

Desarrollo Web en Entorno Cliente

# **UD 01. Sintaxis Javascript ES6**

---

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

 **Importante**

 **Atención**

 **Interesante**

## ÍNDICE DE CONTENIDO

1.	Introducción	4
2.	Hola mundo y comentarios	4
3.	Variables y constantes	5
3.1	Ámbito de variables y constantes	6
3.2	Variables	6
3.3	Constantes	7
3.4	Tipos de datos	7
3.5	Coerción	8
3.6	Conversiones entre tipos	10
3.7	Modo estricto “use strict”	10
4.	Operadores	10
4.1	Operadores de asignación	11
4.2	Operadores aritméticos	12
4.3	Operadores de comparación	13
4.4	Operadores lógicos	14
5.	Entrada y salida en navegadores	15
5.1	alert	15
5.2	console.log	15
5.3	confirm	15
5.4	prompt	16
6.	Estructuras de control	16
6.1	Instrucciones if/else	16
6.2	Ramificaciones anidadas	18
6.3	Switch	18



## UD01. SINTAXIS JAVASCRIPT ES6

### 1. INTRODUCCIÓN

Javascript es un lenguaje de programación que inicialmente nació como un lenguaje que permitía ejecutar código en nuestro navegador (cliente), ampliando la funcionalidad de nuestros sitios web.

Una de las versiones más extendidas de Javascript moderno, es la llamada por por muchos **JavaScript ES6** (ECMAScript 6), también llamado ECMAScript 2015 o incluso por algunos llamado directamente Javascript 6. Al crear esta pequeña guía nos hemos basado en esta versión. Si ya sabes Javascript pero quieres conocer novedades de Javascript ES6, puedes acceder a este link <https://didacticode.com/curso/curso-javascript-es6/>

Actualmente esa es una de sus principales funciones, pero dada su popularidad el lenguaje ha sido portado a otros ámbitos, entre los que destaca el popular NodeJS <https://nodejs.org/> que permite la ejecución de Javascript como como lenguaje escritorio y lenguaje servidor.

Aunque en este módulo nos centramos en la ejecución de Javascript en el navegador, lo aprendido puede ser utilizado para otras implementaciones de Javascript.

### 2. HOLA MUNDO Y COMENTARIOS

Para añadir JavaScript se usa la etiqueta SCRIPT.

Este puede estar en cualquier lugar de la página. El código se ejecuta en el lugar donde se encuentra de forma secuencial a como el navegador lo va encontrando.

```
<SCRIPT LANGUAGE="JavaScript">
Aquí va el código
// Esto es un comentario en Javascript de una línea
</SCRIPT>
```

En Javascript los comentarios se pueden hacer con // para una línea y con /\* \*/ para varias líneas.

Asimismo desde Javascript es posible escribir mensajes a la consola de desarrollo mediante la orden "console.log(texto)".

Este ejemplo podría ser un pequeño hola mundo que al ejecutarse se mostrará en la consola de desarrollo. Ejemplo con "console.log" y comentario multilínea.

```
<SCRIPT LANGUAGE="JavaScript">
console.log ("Hola mundo");
```

```
/* Esto es un comentario en  
Javascript multilínea */  
</SCRIPT>
```

Otra vía para mostrar información al usuario desde una ventana, es el comando “alert(texto)”.

```
<SCRIPT LANGUAGE="JavaScript">  
alert("Hola mundo");  
</SCRIPT>
```

Hay una forma mucho más práctica y ordenada de usar código Javascript. Se pueden incluir uno o varios ficheros con código Javascript en nuestro documento HTML. Se puede incluir tantos como se desee. Esta es la forma más adecuada para trabajar con código Javascript.

Un ejemplo de inclusión de ficheros.

```
<script type="text/javascript" src="rutaDelArchivo1.js"/>  
<script type="text/javascript" src="rutaDelArchivo2.js"/>  
<script type="text/javascript" src="rutaDelArchivo3.js"/>  
  
<!DOCTYPE html>  
<html lang="en">  
<head>  
  
<meta charset="UTF-8">  
<meta http-equiv="X-UA-Compatible" content="IE=edge">  
<meta name="viewport" content="width=device-width, initial-scale=1.0">  
<script src="ejercicioCase"></script>  
<title>Camel case</title>  
</head>  
  
<body>  
  
</body>  
</html>
```

### 3. VARIABLES Y CONSTANTES

En este punto hablaremos de variables, constantes y arrays. Antes de empezar, comentar que es recomendable mantener un estilo de nombramiento de variables. Aquí un enlace sobre formas de nombrar variables:

<https://medium.com/@alonsus91/convenci%C3%B3n-de-nombres-desde-el-camelcase-hasta-el-kebab-case-787e56d6d023>

Durante el curso, usaremos el estilo “CamelCase”  
[https://es.wikipedia.org/wiki/Camel\\_case](https://es.wikipedia.org/wiki/Camel_case)

Importante recordar:

Las constantes siempre en mayúsculas

Las variables la primera en minúscula y la segunda mayúscula – nombreAlumno

Las clases la primera en mayúscula Alumno

### 3.1 Ámbito de variables y constantes

Antes de comenzar, vamos a hablar del ámbito de variables (también llamado “scope” en inglés). El ámbito de variables en un lenguaje de programación indica en qué lugares del programa puede ser accedida una variable/constante. Al comentar cada uno de los tipos, definiremos en qué ámbito existen.

### 3.2 Variables

Las variables son elementos del lenguaje que permiten almacenar distintos valores en cada momento. Se puede almacenar un valor en una variable y consultar este valor posteriormente. También podemos modificar su contenido siempre que queramos.

Para declarar las variables en JavaScript podemos utilizar **let** o **var**, según el ámbito donde queramos que sea accesible.

- **let**: let permite declarar una variable que sea accesible únicamente dentro del bloque donde se ha declarado (**llamamos bloque al espacio delimitado por { }**).
- **var**: var permite declarar una variable que sea accesible por todos los lugares de la función donde ha sido declarada. Si una variable con var se declara fuera de cualquier función, el ámbito de esta son todas las funciones del código.
- **Variables sin declarar**: Javascript nos permite usar variables no declaradas. Si hacemos esto, será equivalente a declararlas con **var** fuera del código, es decir, serán variables accesibles por cualquier función. **NO SE RECOMIENDA, ES UNA MALA PRACTICA.**

```
function ejemplo(){  
  ejemplo=3; // Equivale a declararla fuera de la funcion como var  
  if (ejemplo === 3){  
    var variable1 = 1;  
    let variable2 = 2;  
  }  
  console.log(variable1); // variable1 existe en este lugar  
  console.log(variable2); // variable2 no existe en este lugar  
}
```

### 3.3 Constantes

Las constantes son elementos que permiten almacenar un valor, pero ese valor una vez almacenado es invariable (permanece constante). Para declarar constantes se utiliza la instrucción “const”. Suelen ser útiles para definir datos constantes, como el número PI, el número de Euler, etc...


Su ámbito es el mismo que el de **let**, es decir, solo son accesibles en el bloque que se han declarado.

```
Const PI=3.1416;  
console.log(PI);  
EDADALUMNO=3;
```

Aunque resulta posible definir arrays y objetos usando **const**, no es recomendable hacerlo, ya que es posible que su uso no sea el que pensamos.

Por ejemplo, al declarar un array/objeto, realmente lo que ocurre es que la variable almacena la dirección de memoria del objeto/array. Si lo declaramos usando **const**, lo que haremos es que no pueda cambiarse esa dirección de memoria, pero nos permitirá cambiar sus valores.

```
const miArray=[1,2,3]  
console.log(miArray[0]); // Muestra el valor 1  
miArray[0]=4;  
console.log(miArray[0]); // Muestra el valor 4
```

 **Importante:** En general, recomendamos usar **let**. Deberías tener una buena razón para usar **var**, ya que su uso es peligroso ya que podría modificarse una variable desde un lugar que no controles por accidente. Estos problemas no ocurren únicamente en Javascript, sino en otros lenguajes y pueden hacer que tu programa sea complicado de depurar.

### 3.4 Tipos de datos

Los principales tipos de datos que puede contener variables en Javascript son:

- **Numéricos (tipo “number”)**: puede contener cualquier tipo de número real (0.3, 1.7, 2.9) o entero (5, 3, -1).
- **Enteros grandes (tipo “bigint”)**: pueden contener enteros con valores superiores a  $2^{53} - 1$ . Se pueden nomenciar escribiendo una letra “n” al final del entero. No pueden utilizarse con la mayoría de operadores matemáticos de Javascript.
- **Booleanos (tipo “boolean”)** : puede contener uno de los siguientes valores: true, false, 1 y 0.
- **Cadenas (tipo “string”)**: cualquier combinación de caracteres (letras, números, signos especiales y espacios). Las cadenas se delimitan mediante comillas dobles o

simples ("Lolo","laO"). Para concatenar cadenas puede usarse el operador "+".

El tipo de una variable puede comprobarse usando la estructura **typeof** variable=== "tipo".

```
let edad=23, nueva_edad, incremento=4;
const nombre="Rosa García";
console.log(typeof incremento=== "number")
nueva_edad=edad+incremento;
console.log(nombre+ " tras "+incremento +" años tendrá "+ nueva_edad);
```

Asimismo existen otros tipos que Javascript considera primitivos: "undefined", "null", "symbol", "object" y "function".

Todos los valores que NO son de un tipo básico son considerados objetos: arrays, funciones, String, etc. Esta distinción es muy importante porque los valores primitivos y los valores objetos se comportan de distinta forma cuando son asignados y cuando son pasados como parámetro a una función.

Más información en [https://developer.mozilla.org/es/docs/Web/JavaScript/Data\\_structures](https://developer.mozilla.org/es/docs/Web/JavaScript/Data_structures)

### 3.5 Coerción

Javascript es un lenguaje de tipado blando (es decir, **al declarar una variable no se le asigna un tipo**), aunque internamente Javascript si maneja tipos de datos.

En determinados momentos, resulta necesario convertir un valor de un tipo a otro. Esto en JS se llama "coerción", y puede ocurrir de forma implícita o podemos forzarlo de forma explícita.

Por ejemplo

```
let numero = 5;
console.log(numero);
```

En este código, ocurre una coerción **implícita** de número (que es de tipo number) a un tipo string, de modo que puede ser impreso por consola. Podríamos realizar la conversión de forma **explícita** de la siguiente forma:

```
console.log(numero.toString());
```

Las coerciones implícitas ocurren muy a menudo en JS, aunque muchas veces no seamos conscientes de ello. Resulta muy importante entender cómo funcionan para poder deducir cuál será el resultado de una comparación.



```
let a = "2", b = 5;
console.log( typeof a + " " + typeof b); // string number
console.log( a + b ); // nos muestra 25
```

En los lenguajes de tipado duro (por ejemplo, Java) se nos prohíbe realizar operaciones entre distintos tipos de datos. Sin embargo, Javascript, no hace eso, ya que permite operar entre distintos tipos, siguiendo una serie de reglas:

- Javascript tiene el operador `===` y `!==` para realizar comparaciones estrictas, pero no posee esos operadores para desigualdades (`>`, `<`, `>=`, `<=`).
- Si es posible, JS prefiere hacer coerciones a tipo `number` por encima de otros tipos básicos. Por ejemplo, la expresión `("15" < 100)` se resolverá como `true` porque JS cambiará `"15"`, de tipo `string`, por `15` de tipo `number`.
  - Ten en cuenta que si conviertes `"15"` a `string`, al compararlo con `"100"` la expresión se resolvería como `false`.
- A la hora de hacer coerción a `boolean`, los siguientes valores se convertirán en `false`: `undefined`, `null`, `0`, `""`, `NaN`. El resto de valores se convertirán en `true`.

Ejemplo coerción a `number`

```
// <, <=, >, <= también hacen coercion. No existe >= ni <=
let arr = [ "1", "10", "100", "1000" ];
for (let i = 0; i < arr.length && arr[i] < 500; i++) {
  console.log(i);
} // 0, 1, 2
```

Ejemplo donde no se hace coerción

```
var x = "10";
var y = "9";
console.log(x < y); // true, Los dos son String y los compara como cadena
```

🗨 **Interesante:** al realizar comparaciones, si usas `==` o `!=` para comparar los datos, Javascript realiza coerción. Si quieres que la comparación no convierta tipos y solo sea cierta si son del mismo tipo, debes usar `===` o `!==`. Esta es una buena práctica muy recomendada para que estas conversiones no nos jueguen malas pasadas.

Para más información <https://www.etnassoft.com/2011/04/06/coercion-de-datos-en-javascript/>

### 3.6 Conversiones entre tipos

Javascript no define explícitamente el tipo de datos de sus variables. Según se almacenen, pueden ser cadenas (Entre Comillas), enteros (sin parte decimal) o decimales (con parte decimal).

Elementos como la función “prompt” para leer de teclado con una ventana emergente del navegador, leen los elementos siempre como cadena. Para estos casos y otros, merece la pena usar funciones de conversión de datos.

```
let num="100"; //Es una cadena
let num2="100.13"; //Es una cadena
let num3=11; // Es un entero
let n=parseInt(num); // Almacena un entero. Si hubiera habido parte decimal la truncará
let n2=parseFloat(num); // Almacena un decimal
let n3=num3.toString(); // Almacena una cadena
```

### 3.7 Modo estricto “use strict”

Javascript ES6 incorpora el llamado “modo estricto”. Si en algún lugar del código se indica la sentencia “use strict” indica que ese código se ejecutará en modo estricto:

- Escribir “use strict” fuera de cualquier función afecta a todo el código.
- Escribir “use strict” dentro de una función afecta a esa función.

Entre las principales características de “use strict” tenemos que **obliga a que todas las variables sean declaradas**.

“Use strict” es ignorado por versiones anteriores de Javascript, al tomarlo como una simple declaración de cadena.

Las principales características de “use strict” son:

- No permite usar variables/objetos no declarados (los objetos son variables).
- No permite eliminar (usando delete) variables/objetos/funciones.
- No permite nombres duplicados de parámetros en funciones.
- No permite escribir en propiedades de objetos definidas como solo lectura.
- Evita que determinadas palabras reservadas sean usadas como variables (eval, arguments, this, etc...)

## 4. OPERADORES

Combinando variables y valores, se pueden formular expresiones más complejas. Las expresiones son una parte clave en la creación de programas. Para formular expresiones

utilizamos los llamados operadores. Pasamos a comentar los principales operadores de Javascript.

#### 4.1 Operadores de asignación

Los operadores de asignación se utilizan para asignar valores a las variables. Algunos de ellos, además de asignar el valor, también incluyen operaciones (Ejemplo += asigna y suma).

Operador	Descripción
=	Asigna a la variable de la parte izquierda el valor de la parte derecha.
+=	Suma los operandos izquierdo y derecho y asigna el resultado al operando izquierdo. 5+=1
-=	Resta el operando derecho del operando izquierdo y asigna el resultado al operando izquierdo.
*=	Multiplica ambos operandos y asigna el resultado al operando izquierdo.
/=	Divide ambos operandos y asigna el resultado al operando izquierdo.
%=	Divide ambos operandos y asigna el resto de la división al operando izquierdo.

```
let num1=3;
let num2=5;
num2+=num1;
num2-=num1;
num2*=num1;
num2/=num1;
num2%=num1;
```

Esto simplemente asigna 3 a **num1** y 5 a **num2**.

3. **num2+=num1**; Esta operación es equivalente a **num2 = num2 + num1**.  
Así que, **num2** ahora es 8 (5 + 3).
4. **num2-=num1**; Esta operación es equivalente a **num2 = num2 - num1**.  
Entonces, **num2** se convierte en 5 (8 - 3).
5. **num2\*=num1**; Esta operación es equivalente a **num2 = num2 \* num1**.  
Así que, **num2** ahora es 15 (5 \* 3).
6. **num2/=num1**; Esta operación es equivalente a **num2 = num2 / num1**.

Por lo tanto, `num2` es 5 (`15 / 3`).

7. `num2%=num1`; Esta operación es equivalente a `num2 = num2 % num1`.

Dado que 5 dividido entre 3 da un residuo de 2, `num2` es ahora 2. Por lo tanto, después de ejecutar todas estas operaciones, `num2` tendrá el valor 2.

## 4.2 Operadores aritméticos

Los operadores aritméticos se utilizan para realizar cálculos aritméticos.

Operador	Descripción
+	Suma.
-	Resta.
*	Multiplicación.
/	División.
%	Calcula el resto de una división entera.

Además de estos operadores, también existen operadores aritméticos unitarios: incremento (`++`), disminución (`--`) y la negación unitaria (`-`).

Los operadores de incremento y disminución pueden estar tanto delante como detrás de una variable, con distintos matices en su ejecución. Estos operadores aumentan o disminuyen en 1, respectivamente, el valor de una variable.

Operador	Descripción (Suponiendo <code>x=5</code> )
<code>y = ++x</code>	Primero el incremento y después la asignación. <code>x=6</code> , <code>y=6</code> .
<code>y = x++</code>	Primero la asignación y después el incremento. <code>x=6</code> , <code>y=5</code> .
<code>y = --x</code>	Primero el decremento y después la asignación. <code>x=4</code> , <code>y=4</code> .
<code>y = x--</code>	Primero la asignación y después el decremento <code>x=4</code> , <code>y=5</code> .
<code>y = -x</code>	Se asigna a la variable "y" el valor negativo de "x", pero el valor de la variable "x" no varía. <code>x=5</code> , <code>y= -5</code> .

```
let num1=5, num2=8,resultado1, resultado2;
resultado1=((num1+num2)*200)/100;
resultado2=resultado1%3;
resultado1=++num1;
resultado2=num2++;
resultado1=--num1;
resultado2=num2--;
resultado1=-resultado2;
```

### 4.3 Operadores de comparación

Operadores utilizados para comparar dos valores entre sí. Como valor de retorno se obtiene siempre un valor booleano: *true* o *false*.

Operador	Descripción
===	Compara dos elementos, incluyendo su tipo interno. Si son de distinto tipo, no realiza conversión y devuelve false ya que siempre los considera diferentes. <b><u>Uso recomendado.</u></b>
!==	Compara dos elementos, incluyendo su tipo interno. Si son de distinto tipo, no realiza conversión y devuelve true ya que siempre los considera diferentes. <b><u>Uso recomendado.</u></b>
==	Devuelve el valor <i>true</i> cuando los dos operandos son iguales. Si los elementos son de distintos tipos, realiza una conversión. <b><u>No está recomendado su uso.</u></b>
!=	Devuelve el valor <i>true</i> cuando los dos operandos son distintos. Si los elementos son de distintos tipos, realiza una conversión. <b><u>No está recomendado su uso.</u></b>
>	Devuelve el valor <i>true</i> cuando el operando de la izquierda es mayor que el de la derecha.
<	Devuelve el valor <i>true</i> cuando el operando de la derecha es menor que el de la izquierda.
>=	Devuelve el valor <i>true</i> cuando el operando de la izquierda es mayor o igual que el de la derecha.
<=	Devuelve el valor <i>true</i> cuando el operando de la derecha es menor o igual

	que el de la izquierda.
--	-------------------------

```
let a=4;b=5,c="5";
console.log("El resultado de la expresión 'a==c' es igual a "+(a==c));
console.log("El resultado de la expresión 'a===c' es igual a "+(a===c));
console.log("El resultado de la expresión 'a!=c' es igual a "+(a!=c));
console.log("El resultado de la expresión 'a!==c' es igual a "+(a!==c));
console.log("El resultado de la expresión 'a==b' es igual a "+(a==b));
console.log("El resultado de la expresión 'a!=b' es igual a "+(a!=b));
console.log("El resultado de la expresión 'a>b' es igual a "+(a>b));
console.log("El resultado de la expresión 'a<b' es igual a "+(a<b));
console.log("El resultado de la expresión 'a>=b' es igual a "+(a>=b));
console.log("El resultado de la expresión 'a<=b' es igual a "+(a<=b));
```

#### 4.4 Operadores lógicos

Los operadores lógicos se utilizan para el procesamiento de los valores booleanos. A su vez el valor que devuelven también es booleano: true o false.

Operador	Descripción
&&	Y “lógica”. El valor de devolución es true cuando ambos operandos son verdaderos.
	O “lógica”. El valor de devolución es true cuando alguno de los operandos es verdadero o lo son los dos.
!	No “lógico”. Si el valor es true, devuelve false y si el valor es false, devuelve true.

Se muestra el resultado de distintas operaciones realizadas con operadores lógicos. (En el ejemplo se usa directamente los valore true y false en lugar de variables).

```
console.log("El resultado de la expresión 'false&&false' es igual a "+(false&&false));
console.log("El resultado de la expresión 'false&&true' es igual a "+(false&&true));
console.log("El resultado de la expresión 'true&&false' es igual a "+(true&&false));
```

```
"+(true&&false));
console.log("El resultado de la expresión 'true&&true' es igual a
"+(true&&true));
console.log("El resultado de la expresión 'false||false' es igual a
"+(false||false));
console.log("El resultado de la expresión 'false||true' es igual a
"+(false||true));
console.log("El resultado de la expresión 'true||false' es igual a
"+(true||false));
console.log("El resultado de la expresión 'true||true' es igual a
"+(true||true));
console.log("El resultado de la expresión '!false' es igual a
"+(!false));
```

Para saber más de comparadores y expresiones, puedes consultar:

[https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Expressions\\_and\\_Operator\\_s](https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Expressions_and_Operator_s)

## 5. ENTRADA Y SALIDA EN NAVEGADORES

### 5.1 alert

El método `alert()` permite mostrar al usuario información literal o el contenido de variables en una ventana independiente. La ventana contendrá la información a mostrar y el botón Aceptar.

```
alert("Hola mundo");
```

### 5.2 console.log

El método `console.log` permite mostrar información en la consola de desarrollo.

```
console.log("Otro hola mundo");
```

### 5.3 confirm

A través del método `confirm()` se activa un cuadro de diálogo que contiene los botones Aceptar y Cancelar. Cuando el usuario pulsa el botón Aceptar, este método devuelve el valor `true`; Cancelar devuelve el valor `false`. Con ayuda de este método el usuario puede decidir sobre preguntas concretas e influir de ese modo directamente en la página.

```
let respuesta;  
respuesta=confirm ("¿Desea cancelar la suscripción?");  
alert("Usted ha contestado que "+respuesta);
```

## 5.4 prompt

El método prompt() abre un cuadro de diálogo en pantalla en el que se pide al usuario que introduzca algún dato. Si se pulsa el botón Cancelar, el valor de devolución es false/null. Pulsando en Aceptar se obtiene el valor true y la cadena de caracteres introducida se guarda para su posterior procesamiento.

```
let provincia;  
provincia=prompt("Introduzca la provincia ", "Valencia");  
alert("Usted ha introducido la siguiente información "+provincia);  
console.log("Operación realizada con éxito");
```

## 6. ESTRUCTURAS DE CONTROL

### 6.1 Instrucciones if/else

Para controlar el flujo de información en los programas JavaScript existen una serie de estructuras condicionales y bucles que permiten alterar el orden secuencial de ejecución. Estas son las instrucciones if y else.

La instrucción if es permite la ejecución de un bloque u otro de instrucciones en función de una condición.

#### Sintaxis

```
if (condición) {  
    // bloque de instrucciones que se ejecutan si la condición se cumple  
}  
else{  
    // bloque de instrucciones que se ejecutan si la condición no se cumple  
}
```

Las llaves solo son obligatorias cuando haya varias instrucciones seguidas pertenecientes a la ramificación. Si no pones llaves, el if se aplicará únicamente a la siguiente instrucción.

Puede existir una instrucción if que no contenga la parte else. En este caso, se ejecutarán una serie de órdenes si se cumple la condición y si esto no es así, se continuaría con las órdenes que están a continuación del bloque if. Ejemplos de uso:



```
let diaSem;
diaSem=prompt("Introduce el día de la semana ", "");
if (diaSem === "domingo")
{
    console.log("Hoy es festivo");
}
else // Al no tener {}, es un "bloque de una instrucción"
    console.log("Hoy no es domingo, a descansar!!");
```

#### Otro ejemplo

```
let edadAna,edadLuis;
// Usamos parseInt para convertir a entero
edadAna=parseInt(prompt("Introduce la edad de Ana",""));
edadLuis=parseInt(prompt("Introduce la edad de Luis",""));

if (edadAna > edadLuis){
    console.log("Ana es mayor que Luis.");
    console.log(" Ana tiene "+edadAna+" años y Luis "+ edadLuis);
}
else{
    console.log("Ana es menor o de igual edad que Luis.");
    console.log(" Ana tiene "+edadAna+" años y Luis "+ edadLuis);
}
```

#### Ejercicio 1

Detecta el error en este código que sirve para saber si el valor de la variable número es positivo o negativo

```
let numero=8
if (numero > 0) {
    console.log("El número es positivo");
}
else
    console.log("El número no es positivo");
}
```

#### Ejercicio 2

Completa las líneas de código para conocer si el alumno es mayor o menor de edad

```
let alu-edad = 16;
if (alu-edad _____) {
```

```
    console.log("Eres mayor de edad.");  
  } else {  
  
    console.log("Eres _____ de edad.");  
  }
```

## 6.2 Ramificaciones anidadas

Para las condiciones ramificadas más complicadas, a menudo se utilizan las ramificaciones anidadas. En ellas se definen consultas if dentro de otras consultas if, por ejemplo:

```
let edadAna,edadLuis;  
// Convertirnos a entero las cadenas  
edadAna=parseInt(prompt("Introduce la edad de Ana",""));  
edadLuis=parseInt(prompt("Introduce la edad de Luis",""));  
if (edadAna > edadLuis){  
    console.log("Ana es mayor que Luis.");  
}  
else{  
    if (edadAna<edadLuis){  
        console.log("Ana es menor que Luis.");  
    }else{  
        console.log("Ana tiene la misma edad que Luis.");  
    }  
}  
console.log(" Ana tiene "+edadAna+" años y Luis "+ edadLuis);
```

**Ejercicio 3.** Escribe un programa que pida 3 números y muestre con un alert() el número que sea mayor.

## 6.3 SWITCH

Nos sirve para tomar decisiones en función de distintos estados de las variables. Esta expresión se utiliza cuando tenemos múltiples posibilidades como resultado de la evaluación de una sentencia.

Su sintaxis es la siguiente:

```
switch (expresión) {  
    case valor1:  
        Sentencias a ejecutar si la expresión tiene como valor a valor1  
    break
```

```
    case valor2:
      Sentencias a ejecutar si la expresión tiene como valor a valor2
    break
    case valor3:
      Sentencias a ejecutar si la expresión tiene como valor a valor3
    break
    default:
      Sentencias a ejecutar si el valor no es ninguno de los anteriores
  }
```

La expresión se evalúa, si vale valor1 se ejecutan las sentencias relacionadas con ese caso. Si la expresión vale valor2 se ejecutan las instrucciones relacionadas con ese valor y así sucesivamente, por tantas opciones como deseemos. Finalmente, para todos los casos no contemplados anteriormente se ejecuta el caso por defecto.

La palabra break es opcional, pero si no la ponemos a partir de que se encuentre coincidencia con un valor se ejecutarán todas las sentencias relacionadas con este y todas las siguientes. Es decir, si en nuestro esquema anterior no hubiese ningún break y la expresión valiese valor1, se ejecutarían las sentencias relacionadas con valor1 y también las relacionadas con valor2, valor3 y default.

#### Ejercicio 4

Detecta los errores de este código

```
let fruta = "Platano";
switch(fruta) {
  case 'Manzana':
    console.log("El color de la fruta es rojo.");
    break;
  case 'Plátano':
    console.log("El color de la fruta es amarillo.");
  case 'Naranja':
    console.log("El color de la fruta es naranja.");
  default:
    console.log("Fruta no reconocida.");
}
```

#### Ejercicio 5 Detecta los errores de este código

```
let mes = 2;
```

```
switch(mes) {  
  case 1:  
    console.log("Enero");  
    break;  
  case 2:  
    console.log("Febrero");  
    break;  
  case 3:  
    console.log("Marzo");  
    break;  
  default:  
    console.log("Mes no válido.");  
}
```

**Ejercicio 6.** Escribe un programa que solicite al usuario salario y rango, en caso de que el rango sea 1 incrementa el salario en un 20%, si el rango es 2 incrementa el salario un 10%, si el rango es 3 incrementa el salario un 5% y si el rango es 4 increméntalo en un 3%.