# UNIDAD 2 - ELEMENTOS BÁSICOS

## 2.1 - ELEMENTOS DEL LENGUAJE

Un programa está formado por los elementos siguientes:

**Constantes:** Representan entidades que no se modifican durante la ejecución del programa. Pueden tener un identificador o carecer del mismo.

**Variables:** Posiciones de memoria a las que se asigna un nombre y a las que se puede acceder para obtener el valor que almacenan.

**Operadores:** Símbolos que representan una operación de naturaleza matemática, relacional o lógica.

**Expresiones** Son combinaciones de variables, constantes y operadores. La evaluación de una expresión devuelve un valor.

**Instrucciones o sentencias:** Permiten organizar el flujo de ejecución de un programa estableciendo el orden en el que se encadenan las acciones del mismo.

**Funciones:** Agrupaciones de instrucciones diseñadas para resolver un problema concreto.

El lenguaje presenta algunas normas o reglas que deben seguirse:

• Distingue mayúsculas de minúsculas, de forma que no es lo mismo definir una variable con identificador suma que Suma.

• Los comentarios son iguales que en C/C++: /* ... */ para encerrar un bloque y // para comentarios de una línea.

• Cada sentencia ha de terminar en punto y coma (;). (Aunque no es estrictamente obligatorio)

• Se pueden agrupar sentencias utilizando llaves {sentencia1; sentencia2; ... }

## 2.2 - CONSTANTES

Las constantes representan valores que no se modifican durante la ejecución del programa. Son constantes todos los valores que aparecen directamente en un programa, por ejemplo 3.1415 es una constante que representa el número PI. "Aula Mentor" es otra constante que representa una cadena de caracteres. En Javascript se pueden definir constantes que incorporen un identificador de manera que se asigna un valor a dicho identificador y en el programa se hace referencia a la constante no por su valor sino por el identificador que la representa. La ventaja es que si el programador decide modificar el programa y cambiar el valor de la constante, no tiene que buscar en todo el programa el lugar en el que aparece, limitándose solamente a hacerlo en el lugar en el que ha sido definida la misma. Las constantes con identificador se declaran con la cláusula var que también se usa para las variables. Seguido al identificador tenemos el símbolo = y el valor constante.

```
var tipo_iva=18;
```

Algunos navegadores entienden la cláusula **const**, que se usa en otros lenguajes, pero no es compatible con todos, por lo que no conviene utilizarla.

## 2.3 - VARIABLES

Las variables son zonas de la memoria del ordenador identificadas por un nombre y capaces de almacenar un valor. La variables deben su nombre a la capacidad para almacenar valores que pueden cambiar durante la evolución del programa en contraposición a las constantes cuyo valor tal y como señala su nombre no puede cambiar. Las variables proporcionan una gran flexibilidad a los programas ya que

permiten que estos puedan resolver problemas con cierta generalidad. Por ejemplo, se podría hacer un programa para resolver ecuaciones de segundo grado con constantes. Pero dicho programa sólo podría resolver siempre la misma ecuación. Sin embargo al dotar de variables al programa, este puede utilizarse para resolver cualquier ecuación de segundo grado.

Las variables son elementos fundamentales en cualquier lenguaje de programación. Sirven para guardar y manipular información. Podemos representar una variable como un recipiente en el que se puede depositar información que puede ser consultada o cambiada en cualquier momento de la ejecución del programa. En realidad una variable consiste en un conjunto de posiciones de memoria reservadas que a las que damos un nombre llamado **identificador** de la variable que nos permitirá manipular la información.

Las variables se crean normalmente al comienzo del programa utilizando la palabra **var** seguida del nombre de la variable que queremos crear, como en los siguientes ejemplos:

```
var nombre;
var edad;
```

Se pueden declarar dos o más variables en una instrucción. En estos casos se indicará la palabra reservada var y, a continuación, la lista de las variables que queremos declarar (los nombres separados por comas):

```
var nombre, edad;
```

El nombre o identificador de la variable podrá estar formado por letras (mayúsculas y minúsculas), caracteres de subrayado (_), el símbolo de dólar ($) , los números (del 0 al 9). Pero no podrá comenzar por un número y tampoco podrá coincidir el nombre con alguna de las palabras reservadas del lenguaje (nombres de

**NOTA**

Como JavaScript diferencia mayúsculas y minúsculas en los identificadores es muy fácil confundirse. Por este motivo es necesario seguir dos normas:

**1º-**Declarar todos los identificadores al comienzo del programa para poder comprobar rápidamente cualquier duda.

**2º-** Establecer un criterio a la hora de construir identificadores y seguirlo siempre (Determinar cuándo se utilizan mayúsculas, minúsculas y sub-guiones, etc.). A la hora de establecer este criterio deberemos valorar también la legibilidad del programa.

comandos, etcétera). Son por tanto identificadores válidos: ***vApellido1; $1Apellido; _Apellido*** y no válidos: ***1Apellido; #1Apellido; Apellido1º;***

### 2.3.1 -ALMACENAMIENTO DE INFORMACIÓN EN LA VARIABLE.

Para guardar información en una variable se utilizará el operador de asignación (**=**) precedido por el nombre de la variable y seguido por el valor que queremos asignar. Ejemplos:

```
Nombre = 'Miguel';
Edad = 17;
```

**Atención:** cuando guardamos información en una variable, cualquier otro valor que dicha variable tuviese anteriormente se perderá (salvo que lo guardemos en otra variable antes de hacer la asignación). JavaScript permite asignar un valor a una variable en el mismo momento de su declaración. Ejemplo:

```
var Nombre = 'Miguel';
```

También permite asignar valor a una variable que no ha sido previamente declarada.

```
NuevoNombre = 'Alfonso';
```

En el ejemplo anterior ***NuevoNombre*** no ha sido declarada previamente, sin embargo, al encontrar esta expresión JavaScript creará implícitamente la nueva variable y le asignará el valor indicado. Esta práctica es, de todo punto de vista, desaconsejable; de hecho, la mayoría de los lenguajes de programación la consideran ilegal. Debemos acostumbrarnos a declarar todas las variables que vayamos a utilizar al comienzo del programa.

### 2.3.2 -CONSULTA O UTILIZACIÓN DEL VALOR CONTENIDO EN LA VARIABLE.

Una vez declarada una variable podemos hacer referencia a su contenido para visualizarlo, o para emplearlo con otras expresiones.

```
<HTML>
<HEAD><TITLE>Manejando variables</TITLE></HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
        /* Ejemplo creación y manejo de variables */
        var Num1 = 5;
        var Num2 = 7;
        var Suma;
        Suma = Num1+Num2;
        alert(Suma);
</SCRIPT>
</BODY>
</HTML>
```

***Ejemplo 7 – Manejo de variables***

Este programa crea tres variables (***Num1, Num2 y Suma***), inicializando las dos primeras en el momento de su creación y asignado el valor de la suma de estas a la variable ***Suma***. Finalmente muestra una ventana de alerta con el valor de la variable ***Suma***.

**Nota:** JavaScript elimina automáticamente las variables al finalizar el programa en el que se han declarado por lo que su valor se pierde en futuras ejecuciones del mismo.

## 2.4 - TIPOS DE VARIABLES

Todas las variables en JavaScript se crean de la misma forma, utilizando tal y como se ha señalado la palabra reservada **var**. Sin embargo y a diferencia de otros lenguajes de programación, el tipo de las variables es dinámico, de manera que el programador puede utilizar la misma variable para almacenar diferentes tipos de datos. Dicho de otra manera, las variables adaptan su tipo al contenido que almacenan. Aunque no es una práctica recomendable es aprovechada en algunos casos para hacer que los programas sean más sencillos pero en otras ocasiones provoca tener que añadir código para detectar el tipo de dato que tiene la variable en un instante determinado.

### 2.4.1 -NUMÉRICAS

Se utilizan para almacenar valores numéricos enteros (llamados ***integer*** en inglés) o reales (llamados ***float*** en inglés). En este caso, el valor se asigna indicando directamente el número entero o real. Los números reales utilizan el carácter **.** (punto) en vez de **,** (coma) para separar la parte entera y la parte decimal:

```
var iva = 18; // variable entera
var euro = 166.38665; // variable real
```

Un valor numérico se suele representar mediante una serie de dígitos que pueden ir precedidos de un signo (+ ó -) y que pueden llevar un punto decimal. A continuación podemos ver algunas expresiones numéricas válidas: ***45 389.347 -18 +6789.89 -8768.3243***

JavaScript también admite la notación exponencial para representar valores numéricos en formato de coma flotante: ***56.987E+12 23634E-6 -484.2382E-20***

Se pueden representar números en base distinta a la decimal, por ejemplo en hexadecimal, o en octal. Para expresar un número hexadecimal antepondremos el prefijo 0x (cero equis); y para expresar un número octal antepondremos un 0 (cero). ***0xF34*** (en hexadecimal) ***0567*** (en octal)

```
<HTML>
<HEAD><TITLE>Representaciones en base 8 y 16</TITLE></HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
        alert ("Hexadecimal:"+ 0xF34);
        alert ("Octal:"+ 0567);
// Fin del programa.
</SCRIPT>
</BODY>
</HTML>
```

***Ejemplo 8 – Representación en base 8 y 16***

Observe que al ejecutar el ejemplo anterior, JavaScript transforma el ***F34*** (hexadecimal pues va precedido por 0x) a su correspondiente valor decimal ***3892***. También ha trasformado el ***567*** (octal pues va precedido por 0) al valor correspondiente en decimal: ***375***.

## 2.4.2 -CADENAS DE CARACTERES

Permiten almacenar caracteres que constituyen texto. Para asignar el valor a la variable es necesario encerrar el valor entre comillas dobles o simples:

```
var mensaje = "Hola mundo ";
var nombre = 'Aula Mentor';
var letra = 'b';
```

El motivo de que se puedan utilizar comillas dobles o simples indistintamente es para poder utilizar comillas dentro del propio texto que se desea almacenar. En este caso es necesario encerrar con un tipo de comillas diferente al que se desea almacenar.

```
/* El contenido de texto1 tiene comillas simples, por lo que
se encierra con comillas dobles */
var texto1 = "Una frase con 'comillas simples' dentro";
/* El contenido de texto2 tiene comillas dobles, por lo que
se encierra con comillas simples */
var texto2 = 'Una frase con "comillas dobles" dentro';
```

Pero si el texto que se desea almacenar incluye ambos tipos de comillas así como otros caracteres especiales es necesario utilizar las secuencias de escape que no son más que símbolos precedidos de la barra inclinada **** y cuyo significado es:

| Elemento que se desea incluir | Secuencia de Escape |
|---|---|
| Una nueva línea | `\n` |
| Un tabulador | `\t` |
| Una comilla simple | `\'` |
| Una comilla doble | `\"` |
| Una barra inclinada | `\\` |

```
var texto1 = 'Una frase con \'comillas simples\' dentro';
var texto2 = "Una frase con \"comillas dobles\" dentro";
```

### 2.4.3 -TIPO LÓGICO

Las variables de tipo lógico o booleano son aquellas que permiten representar valores de verdad: true (verdadero) o false (falso).

```
var PagoRealizado = false;
var ivaIncluido = true;
```

### 2.4.4 -TIPOS ESTRUCTURADOS: ARRAYS

Mientras que tal y como se ha observado, los tipos simples son capaces de almacenar un único valor simultáneamente en una variable, los tipos estructurados o estructuras de datos permiten almacenar múltiples valores. Tradicionalmente los lenguajes de programación disponen de numerosos tipos estructurados. Entre los más habituales destacan los Arrays más conocidos en el ámbito matemático por matrices o incluso vectores cuando se limitan a una dimensión. Un array es un conjunto de variables del mismo tipo que tienen un nombre único y se diferencian unas de otras por la posición que ocupan.

El objetivo de los arrays es simplificar el código y establecer una relación entre un conjunto de variables que representan realidades similares. Por ejemplo, si se desea trabajar con los meses del año en un programa se podrían definir doce variables simples y por tanto independientes cada una con su propio nombre y que almacenasen valores diferentes:

```
var mes_uno = "Enero";
var mes_dos = "Febrero";
...
var mes_doce = "Diciembre";
```

Esta estrategia obliga a crear una variable por cada mes y dado que cada una es independiente su nombre debe de ser distinto. En capítulos posteriores, cuando se estudien las estructuras de control, se comprobará que utilizar arrays en vez de variables simples, reduce mucho la complejidad del código y abre un abanico ingente de posibilidades a la hora de resolver problemas que impliquen información relacionada. En este tipo de casos, se pueden agrupar todas las variables relacionadas en una colección

de variables o array. El ejemplo anterior se puede plantear con una array de una dimensión (vector) de la siguiente forma:

```
var meses = ["Enero", "Febrero", "Marzo", "Abril", "Mayo", "Junio", "Julio",
"Agosto","Septiembre","Octubre","Noviembre","Diciembre"];
```

Desde el punto de vista sintáctico, la definición de un array es muy sencilla ya que simplemente hay que utilizar se utilizan los caracteres [ y ] para delimitar su comienzo y su final y se utiliza el carácter , (coma) para separar sus elementos:

```
var nombre_array = [valor1, valor2, ..., valorN];
```

Para acceder a los diferentes elementos de un array simplemente hay que utilizar el nombre de la variable y el índice que representa la posición que ocupa el elemento al que se desea acceder dentro del array. Hay que tener en cuenta que las posiciones comienzan a numerarse en cero, por lo que el primer elemento es el mes[0] y así sucesivamente mes[1], mes[2], etc.

### 2.4.5 -TIPOS ESTRUCTURADOS: OBJETOS

Los objetos son los principales protagonistas del paradigma de la programación orientada a objetos. Son una evolución de otro tipo estructurado denominado registro (estructuras en algunos lenguajes). Los objetos representan entidades cuya complejidad hace que no sea suficiente un valor para almacenar toda la información necesaria. Por este motivo, los registros o estructuras son conjuntos de datos de diferente tipo relacionados entre sí. Por ejemplo, los datos de una persona (nombre, apellidos, fecha de nacimiento, dirección, etc.) se pueden agrupar en un registro o estructura. El paradigma orientado a objetos incorpora a esos datos estructurados, las funciones (llamadas métodos) que manipulan esos datos, haciendo que los datos y los procedimientos conformen un ente llamado objeto. En JavaScript hay numerosos objetos predefinidos y el usuario-programador puede definir los que necesite. Ejemplo de objetos predefinidos en Javascript son las ventanas, los formularios, los controles, etc.

### 2.4.6 -VALORES ESPECIALES

En JavaScript existen dos valores especiales que son el valor ***null*** que indica que el contenido de una variable es nulo y el valor ***undefined*** que indica que la variable no tiene valor. El valor ***null***, por ejemplo, es el valor que devuelve la función ***prompt*** cuando el usuario pulsa el botón Cancelar. El valor ***undefined***, por ejemplo, es el valor que aparece en el control de introducción de datos cuando no se establece valor por defecto en la función prompt.

## 2.5 - OPERADORES

Los operadores son el primer eslabón a la hora de establecer la capacidad de cálculo de un lenguaje de programación. Con ellos es posible realizar operaciones de diversa naturaleza que incluyen como mínimo cálculos aritméticos, relacionales y lógicos. La combinación de operadores con los datos a tratar conformar las expresiones.

Las expresiones son combinaciones de operadores y operandos. Los operandos pueden ser constantes, variables y llamadas a funciones. Todas las expresiones producen o devuelven un resultado de un cierto tipo que estará determinado por el tipo de operandos que manipula y por los operadores que utiliza. Así mismo la operación asociada a un operador puede ser diferente en función de los operandos que maneje. Esto recibe el nombre de sobrecarga de operadores y se manifiesta de forma muy clarificadora cuando se analiza el operado +. Si este operador se aplica a dos operandos numéricos, el resultado será la suma aritmética. Sin embargo, si se aplica a operandos de tipo cadena de caracteres, el resultado será la concatenación (yuxtaposición) de ambas cadenas.

| Tipo de Expresión | Tipo de retorno | Expresión | Resultado |
|---|---|---|---|
| **Expresión de cadena** | **Cadena** | **“Hola” + “Mundo”** | **“HolaMundo”** |
| **Expresión de cadena** | **Cadena** | **10 + 20** | **“1020”** |
| **Expresión aritmética** | **Numérico** | **10 + 20** | **30** |
| **Expresiones relacionales** | **Lógico** | **7 > 5** | **True** |
| **Expresiones lógicas** | **Lógico** | **True && False** | **False** |

### 2.5.1 -OPERADOR DE CONCATENACIÓN.

Para unir cadenas de caracteres se emplea el operador de concatenación (+). Por ejemplo la expresión "Buenos" + "Días" dará como resultado "BuenosDias".

### 2.5.2 -OPERADORES ARITMÉTICOS.

Se utilizan con datos de tipo numérico y devuelven siempre un valor numérico. Son los conocidos: **suma** (+), **resta** (-), **multiplicación** (*), **división** (/) y **módulo** o resto de la división entera (%).

```
<HTML>
<HEAD><TITLE>Operadores aritméticos</TITLE></HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
/* Ejemplo operadores aritméticos*/
var Num1 = 2, Num2 = 4, Num3 = 6;
alert(
" Num1 \t" + (Num1)
+"\nNum2 \t" + (Num2)
+"\nNum3 \t" + (Num3)
+"\nNum1 + Num2 \t" + (Num1 + Num2)
+"\nNum3 / Num2 \t" + (Num3 / Num2)
+"\nNum3 % Num2 \t" + (Num3 % Num2)
+"\nNum1 + 2 \t" + (Num1 + 2)
+"\nNum1 \t" + (Num1)
);
</SCRIPT>
</BODY>
</HTML>
```

***Ejemplo 9 – Operadores aritméticos***

### 2.5.3 -OPERADORES DE INCREMENTO Y DECREMENTO

Un caso particular dentro de los operadores aritméticos son los operadores de incremento y decremento por el hecho de que su uso no solo devuelve un valor sino que modifica el propio contenido de la variable. En realidad estos operadores combinan un operador aritmético con el operador de asignación que aparece explicado en el próximo epígrafe. Se utilizan con variables numéricas y permiten incrementar o decrementar en

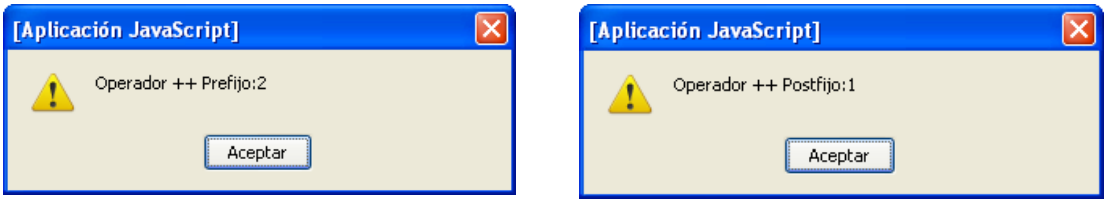
una unidad el valor de la variable sobre la que se aplica. Se trata de operadores unarios y el operando sobre el que se aplican siempre debe ser una variable ya que no tiene sentido intentar incrementar una constante porque no se puede modificar su valor. Los operados son el incremento (*++*) y el decremento (--).

```
var numero = 9;
++numero;
alert(numero); // numero = 10
```

Ambos operandos tienen un efecto diferente si se utilizan de forma prefija (delante del operando) que de forma postfija (detrás del operando). En el primer caso incrementan/decrementan el valor de la variable y la expresión devuelve como resultado el nuevo valor (ya incrementado/decrementado). El caso postfijo primero devuelve el valor antes de incrementar/decrementar y posteriormente incrementa/decrementa el valor de la variable. Evidentemente esta sutil pero importante diferencia sólo se observa cuando el operador aparece formando parte de otra expresión más compleja. Por ejemplo se puede observar en el ejemplo siguiente asociada a una función ***alert***.

```
<HTML>
<HEAD><TITLE>Diferencias en notación prefija y postfija</TITLE></HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
        var pruebaA = 1;
        var pruebaB = 1;
        alert ("Operador ++ Prefijo:" + (++pruebaA));
        alert ("Operador ++ Postfijo:"+ (pruebaB++));
        // Fin del programa.
</SCRIPT>
</BODY>
</HTML>
```

***Ejemplo 10 – Diferencias notación prefija y postfija***




**Notación prefija (++pruebaA)**

**Notación postfija (pruebaB++)**

***Ilustración 3 – Resultado diferencias entre notación prefija y postfija***

## 2.5.4 -OPERADOR DE ASIGNACIÓN

El operador de asignación es uno de los operadores más importantes en los lenguajes de programación ya que permite asignar (almacenar) en una variable, el resultado de una expresión. El símbolo que representa el operador de asignación (=) el cual no debe confundirse con el operador relacional de comparación (==)

```
var numero = 9;
```

## 2.5.5 -OPERADORES ARIMÉTICOS CON ASIGNACIÓN

Dado que es muy frecuente realizar operaciones aritméticas que impliquen que una variable sea operando de la expresión y variable destino del resultado, los lenguajes de programación incorporan operadores que simplifican la escritura de expresiones que suelen ser habituales en los programas. Estos operadores combinan dos símbolos que son el correspondiente a la operación deseada y el operador de asignación. Para el caso de la suma sería (+=)

```
<HTML>
<HEAD><TITLE>Operadores aritméticos con asignación</TITLE></HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
/* Ejemplo operadores asignación*/
var Suma = 6, Resta = 6, Multi = 6, Divi = 6, Modu = 6;
Suma += 2;
Resta -= 2;
Multi *= 2;
Divi /= 2;
Modu %= 5;
alert(
" Suma \t" + (Suma)
+"\nResta \t" + (Resta)
+"\nMulti \t" + (Multi)
+"\nDivi \t" + (Divi)
+"\nModu \t" + (Modu)
);
</SCRIPT>
</BODY>
</HTML>
```

***Ejemplo 11 – Operadores aritméticos con asignación***

El programa anterior muestra el funcionamiento de los operadores de asignación ***+=, -=, *=, /= y %=*** utilizando cinco variables llamadas ***Suma, Resta, Multi, Divi y Modu. Iniciacilizadas*** todas ellas a 6 *y* realizando las correspondientes operaciones de asignación. Finalmente se muestra con una ventana de alerta los nombres de las variables y su valor tras la operación.

## 2.5.6 -OPERADORES RELACIONALES

Los operadores relacionales permiten comparar dos operandos siempre que entre ambos esté definida una relación de orden. En JavaScript están definidos los operadores: mayor que (**>**), menor que (**<**), mayor o igual (**>=**), menor o igual (**<=**), igual que (**==**) y distinto de (***!=***). El resultado de aplicar un operador relación da lugar a una expresión lógica por lo que el resultado siempre será true o false.

```
<HTML>
<HEAD><TITLE>Operador relacional de igualdad</TITLE></HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
        var A = 10;
        var B = 12;
        alert ( A==B);
</SCRIPT>
</BODY>
</HTML>
```

***Ejemplo 12 – Operador relacional igualdad***

Es importante distinguir el operador de igualdad (**==**) del operador de asignación (**=**) ya que es origen de muchos errores y su detección no resulta fácil porque ambos se pueden utilizar en el mismo contexto con resultados completamente diferentes. Cuando se utilizan estos operadores con operandos del tipo cadenas de texto, los operadores "mayor que" (**>**) y "menor que" (**<**) comparan letra a letra comenzando desde la izquierda hasta que se encuentre una diferencia entre las dos cadenas de texto. Para determinar si una letra es mayor o menor que otra, las mayúsculas se consideran menores que las minúsculas y las primeras letras del alfabeto son menores que las últimas. La tabla ASCII (codificación numérica asociada a todos los caracteres visibles de un teclado) determina el orden en estas comparaciones. Se puede consultar una tabla ASCII en http://ascii.cl/es/

### 2.5.7 -OPERADORES LÓGICOS

Los operadores lógicos permiten combinar expresiones lógicas (aquellas que devuelven como resultado un valor ***true*** o ***false***). Responden a tablas de verdad pre-establecidas que determinan el resultado del operador a la combinación de valores de los operandos. El resultado siempre es un valor true o false. Dado que en algunos lenguajes de programación no existe el tipo lógico como tipo de datos, tradicionalmente se considera que cualquier valor numérico diferente de cero se considera true y cero se considera false. Por este motivo se pueden utilizar estos operadores con operandos numéricos. Así mismo, cuando se trabaja con cadenas de caracteres, la cadena vacía se considera false y el resto de cadenas true.

**Operador negación (!):** Utiliza un único operando y cuyo resultado es true cuando el operando vale false y false cuando el operando vale true.

```
var soltero = true;
alert(!soltero); // Muestra "false" y no "true"
```

**Operador Y-lógico (AND) (&&):** El operador lógico AND es un operador binario por lo que obtiene su resultado como resultado de evaluar dos operandos. La tabla de verdad de este operador devuelve true si los dos operandos son true y false en cualquier otro caso.

**Operador O-lógico (OR) (||):** El operador lógico OR es un operador binario por lo que obtiene su resultado como resultado de evaluar dos operandos. La tabla de verdad de este operador devuelve false si los dos operandos son false y true en cualquier otro caso.

## 2.6 - PRECEDENCIA DE OPERADORES

En una expresión pueden aparecer varios operadores y el resultado podría ser diferente si las operaciones se realizan en un orden o en otro. Como parece lógico pensar

no se puede dejar al azar este asunto ya que las expresiones deben ser evaluadas siguiendo unas pautas establecidas. Por ejemplo, la operación ***3+5*4*** proporcionará resultados distintos si primero se suma y luego con el resultado se multiplica que si se hace primero la multiplicación y luego la suma. En el primer caso el resultado sería ***32*** mientras que en el segundo sería ***23***. Para eliminar esta ambigüedad, los lenguajes de programación, como herencia de la matemática, establecen una jerarquía de operadores que determina el orden de evaluación de los operadores.

| Prioridad | Operadores | Operaciones |
|---|---|---|
| *1ª* | ++ -- - ¡ | *Incremento, decremento, cambio de signo, negación* |
| *2ª* | * / % | *Multiplicación, división, resto (módulo) de la división* |
| *3ª* | + - | *Suma, resta* |
| *4ª* | < > <= >= | *Menor, mayor, menor o igual, mayor o igual* |
| *5ª* | == ¡= | *Igual, distinto* |
| *6ª* | && | *Conjunción lógica (AND – Y lógico)* |
| *7ª* | \|\| | *Disyunción lógica (OR – O lógico)* |
| *8ª* | = += -= *= /= %= | *Asignación, asignación y suma, etc.* |

**Observaciones:** Los operadores ***++*** y ***--*** varían su comportamiento en una expresión dependiendo de si se aplican antes (pre-) o después (post-). Los operadores que se encuentran en el mismo grupo tienen la misma precedencia. En estos casos no se garantiza el orden de evaluación. Si queremos que se evalúen en algún orden concreto deberemos utilizar paréntesis.

La precedencia o prioridad establecida por defecto **se puede alterar utilizando paréntesis,** de manera que las expresiones que se encuentran dentro de los paréntesis se realizan antes que las restantes. Se pueden anidar paréntesis y dentro de los mismos se utiliza la precedencia por defecto.

JavaScript permite expresiones como: ***100000 >= SALARIO <= 200000***. Este tipo de expresiones es ilegal en otros lenguajes de programación y provocará un error ya que al evaluar la primera parte de la expresión se sustituirá por un valor lógico de tipo *true/false* y este resultado no puede compararse con un valor numérico. La expresión válida para la mayoría de los lenguajes de programación sería ***SALARIO >= 100000 && SALARIO <= 200000***.