TEMA 2: INTRODUCCIÓN AL LENGUAJE JAVASCRIPT

Módulo: Desarrollo Web en Entorno Cliente
Profesor: Daniel de la Iglesia Rodrigo (daniel.iglrod.1@educa.jcyl.es)
2º DAW

INDICE

2.1 ¿Dónde estamos?

- *JavaScript* es un lenguaje de programación interpretado. El navegador interpreta el código JavaScript.
- Han existido más lenguajes de este tipo: *VBScript* y *JScript*
  - *JScript* es una versión propia de Microsoft basado en BASIC (intregada en sus navegadores a partir de Internet Explorer 3)
- *JavaScript* y *Jscript* no eran muy diferentes, pero era un problema por temas compatibilidad (cada navegador usaba uno): sentencias condicionales dependiendo de qué navegador se utilizaba
- ECMA lo estandarizó en uno (Estándar ECMAScript) y ambos lenguajes siguen el estándar.

2.2 Características básicas

- Sintaxis de JavaScript se asemeja a C++ y a Java. JavaScript está basado en el concepto objeto ¿orientado a objetos o no?
- Débilmente tipado (diferentes tipos de datos en distintos tiempos de ejecución)
- En 2015 se introdujeron las clases

```
class ClassName {
  constructor() { ... }
  method_1() { ... }
  method_2() { ... }
  method_3() { ... }
}
```

2.2 Características básicas

- ¿Embeber? el código JavaScript en una página HTML
- El navegador “sabe” que el contenido de entre la etiqueta de inicio y fin de <script></script> no es HTML. Este código se procesa antes de mostrarlo por pantalla.
- Existen diferentes lenguajes de script por lo que se recomienda utilizar el atributo “type” que nos permite decirle al navegador cuál es el que se utiliza para codificar el script. Obligatorio según el W3C
- W3C (World Wide Web Consortium) es un consorcio internacional que genera recomendaciones y estándares que aseguran el crecimiento de la World Wide Web a largo plazo (Tim Berners-Lee)

2.3 Sintaxis

- Sensible a mayúsculas y minúsculas
- Comentarios:
  - //comentario
  - /*comentario*/
- Tabulación
- ; al final (optativo)
- Concatenar: +
- Palabras reservadas:
  - break, if, this, while, for, continue, case,
  - var, switch, void, function
  - Estándar ECSMAScript (w3schools)

Versión 1:

```
<script type="text/javascript">var i,j=0;
for (i=0;i<5;i++){ alert("Variable i: "+i);
for (j=0;j<5;j++){ if (i%2==0){
document.write
(i + "-" + j + "<br>");}}}</script>
```

Versión 2:

```
<script type="text/javascript">
  var i,j=0;
  for (i=0;i<5;i++){
    alert("Variable i: "+i;
    for (j=0;j<5;j++){
      if (i%2==0){
        document.write(i + "-" + j + "<br>");
      }
    }
  }
</script>
```

2.4 Tipos de datos

Números

- Sólo existe un tipo numérico.
- Todos los números que se utilizan son formato punto flotante (64 bits). Double en Java o C++
- Permite el uso de números hexadecimales (inician en 0x)

Cadenas de texto

- Se llaman string
- Permite representar secuencias de letras, dígitos, signos de puntuación o cualquier carácter UNICODE.
- Entre comillas dobles o simples
- Secuencias de escape (\)

Valores booleanos

- También conocidos como valores lógicos, admiten true (1) o false (0)

2.5 Variables

- Zonas de memoria de un ordenador que se identifican con un nombre y en las cuales se almacenan datos.
- Operador de asignación: =

Declaración de variables:

- Palabra clave: `var nombreVariable;`

```
var miVariable1;
var miVariable2;
var miVariable1, var miVariable2;
```

Inicialización de variables:

- Asignación directa de un valor

```
var miVariable1 = 30;
```

- Asignación indirecta a través de un cálculo

```
var miVariable2 = miVariable1 + 3;
```

- Asignación a través de solicitud de valor al usuario

```
var miVariable3 = prompt (“Introduce un valor”;
```

2.5 Variables

Regla 1. Las instrucciones en JavaScript terminan en un punto y coma. Ejemplo:

```
var s = "hola";
```

Regla 2. Uso de decimales en JavaScript. Los números en JavaScript que tengan decimales utilizarán el punto como separador de las unidades con la parte decimal. Ejemplos de números:

```
var x = 4;
var pi = 3.14;
```

Regla 3. Los literales se pueden escribir entre comillas dobles o simples. Ejemplo:

```
var s1 = "hola";
var s2 = 'hola';
```

Regla 4. Cuando sea necesario declarar una variable, se utilizará la palabra reservada *var.*
Regla 5. El operador de asignación, al igual que en la mayoría de lenguajes, es el símbolo igual (=).
Regla 6. Se pueden utilizar los siguientes operadores aritméticos: (+ – * /) . Ejemplo:

```
var x = (5*4)/2+1;
```

2.5 Variables

Regla 7. En las expresiones, también se pueden utilizar variables. Ejemplo:

```
var t = 4;
var x = (5*t)/2+1;
var y;
y = x * 2;
```

Regla 8. Comentarios en JavaScript. Existen dos opciones para comentar el código:

a) // cuando se desea comentar el resto de la línea a partir de estas dos barras invertidas.
b) /* y */. todo lo contenido entre ambas etiquetas quedará comentado.

Regla 9. Los identificadores en JavaScript comienzan por una letra o la barra baja (_) o el símbolo del dólar ($).
Regla 10. JavaScript es sensible a las mayúsculas y minúsculas (case-sensitive). Ejemplo:

```
var nombre = "Julio";
var Nombre = "Ramón";
```

2.5 Variables

Let:

- Se introdujo en 2015
- Las variables definidas con let no se pueden volver a declarar.
- Deben declararse antes de su uso
- Tienen alcance de bloque

```
let x = 10;
// x es 10

{
   let x = 2;
   // x es 2
}
// x es 10
```

2.5 Variables

Const:

- Se introdujo en 2015
- Las variables definidas con let no se pueden volver a declarar.
- Tiene alcance de bloque
- No se pueden reasignar

```
const PI = 3.141592653589793;
PI = 3.14;      //error
PI = PI + 10;  //error
```

- Deben ser asignadas

```
const PI = 3.14159265359; const PI;
PI = 3.14159265359;
```

- ¿Cuándo usar const? Regla general: cuando no vaya a cambiar el valor

2.5 Variables

¿Qué información saldrá por pantalla?

```
<script type="text/javascript">
var primer_saludo = "hola";
var segundo_saludo = primer_saludo;
primer_saludo = "hello";
alert(segundo_saludo);
</script>
```

PRÁCTICA 00 – Variables

2.6 Operadores

- Aritméticos, lógicos, de asignación, de comparación, condicionales

OPERADORES ARITMÉTICOS: Suma, resta, multiplicación, división, módulo, incremento y decremento

Operador	Nombre	Descripción
+	Suma	Efectúa la suma entre los operandos.
-	Resta	Efectúa la resta entre los operandos.
*	Multiplicación	Efectúa la multiplicación entre los operandos.
/	División	Efectúa la división entre los operandos.
%	Módulo	Extrae la parte entera del resultado de la división entre los operandos.
++	Incremento	Permite incrementar un valor.
--	Decremento	Permite decrementar un valor.

2.6 Operadores

OPERADORES ARITMÉTICOS

- Cálculos elementales entre variables numéricas
- Típicas: suma, resta, multiplicación y división
- Menos comunes: módulo, incremento y decremento

Operador módulo

- Divide un número entre otro y lo que devuelve es el resto de dicha división.
- Comprobar si un año es bisiesto: año módulo 4 = 0

2.6 Operadores

Operador incremento & decremento

- Incrementar o decrementar en uno el valor de una variable

- Resuelve:

```
x=3;
y=++x;
//¿Cuánto valen x e y?

x=3;
y=x++;
//¿Cuánto valen x e y?
```

2.6 Operadores

Operador incremento & decremento

- Antes de la variable (prefijo): incrementa el valor de la variable y luego lo retorna
- Después de la variable (sufijo): retorna el valor de la variable y luego lo incrementa

- Resolución:

```
x=3;
y=++x;
//En este caso tanto x como y valdrían 4
x=3;
y=x++;
//En este caso x vale 3 e y vale 4
```

2.6 Operadores

OPERADORES LÓGICOS: AND, OR, NOT

Operador	Nombre	Descripción
&&	Y	Ejecuta la operación booleana AND sobre los valores. Devuelve true solo si todos los valores son `true`. Devuelve `false` en caso contrario.
||	O	Ejecuta la operación booleana OR sobre los valores. Devuelve true en el caso en que al menos uno de los valores sea `true`. Devuelve `false` en caso contrario.
!	No	Invierte el valor booleano de su operando.

2.6 Operadores

OPERADORES LÓGICOS

- Combinan y manipulan expresiones lógicas con el fin de evaluar si el resultado de esta combinación es verdadero o falso.
- Se suelen utilizar para tomar decisiones dentro de un programa

INPUT	OUTPUT
A	NOT A
0	1
1	0

INPUT		OUTPUT
A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

INPUT		OUTPUT
A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

PRÁCTICA 01 – Operadores lógicos

2.6 Operadores

OPERADORES DE ASIGNACIÓN: Asignación, suma y asigna, resta y asigna, multiplica y asigna

Operador	Nombre	Descripción
+=	Suma y asigna	Ejecuta una suma y asigna el valor al operando de la izquierda.
-=	Resta y asigna	Ejecuta una resta y asigna el valor al operando de la izquierda.
*=	Multiplica y asigna	Ejecuta una multiplicación y asigna el valor al operando de la izquierda.
/*	Divide y asigna	Ejecuta una división y asigna el valor al operando de la izquierda.
%=	Módulo y asigna	Ejecuta el módulo y asigna el valor al operando de la izquierda.

2.6 Operadores

OPERADORES DE ASIGNACIÓN

- Sirven para ahorrar tiempo y disminuir cantidad de código escrito en programas.
- Su uso no es indispensable

Operador de asignación	Expresión abreviada
a+=b	a = a + b
a-=b	a = a - b
a*=b	a = a * b
a/=b	a = a / b
a%=b	a = a % b

2.6 Operadores

OPERADORES DE COMPARACIÓN: <, <=, ==, >, >=, !=, ===, !==

Operador	Nombre	Descripción
<	Menor que	Verifica si el operando a la izquierda del operador es menor que el operando de la derecha. Devuelve `true` en ese caso o `false` en caso contrario.
<=	Menor o igual que	Verifica si el operando a la izquierda del operador es menor o igual que el operando de la derecha. Devuelve `true` en ese caso o false en caso contrario.
==	Igual	Verifica si los dos operandos son iguales. Devuelve `true` en ese caso o `false` en caso contrario.
>	Mayor que	Verifica si el operando a la izquierda del operador es mayor que el operando de la derecha. Devuelve `true` en ese caso o false en caso contrario.
>=	Mayor o igual que	Verifica si el operando a la izquierda del operador es mayor o igual que el operando de la derecha. Devuelve `true` en ese caso o `false` en caso contrario.
!=	Diferente	Verifica si los dos operandos son diferentes. Devuelve `true` en ese caso o `false` en caso contrario.
===	Estrictamente igual	Verifica si el operando a la izquierda del operador es igual y del mismo tipo de datos que el operando de la derecha. Devuelve `true` en ese caso o `false` en caso contrario.
!==	Estrictamente diferente	Verifica si el operando a la izquierda del operador es diferente y/o de tipo diferente que el operando de la derecha. Devuelve `true` en ese caso o `false` en caso contrario.

2.6 Operadores

OPERADORES DE COMPARACIÓN

- Nos permiten comparar todo tipo de variables con el fin de verificar valores.
- El resultado de una comparación siempre es un valor booleano.
- Se puede aplicar a números y a cadenas
- Si se utilizan tipos de datos distintos para la comparación, se convertirán automáticamente para que la comparación se realice.

2.6 Operadores

OPERADORES CONDICIONALES: ?

Operador	Nombre	Descripción
?:	Condicional	Si la expresión antes del operador es verdadera, se utiliza el primer valor a la derecha. En caso contrario se utiliza el segundo valor a la derecha.

- Se trata de un operador condicional que nos permite ejecutar una instrucción después de evaluar una expresión.
- Consta de tres partes: 1º la expresión a evaluar, la 2º es la acción a realizar si la expresión es verdadera y la 3º es la acción a realizar si la expresión es falsa.

PRÁCTICA 02 – Operadores

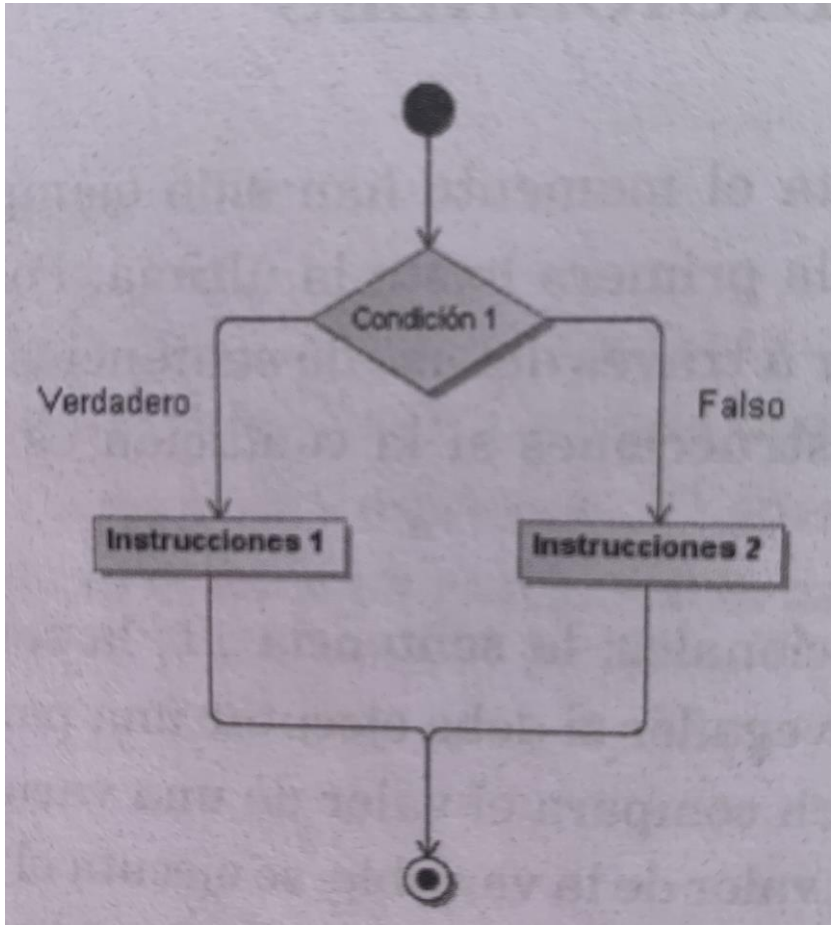
2.7 Sentencias condicionales

Existen cuatro tipos de sentencias condicionales: IF, SWITCH, WHILE y FOR

IF

```
if (expresión){
instrucciones
}
```

- Devuelve valores true/false
- Instrucciones se ejecuta si es true la expresión.
- Si es false, JavaScript ignora instrucciones
- Uso de llaves no obligatorio en caso de ejecutar 1 instrucción

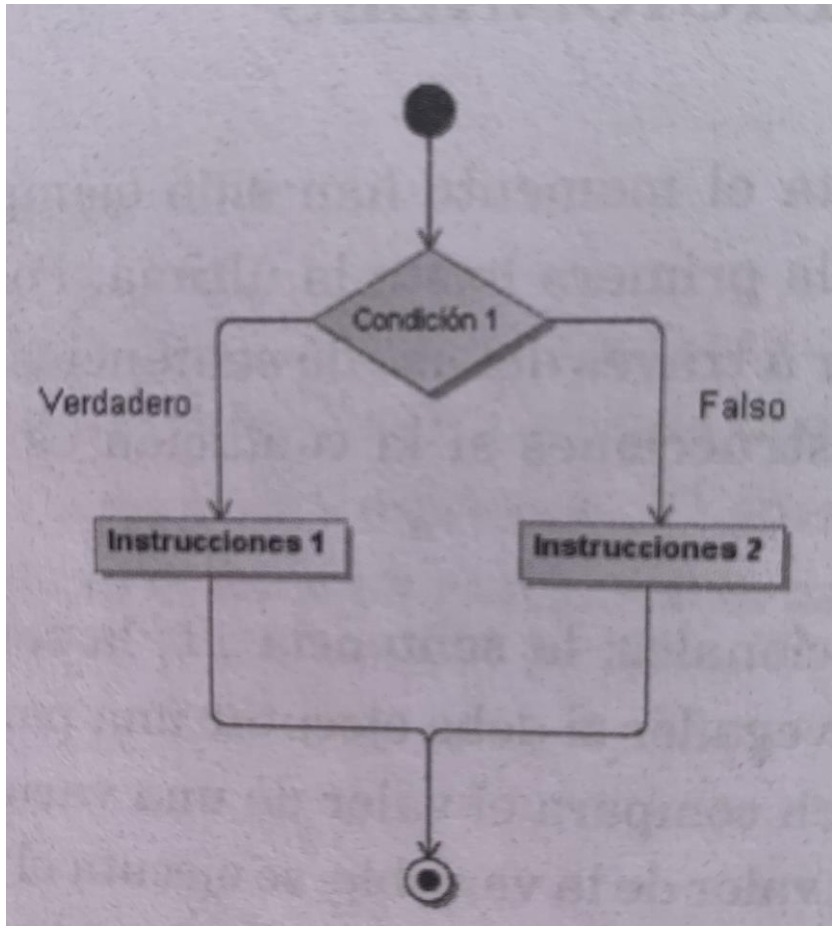


2.7 Sentencias condicionales - IF

Se puede añadir a la sentencia IF, la palabra clave ELSE

IF-ELSE

```
if (expresión){
  instrucciones_si_true
}else{
  instrucciones_si_false
}
```



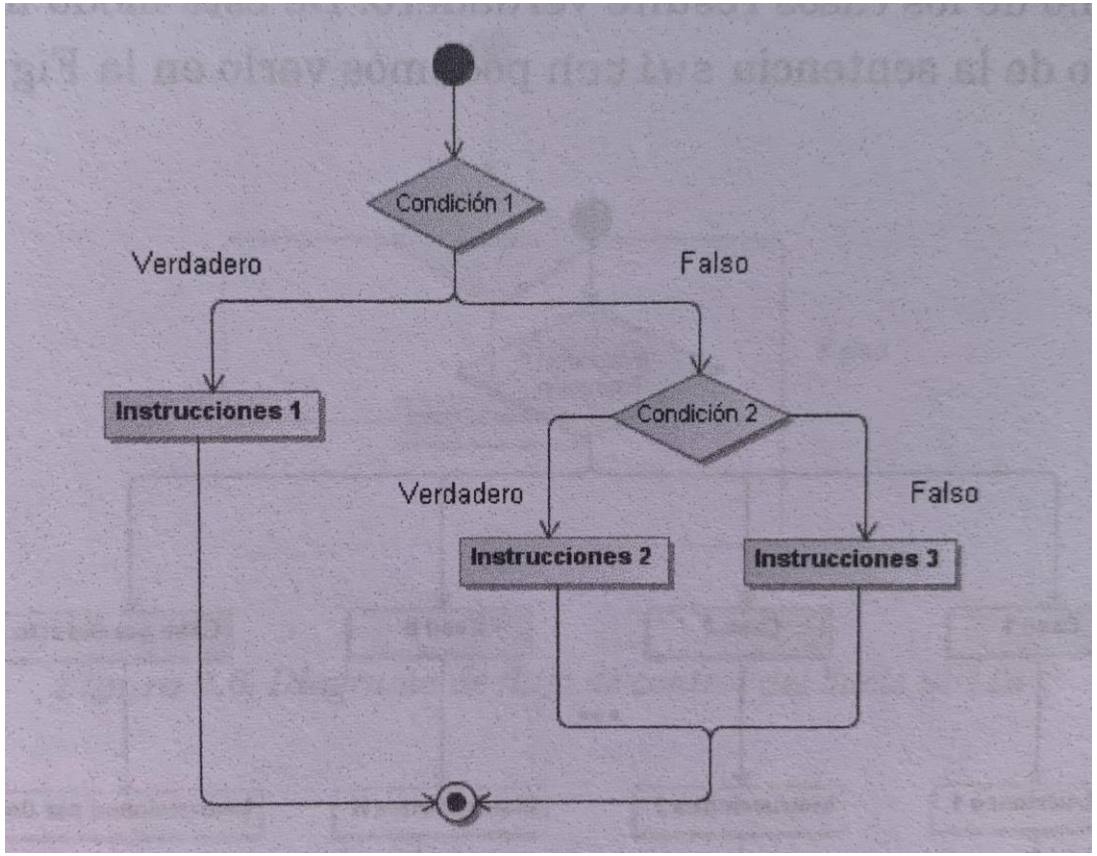
2.7 Sentencias condicionales - IF

Se pueden anidar sentencias IF cuando la primera expresión es false

IF-ELSE-IF

```
if (expresión1){
  instrucciones_1
}else if (expresión2){
  instrucciones_2
}else{
  instrucciones_3
}
```

No se recomienda su uso cuando hay que utilizar muchas sentencias IF.



PRÁCTICA 03 – Sentencias condicionales IF

2.7 Sentencias condicionales - SWITCH

SWITCH

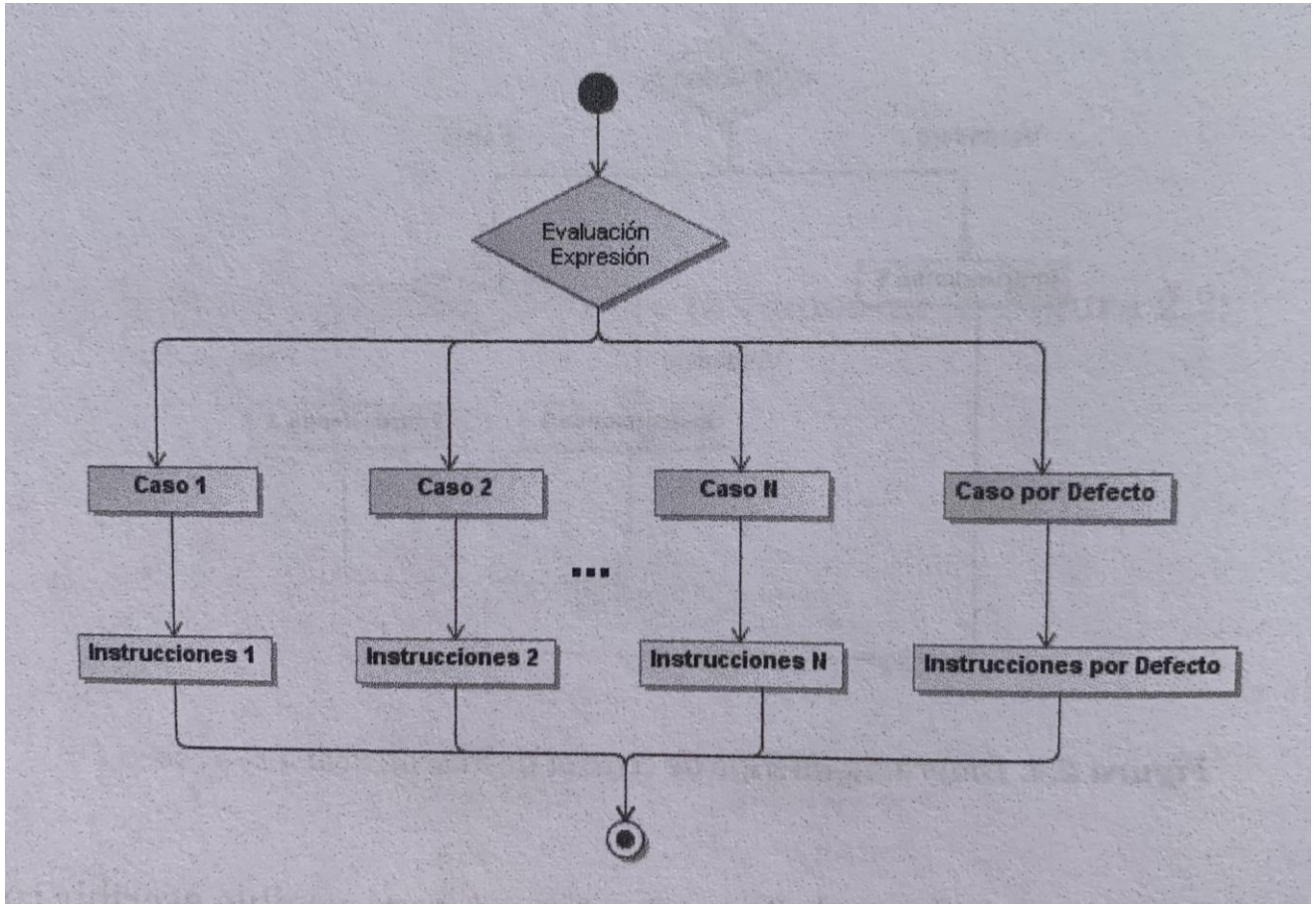
- Sirve para comparar el valor de una variable con algunos valores ya conocidos.
- Se utiliza una expresión a evaluar y una serie de posibles valores llamados casos.
- En cada caso hay instrucciones diferentes a ejecutar
- Después de cada bloque de instrucciones se utiliza `break;` para detener la ejecución en el caso de que uno de los casos resulte verdadero
- Las instrucciones que se encuentran dentro de `default`, se ejecutarán solo cuando ninguno de los valores de cada caso coincida

2.7 Sentencias condicionales - SWITCH

SWITCH

```
switch (expresión){
 case valor1:
  instrucciones a ejecutar si expresión = valor1
  break;
 case valor2:
  instrucciones a ejecutar si expresión = valor 2
  break;
 default:
   instrucciones a ejecutar si expresión es diferente a los
valores  anteriores
}
```

2.7 Sentencias condicionales - SWITCH



PRÁCTICA 05 – Sentencias condicionales SWITCH

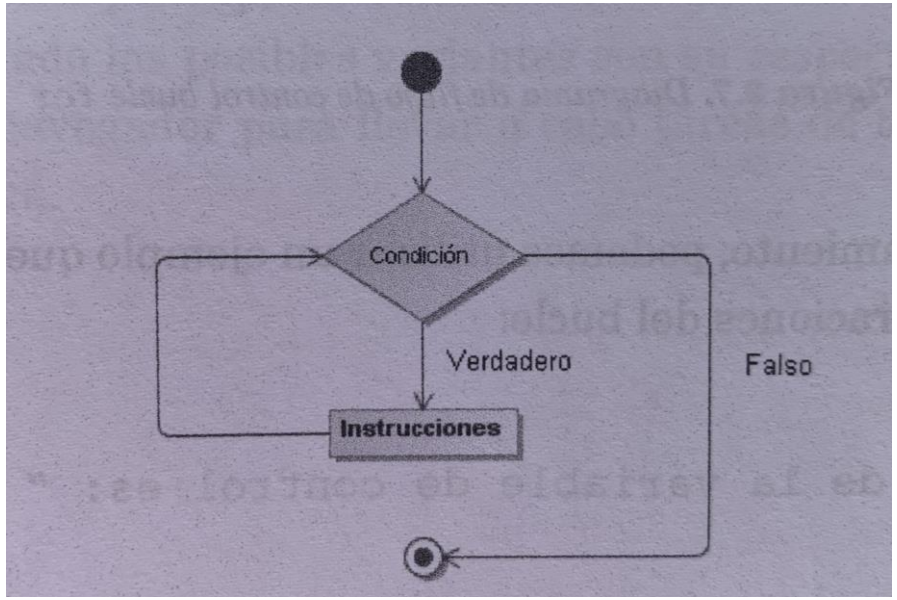
2.7 Sentencias condicionales - WHILE

WHILE

- Es el más sencillo en JavaScript, el navegador ejecutará una o más instrucciones continuamente hasta que una condición deje de ser verdadera.
- Se utiliza cuando no sabemos el número exacto de veces que algo se repite

```
while (expresión){
instrucciones
}

do{
instrucciones
} while (expresión)
```



2.7 Sentencias condicionales - WHILE

DO-WHILE

```
do{
instrucciones
} while (expresión)
```

Es una variable del bucle WHILE y la diferencia es que en el DO-WHILE la ejecución del bucle se lleva a cabo al menos una vez y se evalúa al final si se debe seguir ejecutando o no.

PRÁCTICA 06 – Sentencias condicionales WHILE

2.7 Sentencias condicionales - FOR

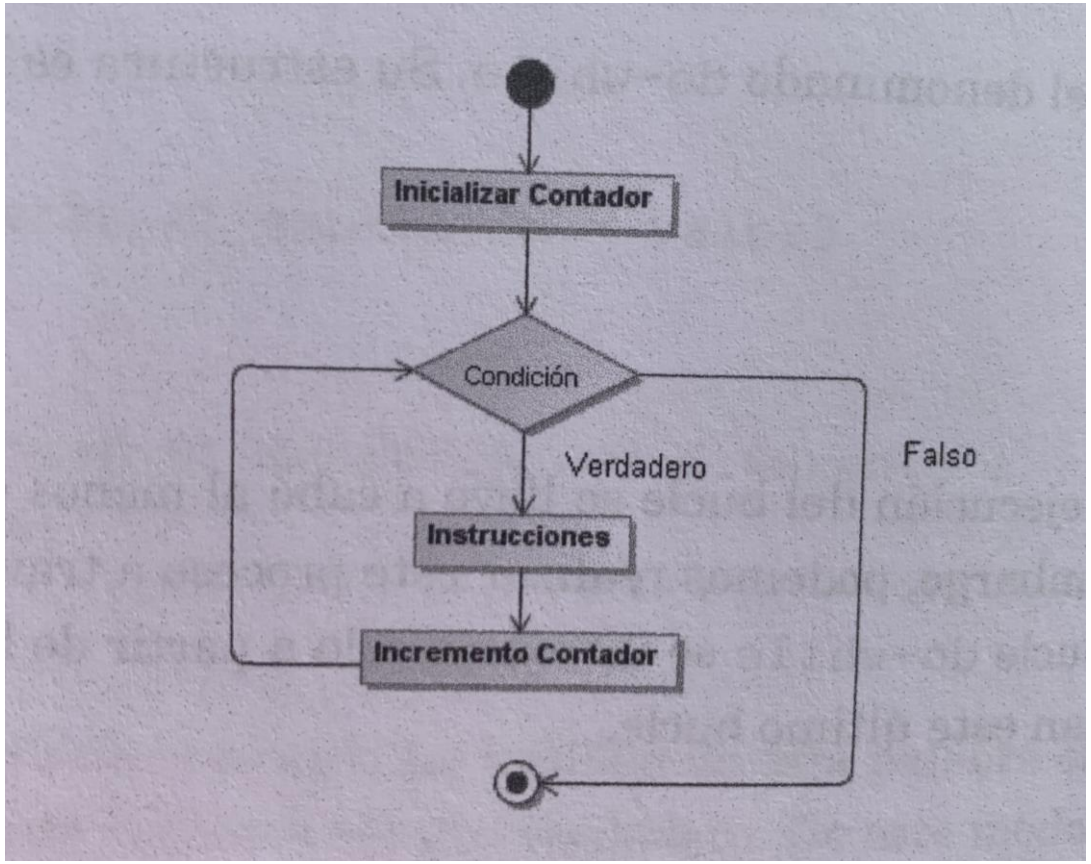
FOR

- Permite ejecutar las instrucciones que se encuentran dentro del mismo bucle hasta que la expresión condicional devuelva false.
- Consta de 5 partes: palabra clave `for`, valor inicial de la variable de control, condición basada en dicha variable, incremento/decremento y el cuerpo del bucle

```
for (valor_inicial;expresión_condicional;incremento/decremento){
  cuerpo_bucle
}
```

2.7 Sentencias condicionales - FOR

FOR



PRÁCTICA 06 – Sentencias condicionales FOR

2.7 Arrays

- Facilitan la gestión de datos para que la programación no sea tan repetitiva
- Un array es un conjunto de valores relacionados
- Cada array está formado por elementos que tiene un índice
- Su índice indica la posición en el array (empieza en 0)
- Todos los arrays deben de tener un nombre
- En JavaScript es un objeto más
- Recorrer arrays con bucle `for (for i=0; i<num;i++) //siendo i el índice del array`

Declaración arrays con objeto Array: Tiene 6 partes

```
var nombreArray operadorAsignacion palabraClaveNew constructor Array paréntesis final
 var nombreArray = new Array();
```

- Se crea una instancia del objeto Array.
- El constructor Array() tiene la función de construir el objeto

```
var nombreArray = new Array (10) -> Array de 10 elementos
```

Declaración arrays sin objeto Array:

```
var nombreArray = [];
```

Iniciacilización arrays

Opción 1:

```
var productos = new Array(); -> array sin límite
productos[0] = “pan” // productos[1] =“brocoli” // productos[2]= “jamón”
```

Opción 2:

```
var productos = new Array (“pan”, “brócoli”, “jamón”); //o con comillas simples
```

Opción 3:

```
var productos = [];
```

productos[0] = “pan” // productos[1] =“brocoli” // productos[2]= “jamón”

Opción 4:

```
Var productos = [“pan”, “brócoli”, “jamón”];
```

Propiedades

*En JavaScript un array es un objeto

- **length:** Devuelve el número de elementos que contiene el array -> `productos.length`

Métodos

- **push():** Añade nuevos elementos al array al final cambiando su length

```
var paises = [”España", ”Francia", ”Colombia", ”Brasil"];
paises.push (“Italia”);
```

- concat(): Concatena arrays en uno solo

```
var paises1 = [”España", ”Francia", ”Colombia", ”Brasil"];
var países2 = [”Rusia", ”Japón”];
var paises = paises1.concat(paises2);
```

- join(): Devuelve un array como string separado por comas

```
var animales = [”León", ”Gato", ”Rinoceronte", ”Perro"];
alert (animales.join()); -> León, Gato, Rinoceronte, Perro
alert (animales.join( “ y ”)); Se puede establecer el separador pasándoselo
```

- reverse(): Invierte el orden de los elementos de un array

```
var frutas = ["Banana", "Orange", "Apple", "Mango"];
frutas.reverse();
```

- unshift(): Agrega nuevos elementos al inicio del array

```
var frutas = ["Banana", "Orange", "Apple", "Mango"];
frutas.unshift("Lemon", "Pineapple");
```

- shift(): Elimina el primer elemento de un array

```
var animales = [”León", ”Gato", ”Rinoceronte", ”Perro"];
animales.shift();
```

- pop(): Elimina el último elemento de un array

```
var animales = [”León", ”Gato", ”Rinoceronte", ”Perro"];
animales.pop();
```

- slice(): Devuelve los elementos seleccionados en una array, como un nuevo objeto del array.

```
var animales = [”León", ”Gato", ”Rinoceronte", ”Perro"];
animales.slice(1,3); -> El 1 lo incluye, el 3 no. Esto sacaría Gato y Rinoceronte
```

- sort(): Ordena los elementos de un array (orden alfabético y ascendente). No sirve para ordenar números porque los considera string y no ordena correctamente. 45 es mayor que 1000 (4 > 1)

```
var animales = [”León", ”Gato", ”Rinoceronte", ”Perro"];
animales.sort();
```

Arrays multidimensionales

En JavaScript es posible crear arrays de más dimensiones, pero se define un array que contiene a la vez nuevos arrays en cada una de sus posiciones.

```
//Array de 10 elementos de animales salvajes
var salvajes = new Array (10);
salvajes[0] = “León” //faltan elementos

//Array de 5 elementos de animales domésticos
var domesticos = new Array (5);
domesticos[0] = “Gato”; //faltan elementos

//Array de animales que incluye a los salvajes y a los domésticos
var animales= new Array (2);
animales[0] = salvajes;
animales[1] = domesticos;
```

Recorrer estructura con FOR

```
for (i=0; i<animales.length;i++){
    //código
    for (j=0; j<animales[i].length;j++){
        //código
    }
}
```

Práctica 07 - Arrays

Práctica 08 – Global