UD 2. Introducción a JavaScript

1. Fundamentos del lenguaje JavaScript
    - 1.1 Entorno de ejecución de JavaScript
    - 1.2 Variables
    - 1.3 Operadores: aclaraciones
    - 1.4 Tipos de datos y conversiones
    - 1.5 strings
    - 1.6 Estructuras de control y bucles
    - 1.7 Interacción básica

2. Funciones
    - 2.1 Parámetros y variables locales
    - 2.2 Funciones como valor
    - 2.3 Funciones anónimas
    - 2.4 Funciones flecha
    - 2.5 Parámetros y sintaxis spread
    - 2.6 Ámbitos y closures

3. Objetos y prototipos
    - 3.1 Objetos: principios básicos
    - 3.2 Asignación desestructurante
    - 3.3 Referencia y copia
    - 3.4 Objetos predefinidos y métodos de primitivas
    - 3.5 Métodos de arrays
    - 3.6 this
    - 3.7 Prototipos y herencia
        - 3.7.1 Funciones constructoras
        - 3.7.2 Objetos enlazados
    - 3.8 Prototipos nativos
    - 3.9 Clases
    - 3.10 JSON

4. Módulos
    4.1 Tipos de módulos
    4.2 import y export
    4.3 Módulos en HTML
    4.4 Librerías externas: CDN
5. Programación funcional
    5.1 Fundamentos: la función filter
    5.2 find
    5.3 forEach
    5.4 map
    5.5 reduce
6. JavaScript moderno
    6.1 Modo estricto
    6.2 ¿Clases u objetos?
    6.3 Manejo de errores
    6.4 Evolución y futuro

1. Fundamentos del lenguaje JavaScript

1. Fundamentos del lenguaje JavaScript

- Lenguaje de gran complejidad. No se van a explicar todas sus características, pero haremos hincapié en las más importantes.

- La página web javascript.info es un gran libro que se mantiene al día con el estándar definido en ECMAScript (que establece el estándar de JavaScript, así como sus nuevas funciones).

- A día de hoy, ECMAScript2025 es el último estándar. Actualmente cambia cada año.

1.1 Entorno de ejecución de JavaScript

- Puede ser usado en el navegador, y ejecutado en el cliente y en un servidor [NodeJS]. El código puede ir definido de las siguientes formas:
  - En una página HTML. Debe ir entre etiquetas **<script>** [no es necesario indicar el atributo type="script"].
  - En un archivo .js independiente. Dicho archivo puede ser ejecutado en NodeJS y cargado por una página HTML mediante **<script src="RUTA_ARCHIVO.js">.**

1.2 Variables

- Los nombres de variables en JS deben contener únicamente letras, dígitos numéricos o los caracteres **$** y _
- El primer carácter **no puede ser un dígito**.
- Palabras reservadas no usables como nombres de variables: **function**, **let**, **for**, **if**, etc.

1.2 Variables

- Existen **tres maneras de declarar** variables en JS:

1. Mediante **var**. Queda definida al ámbito de la función en la que esté declarada. Si no está declarada dentro de una función, la variable quedará definida en el objeto global.
2. Mediante **let**. Es la forma recomendada. Queda definida en el ámbito del bloque de código en el que esté declarada. **Bloque** es un trozo de código limitado por **{ }**. Si se declara fuera de cualquier bloque, quedará definida en el **objeto global** o en el **módulo** donde esté definida.
3. Mediante **const**. Queda definida de la misma manera que let, pero su valor no puede ser reasignado

Módulo: archivo JavaScript independiente que agrupa código (funciones, clases, variables) para ser reutilizado y compartido en otras partes de una aplicación

1.2 Variables

JS busca las variables definidas a través del llamado ***ámbito léxico*** o ***lexical scope***.

Cuando se hace referencia a una variable en el interior de una función [en el caso de var] o un bloque { } [en el caso de let], JS busca primero si existe una ***variable local*** *o un* ***parámetro*** *con el identificador de la variable buscada:*

- *si lo encuentra, lo utiliza;*
- *si no, busca en el siguiente nivel de función [o bloque { }], y así sucesivamente, hasta que llega a la raíz,* ***que es el objeto global*** *[programa convencional]* ***o el módulo*** *[si el código está definido en un módulo].*

1. 2 Variables

En JS moderno se tiende a usar sobre todo **let** para definir variables

Ejemplo1 – declaración de variables

- Siempre hay que minimizar el uso de variables globales.
- Las **variables globales** son accesibles, a no ser que se defina una **variable local** con el mismo nombre.
- Las **variables locales enmascaran a las globales**

Ejemplo 2 – variables locales – globales -enmascaramiento

Utiliza let SIEMPRE que vayas a usar una variable en un BUCLE, tanto si es la variable que vas a iterar como si es una variable auxiliar que vas a declarar de manera local en dicho bucle.

1.2 Variables

Declara las variables con **const** si es necesario, **let** (preferible a var), o **var** antes de usarlas.

Si no, JS creará una **variable automáticamente** en el objeto global, con consecuencias inesperadas.

Otra solución: **usar el modo estricto**, poniendo **"use strict"** al principio del script → así se genera error si la variable no está declarada.

1.3 Operadores: aclaraciones

El operador suma, **+**, se usa también para **concatenación** de **strings**. Si uno de los operandos es una cadena se realiza conversión:

```
1  console.log('2' + 5); // 25
2  console.log(2 + '7'); // 27
```

Para usar el operador suma hay que convertir las cadenas a números previamente.

1.3 Operadores: aclaraciones

Esto no ocurre con el resto de operadores

```
1  console.log('2' - 5); // -3
2  console.log('4' / '2'); // 2
```

Los operadores también permiten realizar modificaciones sobre la misma variable

```
1 let a = 54;
2 a += 4; // Es lo mismo que (a = a + 4)
3 a /= 2; // Es lo mismo que (a = a / 2)
4 console.log(a); // 29
```

1.4 Tipos de datos y conversiones

Los **tipos de datos** son:

- Tipos de datos **primitivos**:
  - **number**. Para números enteros o decimales
  - **bigint**. Para números muy grandes
  - **string**. Para cadenas de caracteres
  - **boolean**. Verdadero [true] o falso [false]
  - **null**. Valor nulo
  - **undefined**. Para variables declaradas no inicializadas [que no han recibido ningún valor]
  - **symbol**. Es un tipo de datos para crear identificadores únicos e inmutables, que se suelen utilizar para definir claves de objetos
- Tipos de datos **compuestos**:
  - **object**. Para objetos [incluidos arrays y funciones]

1.4 Tipos de datos y conversiones

El tipado dinámico:

- Las variables pueden albergar cualquier tipo de datos.
- Problemas en tiempo de ejecución. Al efectuar operaciones es posible que se produzcan conversiones de tipos:

```
let edad = prompt('¿Cuál es tu edad?'); // El usuario introduce 27. Pero "prompt" siempre devuelve un string, así que "edad" = '27'
let edadMasDiez = edad + 10; // '27' + 10 = '2710'
console.log(edadMasDiez); // 2710 en lugar de 37
// Se ha convertido el tipo de 10 a '10'. El operador '+' se ha convertido en un operador de concatenación de cadenas.
```

1.4 Tipos de datos y conversiones

Funciones y expresiones matemáticas realizan conversiones a número [con la **excepción del operador +**, que se convierte en el operador **concatenar**]. Los casos especiales son:

Valor	Conversión a número
undefined	**NaN**
null	**0**
true/false	**1 / 0**
string	Si es una cadena vacía, **0**; si se corresponde con un nº, **el nº en cuestión**; si no corresponde a un nº válido, **NaN**

El valor **NaN** es un valor numérico que significa **Not a Number**

1.4 Tipos de datos y conversiones

Para convertir a número podemos utilizar varios métodos

```
1  parseInt('3'); // 3
2  parseFloat('3.856'); // 3.856
3  Number('3'); // 3
4  +'3'; // 3
5
6  console.log(+'3' + 5); // 8
7
8  // Ejemplo anterior ejecutado de manera correcta
9  let edad = prompt('¿Cuál es tu edad?'); // El usuario introduce 27
10 let edadMasDiez = Number(edad) + 10; // 27 + 10 = 37. Convertimos la cadena '27' a númer
11 console.log(edadMasDiez); // 37
```

1.4 Tipos de datos y conversiones

Es posible comprobar si una cadena de texto no corresponde a un nº mediante la **función isNaN** [is not a number]

```
1 isNaN('5rt'); // true (No es un número válido)
2 isNaN('654'); // false
3 isNaN('285.32'); // false
4 isNaN('2e16'); // false
```

Conversiones a string automáticas. P.ej. La función **alert** puede tomar como parámetro un número que convierte automáticamente a string. Conversión explícita mediante función **String(valor)**.

Las **conversiones a boolean** siguen las siguientes reglas:

Valor	Conversión a boolean
0, null, undefined, NaN, “”	false
Otro valor [incluida la cadena “0”	true

1.4 Tipos de datos y conversiones

Los operadores de **comparación de igualdad** o "**distinto de**" tienen **dos versiones**:

- **Con comprobación de tipo: ===, !==**
- **Sin comprobación de tipo: ==, !=, *conversión de tipo automática.***

```
1  let (0 == false); // true
2  let ('' == false); // true
3
4  let (0 === false); // false (distinto tipo)
5  let ('' === false); // false (distinto tipo)
```

null y **undefined** son **iguales ==** [igualdad no estricta] **entre sí y distintos a cualquier otro valor.**

1.5 strings

Las cadenas de texto o **strings** pueden definirse mediante **comillas dobles** “ o **sencillas** ‘ .

Si se desea usar **comillas dentro de una cadena**, habrá que emplear un **entrecomillado distinto** .
Por ejemplo:

```
1  let cad = "Texto con 'comillas' dentro";
```

Existe una tercera manera de definir una cadena de texto, mediante el **acento grave [backtick]** `. Este modo tiene las siguientes **ventajas**:

- Permite crear **strings multilínea**
- Permite **incorporar expresiones** usando la estructura **${ }**.

Ejemplo 3 -- strings

1.6 Estructuras de control y bucles

Las estructuras clásicas de programación:

- **Condicionales [if, switch]** y **bucles [for, while]** son **ya conocidas** de otros lenguajes:
- Sentencia if (javascript.info)
- Sentencia Switch
- Bucles while y for

1.6 Estructuras de control y bucles

- **Operador ternario, ?**: Permite evaluar una condición de manera más concisa.
- Se usa para asignar un valor a una variable en función de una condición

```
let mayorDe30;
let edad = prompt('¿Cuál es tu edad?');
if (edad >= 30) {
    mayorDe30 = true;
} else {
    mayorDe30 = false;
}
```

El condicional puede reescribirse así:

```
let mayorDe30 = (edad >= 30) ? true : false;
// Se evalúa la condición (edad >= 30). Si es verdadera, se asigna a la variable "mayorDe30" el valor indicado tras '?' (true).
// Si es falsa, se asigna el valor indicado tras ':' (false).

let colores = ['azul', 'rojo', 'blanco'];
```

1.6 Estructuras de control y bucles

Bucle for of, que se usa para **iterar sobre arrays**:

```
1  // Bucle 'for' tradiciional
2  for (let i = 0; i < colores.length; i++) {
3      console.log(colores[i]);
4  }
5
6  // Bucle 'for...of'
7  for (let color of colores) {
8      // La variable 'color' almacena el elemento del array que se está iterando
9      // Equivalencia: color = colores[i]
10     console.log(color);
11 }
```

1.6 Estructuras de control y bucles

La estructura **for in**, aunque parecida a **for of**, se usa para **iterar sobre las propiedades de un objeto**.

Como en JS los **arrays son objetos**, también funcionará, pero **no es recomendable** usarla pues es más lenta y accede también a otras propiedades que no son los elementos del array.

1.7 Interacción básica

alert, **prompt** y **confirm** son parte de la **API del navegador** [dentro del **objeto window**]. Por tanto, **no** están **disponibles en NodeJS**

Las tres funciones básicas de interacción con el usuario son: **alert, prompt y confirm.**

- La función **alert** muestra un mensaje en un cuadro de diálogo mediante alert("MENSAJE");
- La función **prompt** muestra un cuadro de diálogo con un texto para que el usuario introduzca un valor, y devuelve dicho valor. La función tiene dos parámetros (el segundo es opcional):

1.7 Interacción básica

```
1  let respuesta = prompt("Texto para mostrar", "Valor por defecto del cuadro de texto (opcional)");
2  // respuesta almacena el texto escrito por el usuario en el cuadro de diálogo
3  // OJO: siempre almacenará un string. Puede que sea neceario convertirlo a otro tipo de datos.
4
5  let acepta = confirm("¿Estás seguro de que quieres continuar?");
6  // acepta almacena true si el usuario pulsa "Aceptar" y false si pulsa "Cancelar"
```

prompt devuelve un **string** si el usuario escribe algo y **null** si el usuario cancela la acción

1.7 Interacción básica

La función **confirm** muestra un cuadro de diálogo con un texto y dos botones: **devuelve true** si el usuario pulsa en **Aceptar** y **false** si pulsa en **Cancelar**.

```
1  let acepta = confirm("¿Estás seguro de que quieres continuar?");
2  // acepta almacena true si el usuario pulsa "Aceptar"
3  // y false si pulsa "Cancelar"
```

Por último, podemos citar la función **console.log("MENSAJE")**;

Ésta escribe un mensaje en la consola. Esta función sí que está disponible tanto en el navegador como en NodeJS.