JAVASCRIPT

FUNCIONES Y PROPIEDADES BÁSICAS

CADENAS DE TEXTO

- Existen **3 formas** distintas de expresar un literal de texto:
  - `let cadenaTexto1 = "Esto es una cadena de texto";`
  - `let cadenaTexto2 = 'Esto es una cadena de texto';`
  - `let cadenaTexto3 = \`Esto es una cadena de texto\`;`

- Las 2 primeras son equivalentes.
  - `'Y me dijo: "Hola" '`
  - `"La variable 'x' no está definida"`
- La tercera forma se conoce como **template string** y fue introducida con ES2015. Permite mostrar el valor de variables y ejecutar código JavaScript sin tener que salir de la cadena de texto. Además permite cadenas multilínea.

CADENAS DE TEXTO

- Ejemplo usando template string

```
let x = 5;
console.log ( `La variable 'x' vale ${x} unidades` );
console.log ( `El cubo de 'x' es ${Math.pow(x,3)} unidades` );
// multilínea
console.log ( `El cubo de 'x' es
                    ${Math.pow(x,3)} unidades` );
```

TEXTO UNICODE

- **Unicode** es el nombre por el que se conoce al sistema moderno de codificación de caracteres que se usa en informática. A grandes rasgos, cada carácter como podría ser la A, la B (o cualquier otro), tiene su representación Unicode, que **se basa en un código o code point**.
- Por ejemplo, tomemos como ejemplo el carácter A (mayúsculas). Este carácter corresponde al **código Unicode U+0041**. Con esto, podemos representar esos caracteres en HTML o en texto.

FUNCIONES Y PROPIEDADES BÁSICAS

- JavaScript incorpora una serie de herramientas y utilidades (llamadas funciones y propiedades) para el manejo de las variables. De esta forma, muchas de las operaciones básicas con las variables, se pueden realizar directamente con las utilidades que ofrece JavaScript.

FUNCIONES PARA CADENAS DE TEXTO

- A continuación se muestran algunas de las funciones más útiles para el manejo de cadenas de texto:
- **length**, calcula la longitud de una cadena de texto (el número de caracteres que la forman)

```
let mensaje = "Hola Mundo";
let numeroLetras = mensaje.length; // numeroLetras = 10
```

- **+**, se emplea para concatenar varias cadenas de texto

```
let mensaje1 = "Hola";
let mensaje2 = " Mundo";
let mensaje = mensaje1 + mensaje2; // mensaje = "Hola Mundo"
```

FUNCIONES PARA CADENAS DE TEXTO

- Además del operador **+**, también se puede utilizar la función **concat()**

```
let mensaje1 = "Hola";
let mensaje2 = mensaje1.concat(" Mundo"); // mensaje2 = "Hola Mundo"
```

- Las cadenas de texto también se pueden unir con variables numéricas:

```
let variable1 = "Hola ";
let variable2 = 3;
let mensaje = variable1 + variable2; // mensaje = "Hola 3“
```

- Cuando se unen varias cadenas de texto es habitual olvidar añadir un espacio de separación entre las palabras. Los espacios en blanco se pueden añadir al final o al principio de las cadenas y también se pueden indicar forma explícita:

```
let mensaje1 = "Hola";
let mensaje2 = "Mundo";
let mensaje = mensaje1 + " " + mensaje2; // mensaje = "Hola Mundo"
```

FUNCIONES PARA CADENAS DE TEXTO

- **toUpperCase()**, transforma todos los caracteres de la cadena a sus correspondientes caracteres en mayúsculas:

```
let mensaje1 = "Hola";
let mensaje2 = mensaje1.toUpperCase(); // mensaje2 = "HOLA"
```

- **toLowerCase()**, transforma todos los caracteres de la cadena a sus correspondientes caracteres en minúsculas:

```
let mensaje1 = "HolA";
let mensaje2 = mensaje1.toLowerCase(); // mensaje2 = "hola"
```

- **charAt(posicion)**, obtiene el carácter que se encuentra en la posición indicada:

```
let mensaje = "Hola";
let letra = mensaje.charAt(0); // letra = H
letra = mensaje.charAt(2); // letra = l
```

FUNCIONES PARA CADENAS DE TEXTO

- **indexOf(caracter)**, calcula la posición en la que se encuentra el carácter indicado dentro de la cadena de texto. Si el carácter se incluye varias veces dentro de la cadena de texto, se devuelve su primera posición empezando a buscar desde la izquierda. Si la cadena no contiene el carácter, la función devuelve el valor -1:

```
let mensaje = "Hola";
let posicion = mensaje.indexOf('a'); // posicion = 3
posicion = mensaje.indexOf('b'); // posicion = -1q
```

- **lastIndexOf(caracter)**, calcula la última posición en la que se encuentra el carácter indicado dentro de la cadena de texto. Si la cadena no contiene el carácter, la función devuelve el valor -1:

```
let mensaje = "Hola";
let posicion = mensaje.lastIndexOf('a'); // posicion = 3
posicion = mensaje.lastIndexOf('b'); // posicion = -1
```

- La función lastIndexOf() comienza su búsqueda desde el final de la cadena hacia el principio, aunque la posición devuelta es la correcta empezando a contar desde el principio de la palabra.

FUNCIONES PARA CADENAS DE TEXTO

- **substring(inicio, final)**, extrae una porción de una cadena de texto. El segundo parámetro es opcional. Si sólo se indica el parámetro inicio, la función devuelve la parte de la cadena original correspondiente desde esa posición hasta el final:

```
let mensaje = "Hola Mundo";
let porcion = mensaje.substring(2); // porcion = "la Mundo"
porcion = mensaje.substring(5); // porcion = "Mundo"
porcion = mensaje.substring(7); // porcion = "ndo"
```

- Si se indica un inicio negativo, se devuelve la misma cadena original:

```
let mensaje = "Hola Mundo";
let porcion = mensaje.substring(-2); // porcion = "Hola Mundo"
```

- Cuando se indica el inicio y el final, se devuelve la parte de la cadena original comprendida entre ambas posiciones (incluida la primera y sin incluir la última) :

```
let mensaje = "Hola Mundo";
let porcion = mensaje.substring(1, 8); // porcion = "ola Mun"
porcion = mensaje.substring(3, 4); // porcion = "a"
```

FUNCIONES PARA CADENAS DE TEXTO

- Si se indica un final más pequeño que el inicio, JavaScript los considera de forma inversa, ya que automáticamente asigna el valor más pequeño al inicio y el más grande al final:

```
let mensaje = "Hola Mundo";
let porcion = mensaje.substring(5, 0); // porcion = "Hola "
porcion = mensaje.substring(0, 5); // porcion = "Hola "
```

- **split(separador)**, convierte una cadena de texto en un array de cadenas de texto. La función parte la cadena de texto determinando sus trozos a partir del carácter separador indicado:

```
let mensaje = "Hola Mundo, soy una cadena de texto!";
let palabras = mensaje.split(" ");
// palabras = ["Hola", "Mundo,", "soy", "una", "cadena", "de", "texto!"];
let palabras = mensaje.Split();
//palabras[“Hola Mundo, soy una cadena de texto!”]
```

- Con esta función se pueden extraer fácilmente las letras que forman una palabra:

```
let palabra = "Hola";
let letras = palabra.split(""); // letras = ["H", "o", "l", "a"]
```

FUNCIONES PARA ARRAYS

- A continuación se muestran algunas de las funciones más útiles para el manejo de arrays:
- **length**, calcula el número de elementos de un array

```
let vocales = ["a", "e", "i", "o", "u"];
let numeroVocales = vocales.length; // numeroVocales = 5
```

- **concat(),** se emplea para concatenar los elementos de varios arrays

```
let array1 = [1, 2, 3];
array2 = array1.concat(4, 5, 6); // array2 = [1, 2, 3, 4, 5, 6]
array3 = array1.concat([4, 5, 6]); // array3 = [1, 2, 3, 4, 5, 6]
```

- **join(separador)**, une todos los elementos de un array para formar una cadena de texto. Para unir los elementos se utiliza el carácter separador indicado (si no hay usa comas):

```
let array = ["hola", "mundo"];
let mensaje = array.join(""); // mensaje = "holamundo"
mensaje = array.join(" "); // mensaje = "hola mundo“
mensaje = array.join();//mensaje = “hola, mundo”
```

- **pop()**, elimina el último elemento del array y lo devuelve. El array original se modifica y su longitud disminuye en 1 elemento.

```
let array = [1, 2, 3];
let ultimo = array.pop();
// ahora array = [1, 2], ultimo = 3
```

FUNCIONES PARA ARRAYS

- **push()**, añade un elemento al final del array. El array original se modifica y aumenta su longitud en 1 elemento (es posible añadir más de un elemento a la vez):

```
let array = [1, 2, 3];
array.push(4);
// ahora array = [1, 2, 3, 4]
```

- **shift()**, elimina el primer elemento del array y lo devuelve. El array original se ve modificado y su longitud disminuida en 1 elemento:

```
let array = [1, 2, 3];
let primero = array.shift();
// ahora array = [2, 3], primero = 1
```

- **unshift()**, añade un elemento al principio del array. El array original se modifica y aumenta su longitud en 1 elemento (es posible añadir más de un elemento a la vez):

```
let array = [1, 2, 3];
array.unshift(0);
// ahora array = [0, 1, 2, 3]
```

- **reverse()**, modifica un array colocando sus elementos en el orden inverso a su posición original:

```
let array = [1, 2, 3];
array.reverse();
// ahora array = [3, 2, 1]
```

FUNCIONES PARA NÚMEROS

- **NaN**, (del inglés, *"Not a Number"*) JavaScript emplea el valor NaN para indicar un valor numérico no definido (por ejemplo, la división 0/0):

```
let numero1 = 0;
let numero2 = 0;
alert(numero1/numero2); // se muestra el valor NaN
```

- **isNaN()**, permite proteger a la aplicación de posibles valores numéricos no definidos:

```
let numero1 = 0;
let numero2 = 0;
if(isNaN(numero1/numero2)) {
  alert("La división no está definida para los números indicados");
}
else {
  alert("La división es igual a => " + numero1/numero2);
}
```

- **Infinity**, hace referencia a un valor numérico infinito y positivo (también existe el valor **-Infinity** para los infinitos negativos):

```
let numero1 = 10;
let numero2 = 0;
alert(numero1/numero2); // se muestra el valor Infinity
```

FUNCIONES PARA NÚMEROS

- **toFixed(digitos)**, devuelve el número original con tantos decimales como los indicados por el parámetro dígitos y realiza los redondeos necesarios. Se trata de una función muy útil, por ejemplo, para mostrar precios.

```
let numero1 = 4564.34567;
numero1.toFixed(2); // 4564.35
numero1.toFixed(6); // 4564.345670
numero1.toFixed(); // 4564
```

FUNCIONES DE CONVERSIÓN DE TIPOS

La mayoría de las veces, los operadores y funciones convierten automáticamente los valores que se les pasan al tipo correcto. Esto es llamado "**conversión de tipo**".
Por ejemplo, *alert()* convierte automáticamente cualquier valor a *string* para mostrarlo. Las operaciones matemáticas convierten los valores a números.
También **hay casos donde necesitamos convertir de manera explícita** un valor al tipo esperado.

```
let a = "12"; // es string
let b = "23.01"; // es string
let c = 50; // es number

console.log ( c + c ); // 100
console.log ( c + a ); // 5012
console.log ( c + parseInt(a) ); // 62
console.log ( a + b ); // 1223.01
console.log ( c + parseFloat(b) ); // 73.01
c.toString(); // "50", es string
```

FUNCIONES DE CONVERSIÓN DE TIPOS

- **toString()**

La conversión a string ocurre cuando necesitamos la representación en forma de texto de un valor.

```
let value = true;
alert(typeof value); // boolean

value = String(value); // ahora value es el string "true"
alert(typeof value); // string
</script>
```

FUNCIONES DE CONVERSIÓN DE TIPOS

- **toNumber()**

La conversión numérica ocurre automáticamente en funciones matemáticas y expresiones. Por ejemplo, cuando se dividen valores no numéricos usando `/`:

```
alert( "6" / "2" ); // 3, los strings son convertidos a números
```

Podemos usar la función **Number(value)** para convertir de forma explícita un valor a un número:

```
let str = "123";
alert(typeof str); // string
let num = Number(str); // se convierte en 123
alert(typeof num); // number
```

La **conversión explícita** es requerida **cuando leemos** un valor **desde una fuente basada en texto**, como lo son los campos de texto en los formularios, **pero** que **esperamos que contengan un valor numérico**.

FUNCIONES DE CONVERSIÓN DE TIPOS

Si el string no es un número válido, el resultado de la conversión será *NaN*. Por ejemplo:

```
let age = Number("un texto arbitrario en vez de un número");
alert(age); // NaN, conversión fallida
```

Reglas de conversión numérica:

Valor	Se convierte en…
undefined	NaN
null	0
true and false	1 y 0
string	Se eliminan los espacios (incluye espacios, tabs \t, saltos de línea \n, etc.) al inicio y final del texto. Si el string resultante es vacío, el resultado es 0, en caso contrario el número es “leído” del string. Un error devuelve NaN.

FUNCIONES DE CONVERSIÓN DE TIPOS

- **Ejemplos**:

```
alert( Number(" 123 ") ); // 123
alert( Number("123z") ); // NaN (error al leer un número en "z")
alert( Number(true) ); // 1
alert( Number(false) ); // 0
```

- *null* y *undefined* se comportan de distinta manera aquí:
  - null se convierte en 0, mientras que
  - undefined se convierte en NaN.

- **CUIDADO**: casi todas las operaciones matemáticas convierten valores a números. Una **excepción** notable **es la suma +.** Si uno de los valores sumados es un string, el otro valor es convertido a string. Luego, los concatena (une):

```
alert( 1 + '2' ); // '12' (string a la derecha)
alert( '1' + 2 ); // '12' (string a la izquierda)
```

Esto ocurre solo si uno de los argumentos es un *string*, en caso contrario los valores son convertidos a número.

FUNCIONES DE CONVERSIÓN DE TIPOS

- **ToBoolean()**:

La conversión a boolean es la más simple. Ocurre en operaciones lógicas, pero también puede realizarse de forma explícita llamando a la función **Boolean(value)**.

- Las reglas de conversión:
  - los valores que son intuitivamente “vacíos”, como 0, "", null, undefined, y NaN, se convierten en **false**.
  - Otros valores se convierten en **true**.
- Por ejemplo:

```
alert( Boolean(1) ); // true
alert( Boolean(0) ); // false
alert( Boolean("hola") ); // true
alert( Boolean("") ); // false
```

- **CUIDADO**: el string con un cero "0" es true. En JavaScript, un string no vacío es siempre true.

```
alert( Boolean("0") ); // true
alert( Boolean(" ") ); // sólo espacios, también true
alert( Boolean("") ); // vacío, false
```

FUNCIONES DE CONVERSIÓN DE TIPOS

- **RESUMEN**
- Las tres conversiones de tipo más usadas son a *string*, a *number* y a *boolean*.
  - *ToString:* ocurre cuando se muestra algo. Se puede realizar con **String(value)**. La conversión a string es usualmente obvia para los valores primitivos.
  - *ToNumber:* ocurre en operaciones matemáticas. Se puede realizar con **Number(value)**.
  - *ToBoolean:* ocurre en operaciones lógicas. Se puede realizar con **Boolean(value)**.

- Las excepciones más notables donde tener cuidado son:
  - undefined es NaN como número, no 0.
  - "0" y textos que solo contienen espacios como “ “ son true como boolean.
  - Los objetos no son cubiertos aquí.

CLASES PREDEFINIDAS

- JavaScript dispone de varias clases predefinidas para acceder a muchas de las funciones normales de cualquier lenguaje, como puede ser el manejo de vectores o el de fechas.
- Date()
  - Esta clase nos permitirá manejar fechas y horas. Se invoca así:

```
//creación de un objeto de la clase Date
fecha = new Date();
fecha = new Date(año, mes, dia);
fecha = new Date(año, mes, dia, hora, minuto, segundo);
```

- El objeto Date dispone, entre otros, de los siguientes métodos:
  - **getYear()**: el año de la fecha. Devuelve un número de 4 dígitos (2 entre 1900 – 1999).
  - **getFullYear():** la misma función, pero siempre devuelve números con todos sus dígitos.
  - **gtMonth():** obtiene el mes de la fecha.
  - **getDate():** obtiene el día de la fecha.
  - **getDay():** devuelve el día de la semana de la fecha en forma de número (0 domingo, al 6 sábado)
  - **getHours():** devuelve la hora actual.
  - **getMinutes():** devuelve los minutos actuales.
  - **getSeconds():** devuelve los segundos actuales.

CLASES PREDEFINIDAS

- Ejemplo uso clase Date():

```
function mostrarFechaHora() {
  let fecha;
  fecha=new Date();
  alert(`Hoy es
${fecha.getDate()}/${fecha.getMonth()+1}/${fecha.getFullYear()}`);

  alert(`Es la hora
${fecha.getHours()}:${fecha.getMinutes()}:${fecha.getSeconds()}`);
}
//Llamada a la función
mostrarFechaHora();
```

CLASES PREDEFINIDAS

- Math()
  - Esta clase es un contenedor que tiene diversas constantes (como Math.E y Math.PI) y los siguientes métodos matemáticos:

Método	Descripción	Expresión de ejemplo	Resultado del ejemplo
abs	Valor absoluto	Math.abs(-2)	2
sin, cos, tan	Funciones trigonométricas, reciben el argumento en radianes	Math.cos(Math.PI)	-1
asin, acos, atan	Funciones trigonométricas inversas	Math.asin(1)	1.57
exp, log	Exponenciación y logaritmo, base E	Math.log(Math.E)	1
ceil	Devuelve el entero más pequeño mayor o igual al argumento	Math.ceil(-2.7)	-2
floor	Devuelve el entero más grande menor o igual al argumento	Math.floor(-2.7)	-3
round	Devuelve el entero más cercano o igual al argumento	Math.round(-2.7)	-3
min, max	Devuelve el menor (o mayor) de sus dos argumentos	Math.min(2,4)	2
pow	Exponenciación, siendo el primer argumento la base y el segundo el exponente	Math.pow(2,3)	8
sqrt	Raíz cuadrada	Math.sqrt(25)	5
random	Genera un valor aleatorio comprendido entre 0 y 1.	Math.random()	Ej. 0.7345

CLASES PREDEFINIDAS

- Ejemplo clase Math():

```
let selec = prompt('Ingrese un valor entre 1 y 10','');
selec = Number(selec);

/*random genera aleatorio entre 0 y 1, excluyendo el 1.
con floor garantizo entre 0 y 9. como pido entre 1 y 10, sumo 1*/
let num = Math.floor(Math.random()*10)+1;
console.log(`random = ${num}`);

if (num == selec)
  alert(`Ganó el número que se sorteó. Es el: ${num}`);
else
  alert(`Lo siento. Se sorteó el valor: +num+', y usted eligió: ${selec}`);
```