# JavaScript

**Sintaxis**

# Sentencias


Una *sentencia* es una línea de código que realiza una operación.

- Una sentencia por línea.

```
alert("HOLA");                    // ; opcional
alert("ADIOS")
```

- Varias sentencias en línea.

```
alert('HOLA'); alert("ADIOS");      // ; obligatorio
```

***(*) Se recomienda terminar todas las sentencias con ; sea o no obligatorio.***

# Comentarios


Los *comentarios* son anotaciones en el código que no son interpretados por el navegador y se añaden para aclarar su funcionamiento.

- Comentarios de una línea:

```
// Esto es una sentencia
alert("HOLA"); // Comentario de una línea
```

- Comentarios multi-línea:

```
/*
*   Comantario de
*   varias líneas.
*/
```

# Variables y tipos


Las *variables* almacenan datos que los scripts necesitan para funcionar.

- Datos introducidos en campos de un formulario
- Datos concatenados en la URL de la página.
- Datos incluidos en el propio código.

Los datos pueden ser de diferentes *tipos*:

- números
- textos
- lógicos

# Declaración y asignación

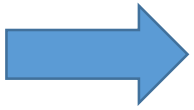

- Declaración (***let***):  (se declara la variable para su uso):

```
let nombre;                    // Declaraciones
let edad;
```

- Asignación: (se le asigna un valor inicial ):

```
nombre = "pepe"                // Asignaciones
edad = 10
```

- Declaración + Asignación:



```
let nombre = "Pepe";
let edad = 10;
```

***(*) NO puede asignarse un valor a una variable si no está declarada primero.***

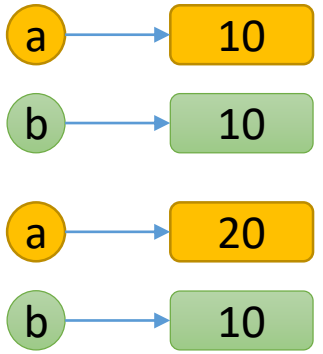
# Uso de variables

- La asignación de un valor destruye el anterior.

```
let valor;
valor = 10
alert(valor);   // Muestra 10
valor = 20
alert(valor);   // Muestra 20
```

- Cada variable tiene su propio valor ( modificar el valor de una no afecta a otra ).

```
let a = 10;
let b = a;
console.log(a);     // Muestra 10
console.log(b);     // Muestra 10 ( toma valor de a)
a = 20;
console.log(a);     // Muestra 20
console.log(b);     // Muestra 10
```



# Normas de los identificadores

- No pueden declararse dos variables con el mismo nombre:

```
let valorA = 10;
let valorA = 20;    // Redeclaración de la misma variable !! ERROR !!
```

```
Uncaught SyntaxError: Identifier 'valorA' has already been declared                prueba.js:4
```

- Se distingue entre mayúsculas y minúsculas. La variable "*nombre*" es considerada distinta de la variable "*Nombre*", o "*NOMBRE*".
- Las variables no pueden tener como nombres palabras claves del lenguaje:

```
let let = 10;
alert(let);
```

```
Uncaught SyntaxError: let is disallowed as a lexically bound name                  prueba.js:3
```

# Normas de los identificadores


- Los identificadores pueden contener dígitos salvo en el primer carácter:

```
let nombre1;
let 1nombre;        // Error de identificador
```

- Los identificadores con nombres compuestos emplean la *nomenclatura camello:*

```
let contadorPersonas;
let numeroErroresRegistrados;
let claveCorrecta;
```

*La primera letra del identificador es minúscula, y las primeras letras del resto de palabras son mayúsculas.*

- Caracteres prohibidos: (*+,-,*,/,|,&,<,>,=*),
- Caracteres permitidos: (_, $).

# Constantes

Las constantes son variables que almacenan un valor fijo.

- Declaración + asignación:

```
const VELOCIDAD = 10;
```

- Debe anteponerse la palabra clave ***const*** en vez de ***let***.
- El identificados *en MAYUSCULAS* para diferenciarlo de las variables ordinarias.
- Cualquier intento de modificar el valor de la constante genera un error:

```
const VELOCIDAD = 10;
VELOCIDAD = 20;     // Intento cambio valor
```

```
▶ Uncaught TypeError: Assignment to constant variable.
    at variables.html:18:19
```

- USO: Las constantes se emplean para representar valores fijos con nombres identificativos:

```
const IVA = 0.21    // Tasa de Impuesto valor añadido.
```

# Tipos de datos primitivos.


Las variables pueden almacenar valores siguientes tipos:

- **Number**: Valores numéricos enteros o decimales: 5, 3.14, -10.
- **String**: Textos declarados entre comillas simples (') o dobles ("): 'Hola', "Mundo".
- **Boolean**: Valor lógico resultado de una comparación que puede ser ***true*** (cierto) o ***false*** (falso)

```
let cadena = "HOLA";        // string
let edad = 20;              // number
let altura = 1.45;          // number
let activo = true;          // boolean
```

- **Undefined**: Indica una variable declarada pero sin valor asignado.

```
let valor;
console.log(valor);     // Devuelve "undefined"
```

- **Null**: Indica una variable a la que se le asignado un valor vacío ( ***null*** ).

```
let valor = null;   // valor nulo.
```

# Operadores aritméticos.


Permiten realizar operaciones entre valores numéricos:

| Operador | Descripción | Ejemplo | Resultado |
|---|---|---|---|
| + | Suma | `resultado = 5 + 3;` | 8 |
| – | Resta | `resultado = 5 - 3;` | 2 |
| * | Multiplicación | `resultado = 5 * 3;` | 15 |
| / | División | `resultado = 6 / 3;` | 2 |
| % | Módulo (Resto de División) | `resultado = 7 % 3;` | 1 |

- Los operandos de una operación pueden ser valores u otras variables:

```
let precio_fruta = 1.20
let precio_carne = 6.50
let total = precio_fruta + precio_carne      // suma
```

# Operadores aritméticos.


Simplifican las operaciones sobre el valor de una variable.

| Operador | Descripción | Ejemplo | Resultado |
|---|---|---|---|
| ++ | Unario de incremento (añade 1 al valor) | `let numero = 5;`<br>`numero++;` | numero resulta 6 |
| — | Unario de decremento (resta 1 al valor) | `let numero = 5;`<br>`numero--;` | numero resulta 4 |
| += | Incremento por un valor específico | `let numero = 5;`<br>`numero += 3;` | numero resulta 8 |
| -= | Decremento por un valor específico | `let numero = 5;`<br>`numero -= 2;` | numero resulta 3 |
| *= | Multiplicación por un valor específico | `let numero = 5;`<br>`numero *= 2;` | numero resulta 10 |
| /= | División por un valor específico | `let numero = 10;`<br>`numero /= 2;` | numero resulta 5 |
| %= | Módulo (Resto de División) | `let numero = 7;`<br>`numero %= 3;` | numero resulta 1 |

# Concatenación de textos.


El operador “+” permite concatenar dos textos en una solos:

```
let amigo = "Luis";
let amiga = "Ana";

console.log( amigo + " y " + amiga + " son amigos.");   // Luis y Ana son amigos.
```

También pueden definirse cadenas combinadas con la tilde (`):

```
console.log(`${amigo} y ${amiga} son amigos.`);   // Luis y Ana son amigos.
```

Cada variable a concatenar se incluye entre ***${variable}***

# Operadores comparación.


Permiten comparar dos valores retornando un resultado lógico ***cierto/falso***:

| Operador | Significado | Ejemplo | Resultado |
|---|---|---|---|
| == | Igual a | 5 == 5<br>5 == "5" | true<br>true |
| === | Igual a (estricto) | 5 === '5'<br>5 === 5 | false<br>true |
| != | Diferente de | 5 != 3 | true |
| !== | Diferente de (estricto) | 5 !== '5' | true |
| > | Mayor que | 5 > 3 | true |
| < | Menor que | 3 < 5 | true |
| >= | Mayor o igual que | 5 >= 5 | true |
| <= | Menor o igual que | 3 <= 5 | true |

Los operadores ***===*** y ***!==*** y comparan tanto los valores como sus tipos.

# Operadores comparación.


Comparación de valores numéricos:

```
let a = 10;
let b = 20;
let iguales = ( a == b );   // false
let mayor = ( a > b );      // false
let menor = ( a < b);       // true
```

Comparación de cadenas:

```
let a = "Maria";
let b = "Pedro";
let iguales = ( a == b );   // false
let mayor = ( a > b );      // false
let menor = ( a < b);       // true
```

(*) Los operadores **>, >=, <, <=** se resuelven entre cadenas aplicando el orden alfabético.

# Operadores lógicos.


Combinan dos comparaciones simples en una comparación más compleja:

| Operador | Uso | Descripción |
|---|---|---|
| && | `expr1 && expr2` | Devuelve verdadero si ambos operandos son verdaderos |
| \|\| | `expr1 \|\| expr2` | Devuelve verdadero si uno de los operandos es verdadero |
| ! | `!expr` | Devuelve verdadero si el operando es falso, o falso si es verdadero. |

Ejemplo operador **//** ( OR )

```
let x = 10;
let cond_A;
// Comprueba si X vale 5, 10 o 7.
cond_A = ( x == 5 || x == 10 || x == 7 );
console.log(cond_A);            // Muestra TRUE
x = 20;
cond_A = ( x == 5 || x == 10 || x == 7 );
console.log(cond_A);            // Muestra FALSE
```

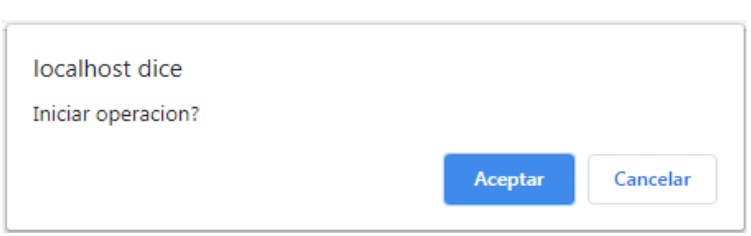
Ejemplo operador **&&** ( AND )

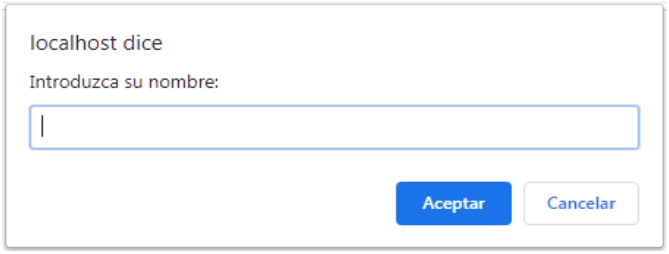
```
let x = 10;
let cond_A;
// Comprueba si X está entre 5 y 15 ambos incluidos.
cond_A = ( x >= 5 && x <= 15 );
console.log(cond_A);            // Muestra TRUE
x = 20;
cond_A = ( x == 5 || x == 10 || x == 7 );
console.log(cond_A);            // Muestra FALSE
```

# Cuadros de diálogo predefinidos

Los cuadros de diálogo son ventanas que se muestran sobre la web para:

- Mostrar una información o advertencia ( *alerta* )
- Solicitar permiso ( *confirmación* )
- Requerir la inserción de un valor ( *petición* )


El aspecto de estos cuadros de diálogo depende del navegador y no puede personalizarse.

# Cuadro de alerta

Esta función puede recibir un valor como argumento que puede ser de cualquier tipo (*any*) y que muestra en un cuadro de diálogo de información:

***alert***( *mensaje: any) : void*

```
alert();                    // Muestra un cuadro de diálogo 'vacio'
let nombre = "Marcelo";
let edad = 21
let activo = true;
alert(nombre);              // Muestra cuadro de diálogo 'Marcelo'
alert(edad);                // Muestra cuadro de diálogo '21'
alert(activo);              // Muestra cuadro de diálogo 'true'
```

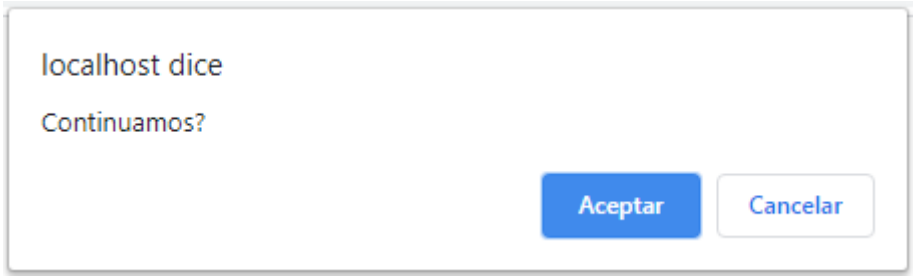
# Cuadro de confirmación


Muestra un mensaje solicitando al usuario una respuesta mediante los botones *Aceptar* o *Cancelar*.

***confirm****( mensaje: string) : boolean*

- La función retorna un valor lógico (*boolean*) ***cierto*** si el usuario pulsa "*Aceptar*" o ***falso*** si pulsa "*Cancelar*".

```
let ok = confirm("Continuamos?");
alert(ok);
            // Muestra 'true' si pulsa "ACEPTAR"
            // Muestra 'false' si pulsa "CANCELAR"
```



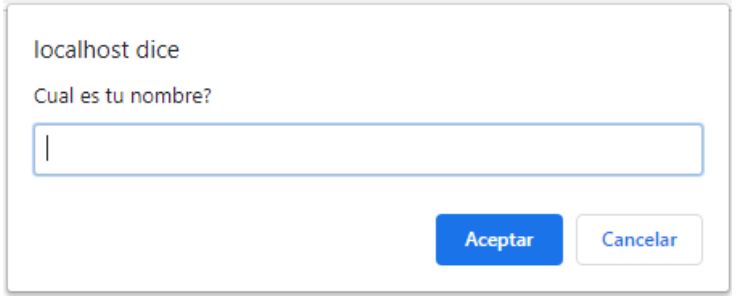
# Cuadro de solicitud.


Solicita un valor al usuario a ser indicado en la caja de texto del cuadro de diálogo y pulsar los botones Aceptar o Cancelar:

***prompt****( mensaje: string, valor_defecto:string) : string*

- La función requiere una cadena con el mensaje para el usuario, una segunda cadena con el valor mostrado por defecto:

```
let nombre = prompt("Cual es tu nombre?", "");
alert(nombre);
```




- La función retorna *una cadena* con el valor indicado si se pulsa "*Aceptar*", o el valor nulo (***null***) si se pulsa "*Cancelar*".

# Conversiones de tipos

Las conversiones permiten convertir un valor de un tipo a otro:

- ***Conversión a número***→ Convertir cualquier valor dado a número (***si es posible***).
  - ➢***Number****(value:any):number*

```
let nombre = "HOLA";          // Valor cadena
let edad = "21";              // Cadena con un valor numérico.
// Conversión cadena a número.
let numero_cadena = Number( nombre );       // Devuelve 'NaN' (Not a Number)
let numero_edad = Number(edad);             // Devuelve valor 21
```

***(*) Si la cadena no contiene un valor numérico se obtiene un valor "NaN" que representa un error de conversión de cadena a número:***

```
let cadena_vacia = "";
// Conversion a numero de cadena vacía
let valor_cadena_vacia = Number(cadena_vacia);      // Retorna 0;
```

# Conversiones de tipos


La conversión de cadena a número es necesaria en la solicitud de valores numéricos para cálculos:

```
let v1 = prompt("Indica un valor A: ", "");
let v2 = prompt("Indica un valor B: ", "");
let r = v1 + v2;    // Concatena NO SUMA
alert(`El resultado de la suma es: ${r}.`);
```

El valor retornado por la función ***prompt()*** *es retornado como una cadena de texto* aunque el usuario indique un valor numérico.

Para obtener el resultado correcto debe convertirse la cadena en un valor numérico al que podemos sumar.

```
let v1 = Number( prompt("Indica un valor A: ", "") );
let v2 = Number( prompt("Indica un valor B: ", "") );
let r = v1 + v2;
alert(`El resultado de la suma es: ${r}.`);
```

Es preciso emplear la función ***Number()*** para convertir las cadenas a valores numéricos antes de operarlo aritméticamente.