'` al finalizar cada caso.

Estructura `switch` III

- La instrucciones que hay después de la palabra clave `'default'` se ejecutan en caso que la variable no se verifique en algún case.

- De todos modos el *default* es opcional en esta instrucción.

- Plantearemos un segundo problema para ver que podemos utilizar variables de tipo cadena con la instrucción `switch`.

Ejemplo de `switch` II

- Ingresar por teclado el nombre de un color (rojo, verde o azul), luego pintar el fondo de la ventana con dicho color:

```
<html>
     <head></head>
     <body>
          <script type="text/javascript">
               var col;
               col = prompt('Ingresa el color con que quierar pintar el fondo de
               la ventana (rojo, verde, azul)' , '' );
               switch (col)
               {
                    case 'rojo': document.bgColor='#ff0000';
                              break;
                    case 'verde': document.bgColor='#00ff00';
                              break;
                    case 'azul': document.bgColor='#0000ff';
                              break;
               }
          </script>
     </body>
</html>
```

Ejemplo de `switch` II

- Cuando verificamos cadenas debemos encerrarlas entre comillas el valor a analizar:

```
case 'rojo': document.bgColor='#ff0000';
break;
```

- Para cambiar el color del fondo de la ventana debemos asignarle a la propiedad `bgColor` del objeto `document` el color a asignar (el color está formado por tres valores hexadecimales que representan la cantidad de rojo, verde y azul), en este caso al valor de rojo le asignamos `ff` (255 en decimal) es decir el valor máximo posible, luego `00` para verde y azul (podemos utilizar algún software de gráficos para que nos genere los tres valores).

Estructura `while` I

- Hasta ahora hemos empleado estructuras secuenciales y condicionales. Existe otro tipo de estructuras tan importantes como las anteriores que son las estructuras repetitivas.

- Una estructura repetitiva permite ejecutar una instrucción o un conjunto de instrucciones varias veces.

- Una ejecución repetitiva de sentencias se caracteriza por:
  - La o las sentencias que se repiten.
  - El test o prueba de condición antes de cada repetición, que motivará que se repitan o no las sentencias.

Estructura `while` II

- El funcionamiento del `while`: En primer lugar se verifica la condición, si la misma resulta verdadera se ejecutan las operaciones que indicamos entre las llaves que le siguen al `while`.

- En caso que la condición sea falsa continúa con la instrucción siguiente al bloque de llaves. El bloque se repite mientras la condición sea verdadera.

- Importante: Si la condición siempre retorna verdadero estamos en presencia de un ciclo repetitivo infinito.

Estructura `while` III

- Dicha situación es un error de programación, nunca finalizará el programa.

- Ejemplo: Proponer un programa que imprima en pantalla los números del 1 al 100.
  - Sin conocer las estructuras repetitivas podemos resolver el problema empleando una estructura secuencial.
  - Inicializamos una variable con el valor 1, luego imprimimos la variable, incrementamos nuevamente la variable y así sucesivamente.
  - Pero esta solución es muy larga.

Estructura `while` IV

- La mejor forma de resolver este problema es emplear una estructura repetitiva.

```
<html>
    <head></head>
    <body>
        <script type="text/javascript">
            var x;
            x=1;
            while (x<=100)
            {
                document.write(x);
                document.write('<br>');
                x=x+1;
            }
        </script>
    </body>
</html>
```

- Para que se impriman los números, uno en cada línea, agregamos la marca HTML de `<br>`.

Concepto de *acumulador* I

- Explicaremos el concepto de un acumulador con un ejemplo.

- Problema: Desarrollar un programa que permita la carga de 5 valores por teclado y nos muestre posteriormente la suma.

Concepto de *acumulador* II

```
<html>
    <head></head>
    <body>
        <script type="text/javascript">
            var x=1;
            var suma=0;
            var valor;
            while (x<=5)
            {
                valor = prompt('Ingresa el valor:', '');
                valor = parseInt(valor);
                suma = suma+valor;
                x = x+1;
            }
            document.write("La suma de los valores es "+suma+"<br>");
        </script>
    </body>
</html>
```

Concepto de *acumulador* III

- Para este problema, si lo comparamos con los anteriores, llevamos un contador llamado `x` que nos sirve para contar las veces que se debe repetir el `while`.

- También aparece el concepto de *acumulador* (un acumulador es un tipo especial de variable que se incrementa o decrementa con valores variables durante la ejecución del programa). En este caso, el nombre de nuestro acumulador es `suma`.

- Cada ciclo que se repita la estructura repetitiva, la variable suma se incrementa con el contenido ingresado en la variable valor.

Concepto de *acumulador* IV

- Para este problema, si lo comparamos con los anteriores, llevamos un contador llamado `x` que nos sirve para contar las veces que se debe repetir el `while`.

- También aparece el concepto de *acumulador* (un acumulador es un tipo especial de variable que se incrementa o decrementa con valores variables durante la ejecución del programa). En este caso, el nombre de nuestro acumulador es `suma`.

- Cada ciclo que se repita la estructura repetitiva, la variable suma se incrementa con el contenido ingresado en la variable valor.

Concepto de *acumulador* V

- La prueba del diagrama se realiza dándole valores a las variables:

valor	*suma*	*x*
0	0	1

- (Antes de entrar a la estructura repetitiva estos son los valores)

5	5	1
16	21	2
7	28	3
10	38	4
2	40	5

Concepto de *acumulador* VI

- Este es un seguimiento del programa planteado.

- Los números que toma la variable valor dependerá de qué cifras cargue el operador durante la ejecución del programa.

- Hay que tener en cuenta que cuando en la variable valor se carga el primer valor (en este ejemplo es el valor 5), al cargarse el segundo valor (16), el valor anterior 5 se pierde, por ello la necesidad de ir almacenando en la variable suma el valor acumulado de los valores ingresados.

Estructura `do-while`

- La sentencia `do/while` es otra estructura repetitiva, la cual ejecuta al menos una vez su bloque repetitivo, a diferencia del `while` que puede no ejecutar el bloque.

- Esta estructura repetitiva se utiliza cuando conocemos de antemano que por lo menos una vez se ejecutará el bloque repetitivo.

- La condición de la estructura está abajo del bloque a repetir, a diferencia del `while` que está en la parte superior.

Estructura `do-while` II

- Finaliza la ejecución del bloque repetitivo cuando la condición retorna falso, es decir igual que el `while`.

- Problema: Escribir un programa que solicite la introducción de un número entre 0 y 999, y nos muestre un mensaje de cuántos dígitos tiene el mismo. Finalizar el programa cuando se escriba el valor 0.

- En este problema por lo menos se carga un valor. Si se carga un valor menor a 10 se trata de un número de una cifra, si es mayor a 10 pero menor a 100 se trata de un valor de dos dígitos, en caso contrario se trata de un valor de tres dígitos.

Estructura do-while III

- Este bloque se repite mientras se ingresa en la variable 'valor' un número distinto a 0.

```
<html>
      <head></head>
      <body>
            <script type = “text/javascript”>
                  var valor;
                  do{
                        valor = prompt(‘Ingresa un valor entre 0 y 999:’, ‘’);
                        valor = parseInt(valor);
                        document.write(‘El valor ’+valor+‘ tiene ’);
                        if (valor<10){
                              document.write(‘Tiene 1 dígitos’);
                        }
                        else{
                              if (valor<100){
                                    document.write(‘Tiene 2 dígitos’);
                              }
                              else{
                                    document.write(‘Tiene 3 dígitos’);
                              }
                        }
                        document write(‘<br>’);
                  }while(valor!=0);
            </script>
      </body>
</html>
```

Estructura `for` I

- Cualquier problema que requiera una estructura repetitiva se puede resolver empleando la estructura `while`.

- Pero hay otra estructura repetitiva cuyo planteamiento es más sencillo en ciertas situaciones.

- Esta estructura se emplea en aquellas situaciones en las cuales conocemos la cantidad de veces que queremos que se ejecute el bloque de instrucciones.
  - Ejemplo: cargar 10 números, ingresar 5 notas de alumnos, etc.

Estructura `for` II

- Conocemos de antemano la cantidad de veces que queremos que el bloque se repita.

- Por último, hay que decir que la ejecución de la sentencia break dentro de cualquier parte del ciclo provoca la salida inmediata del mismo.

- Sintaxis:

```
for (<Inicialización> ; <Condición> ; <Incremento o Decremento>)
{
        <Instrucciones>
}
```

Estructura `for` III

- Esta estructura repetitiva tiene tres argumentos: variable de inicialización, condición y variable de incremento o decremento.

- Funcionamiento:
  - Primero se ejecuta por única vez el primer argumento.
  - Por lo general se inicializa una variable.
  - El segundo paso es evaluar la (Condición), en caso de ser verdadera se ejecuta el bloque, en caso contrario continúa el programa.
  - El tercer paso es la ejecución de las instrucciones.
  - El cuarto paso es ejecutar el tercer argumento (Incremento o Decremento).
  - Luego se repiten sucesivamente del Segundo al Cuarto Paso.

Estructura `for` IV

- Este tipo de estructura repetitiva se utiliza generalmente cuando sabemos la cantidad de veces que deseamos que se repita el bloque.

- Ejemplo: Mostrar en la pantalla los números del 1 al 10.

```
<html>
    <head></head>
        <body>
        <script type="text/javascript">
            var f;
            for(f=1; f<=10; f++)
            {
                document.write(f+" ");
            }
        </script>
    </body>
</html>
```

Funciones I

- En programación es muy frecuente que un determinado procedimiento de cálculo definido por un grupo de sentencias tenga que repetirse varias veces, ya sea en un mismo programa o en otros programas, lo cual implica que se tenga que escribir tantos grupos de aquellas sentencias como veces aparezca dicho proceso.

- La herramienta más potente con que se cuenta para facilitar, reducir y dividir el trabajo en programación, es escribir aquellos grupos de sentencias una sola y única vez bajo la forma de una función.

- Un programa es una cosa compleja de realizar y por lo tanto es importante que esté bien estructurado y también que sea inteligible para las personas.

Funciones II

- Si un grupo de sentencias realiza una tarea bien definida, entonces puede estar justificado el aislar estas sentencias formando una función, aunque resulte que sólo se le llame o use una vez.

- Hasta ahora hemos visto como resolver un problema planteando un único algoritmo.

- Con funciones podemos segmentar un programa en varias partes. Frente a un problema, planteamos un algoritmo, éste puede constar de pequeños algoritmos

Funciones III

- Una función es un conjunto de instrucciones que resuelven una parte del problema y que puede ser utilizado (llamado) desde diferentes partes de un programa.

- Consta de un nombre y parámetros. Con el nombre llamamos a la función, es decir, hacemos referencia a la misma. Los parámetros son valores que se envían y son indispensables para la resolución del mismo.

- La función realizará alguna operación con los parámetros que le enviamos. Podemos cargar una variable, consultarla, modificarla, imprimirla, etc. Incluso los programas más sencillos tienen la necesidad de fragmentarse.

Funciones IV

- Las funciones son los únicos tipos de subprogramas que acepta JavaScript. Tienen la siguiente estructura:

```
function <nombre de función>(argumento1, argumento2, ..., argumento n)
{
        <código de la función>
}
```

- Debemos buscar un nombre de función que nos indique cuál es su objetivo (Si la función recibe un `string` y lo centra, tal vez deberíamos llamarla centrar Titulo).

- Veremos que una función puede variar bastante en su estructura, puede tener o no parámetros, retornar un valor, etc.

Funciones V

- Ejemplo: Mostrar un mensaje que se repita 3 veces en la página con el siguiente texto:
  - 'Cuidado' 'Ingresa tu documento correctamente'
  - 'Cuidado' 'Ingresa tu documento correctamente'
  - 'Cuidado' 'Ingrese su documento correctamente'

- La solución sin emplear funciones es:

```
<html>
     <head></head>
     <body>
          <script type = "text/javascript">
               document.write("Cuidado<br>");
               document.write("Ingresa tu documento correctamente<br>");
               document.write("Cuidado<br>");
               document.write("Ingresa tu documento correctamente<br>");
               document.write("Cuidado<br>");
               document.write("Ingresa tu documento correctamente<br>");
          </script>
     </body>
</html>
```

Funciones VI

- Empleando una función:

```
<html>
    <head></head>
    <body>
        <script type="text/javascript">
            function mostrarMensaje()
            {
                document.write("Cuidado<br>");
                document.write("Ingresa tu documento correctamente<br>");
            }
            mostrarMensaje();
            mostrarMensaje();
            mostrarMensaje();
        </script>
    </body>
</html>
```

Funciones VII

- Recordemos que JavaScript es sensible a mayúsculas y minúsculas.

- Si establecemos como nombre a la función `mostrarTitulo` (es decir la segunda palabra con mayúscula) debemos respetar este nombre cuando la llamemos a dicha función.

- Es importante notar que para que una función se ejecute debemos llamarla desde fuera por su nombre (en este ejemplo: `mostrarMensaje()`).

Funciones VIII

- Cada vez que se llama una función se ejecutan todas las líneas contenidas en la misma.
- Si no se llama a la función, las instrucciones de la misma nunca se ejecutarán.
- A una función la podemos llamar tantas veces como necesitemos.

Funciones IX

- Las funciones nos ahorran escribir código que se repite con frecuencia y permite que nuestro programa sea más entendible.

- Explicaremos con un ejemplo, una función que tiene datos de entrada.

- Ejemplo: Proponer una función que reciba dos números y muestre en la página los valores comprendidos entre ellos de uno en uno. Leer por teclado estos dos valores.

Funciones X

```
<html>
    <head></head>
    <body>
        <script type = “text/javascript”>
            function mostrarRango(x1,x2)
            {
                var inicio;
                for(inicio=x1; inicio<=x2; inicio++)
                {
                        document.write(inicio+‘ ’);
                }
            }
            var valor1,valor2;
            valor1 = prompt(‘Ingresa el valor inferior:’, ‘’);
            valor1 = parseInt(valor1);
            valor2 = prompt(‘Ingresa el valor superior:’, ‘’);
            valor2 = parseInt(valor2);
            mostrarRango(valor1,valor2);
        </script>
    </body>
</html>
```

Funciones XI

- El programa de JavaScript empieza a ejecutarse donde definimos las variables `valor1` y `valor2` y no donde se define la función.

- Luego de asignar los dos valores por teclado se llama a la función `mostrarRango` y le enviamos las variables `valor1` y `valor2`.

- Los parámetros `x1` y `x2` reciben los contenidos de las variables `valor1` y `valor2`.

Funciones XII

- Es importante notar que a la función la podemos llamar la cantidad de veces que la necesitemos.

- Los nombres de los parámetros, en este caso se llaman `x1` y `x2`, no necesariamente se deben llamar igual que las variables que le pasamos cuando la llamamos a la función, en este caso le pasamos los valores `valor1` y `valor2`.