UF1305. PROGRAMACIÓN CON LENGUAJES DE GUIÓN EN PÁGINAS WEB.

UF1842. DESARROLLO Y REUTILIZACIÓN DE COMPONENTES SOFTWARE Y MULTIMEDIA MEDIANTE LENGUAJES DE GUION.

JAVASCRIPT.

- Se crea en 1995 de la mano de Netscape.

ECMA

- European Computer Manufacturers Association.
- Se encarga de desarrollarlo, estandarizarlo y actualizarlo.
- Primer estándar se denomina EcmaScript o ES.

Versiones.

- ES1 (junio 1997) hasta la ES15 (2024).
- Desde el 2016 se denominan ES2016, ES2017, ES2021, … , ES2024, etc.

Concepto:

- Lenguaje de programación por secuencias de comandos (script) interpretado.
- Orientado a objetos, basado en prototipos, dinámico, imperativo y débilmente tipado.
- Es un lenguaje orientado a eventos**:**
    - Software que está a la espera de que se produzca una acción o eventos.

Objeto prototípico:

- Equivale a una clase en POO.
- Plantilla a partir de la cual se obtiene el conjunto inicial de propiedades y funcionalidad de un nuevo objeto o instancia.

Objeto Intrínseco:

- Objetos que siempre están presentes en el inicio de la ejecución de JavaScript como Number, String, Object, Function, Boolean, etc.
- Existe un objeto intrínseco denominado “Object”, que permite la creación de un objeto.

Instancia:

- Equivale a un objeto en POO.
- Se crean a partir de objeto prototípicos.
- Contiene los miembros de un objeto prototípico (propiedades o atributos y métodos o funciones).

Constructor:

- Función o método que sirve para crear e inicializar un objeto o instancia.

Lenguajes POO (Java, C++, C#, Visual Basic)	**JavaScript o Kotlin**
Clase	Objeto Prototípico
Objeto o instancia	Instancia
Método	Función

Script.

- Fichero de texto plano que permite describir ordenes (secuencias de comandos) que un intérprete ejecutara en tiempo real.
- No necesita ser compilado.

JavaScript ámbitos de uso.

- JavaScript hoy es un lenguaje con amplios usos. A veces las soluciones no son demasiado efectivas y la gran crítica que tiene es el bajo rendimiento al momento de solucionar ciertas tareas.
- Sin embargo, con aras de responder a la pregunta, JavaScript puede ser usado para:

    - **Aplicaciones móviles**: React Native, NativeScript, Ionic o Cordova/PhoneGap
    - **Aplicaciones web**: NodeJS (Server Side) y VueJS, Angular o React (Client Side).
    - **Bases de datos**: MongoDB, PouchDB, Firebase (Baas).
    - **Videojuegos**: PhaserJS.
    - **Software de escritorio**: ElectronJS, React Desktop.

Uso lado cliente:

- Implementación de funciones complejas en páginas web, mejoras en la interfaz de usuario y creación de páginas web dinámicas.
- Interpretado por un navegador.
- Los navegadores interpretan el código JavaScript integrado en páginas web.
- Para poder hacerse provee a JavaScript de una implementación del DOM.

Lado servidor.

- Programación operaciones complejas.
- Procesamiento de datos enviados por usuarios.
- Programación con Node.js

OTROS CONCEPTOS RELACIONADOS:

- **Librerías (Bibliotecas).**
    - Archivo o conjunto de archivos que se utilizan para facilitar la programación.
    - Incluyen código que alguien ha realizado para poder reutilizarlo dentro de otros proyectos.
    - Para JavaScript:
        - JQuery, Mootools y Prototype.
        - Facilitar el uso de JavaScript para interactuar con la web.
- **Frameworks o Entornos de Trabajo.**
    - Estructura previa que se puede aprovechar para desarrollar un proyecto.
    - Es una especie de **esquema conceptual** que simplifica la elaboración de una tarea.
    - Puede incluir soporte de programas, bibliotecas, y un lenguaje interpretado, entre otras herramientas, para así ayudar a desarrollar y unir los diferentes componentes de un proyecto.
    - Entornos de trabajo basados en JavaScript:
        - AngularJS, React y Backbone.js.
- **API's**
    - **A**pplication **P**rogramming **I**nterfaces - Interfaz de Programación de Aplicaciones.
    - Conjunto de definiciones y protocolos que especifican **cómo las aplicaciones de software deben comunicarse o interactuar entre sí para realizar** una o varias funciones.
    - En JavaScript:
        - Bloques de código listos para usar que permiten desarrollar programas con JavaScript para aumentar su funcionalidad.
        - API Navegador:
            - API del DOM, API de geolocalización, API Canvas y WebGL, API de audio y video.

- API Terceros:
    - API Twiter, API Google Maps o API Open Street Maps.

LENGUAJE DE PROGRAMACIÓN.

Concepto:

- Software que sirve para crear otros programas.
- Los lenguajes de programación son la base para construir todos los programas y aplicaciones digitales que se utilizan en el día a día.

Tipos:

- **Lenguaje de programación de bajo nivel.**
    - Son lenguajes totalmente orientados a la máquina.
    - Este lenguaje sirve de interfaz y crea un vínculo inseparable entre el hardware y el software.
    - Además, ejerce un control directo sobre el equipo y su estructura física. Para aplicarlo adecuadamente es necesario que el programador conozca sólidamente el hardware. Éste se subdivide en dos tipos:
    - **Lenguaje máquina**
        - Es el más primitivo de los lenguajes y es una colección de dígitos binarios o bits (0 y 1) que la computadora lee e interpreta.
        - Ejemplo: **10110000 01100001**
    - **Lenguaje ensamblador**
        - El lenguaje ensamblador es el primer intento de sustitución del lenguaje de máquina por uno más cercano al utilizado por los humanos.
        - Un programa escrito en este lenguaje es almacenado como texto (tal como programas de alto nivel) y consiste en una serie de instrucciones que corresponden al flujo de órdenes ejecutables por un microprocesador.
        - Sin embargo, dichas máquinas no comprenden el lenguaje ensamblador, por lo que se debe convertir a lenguaje máquina mediante un programa llamado Ensamblador.
        - Este genera códigos compactos, rápidos y eficientes creados por el programador que tiene el control total de la máquina.
- **Lenguaje de programación de alto nivel.**
    - Tienen como objetivo facilitar el trabajo del programador, ya que utilizan unas instrucciones más fáciles de entender.
    - Además, el lenguaje de alto nivel permite escribir códigos mediante idiomas que conocemos (español, inglés, etc.) y luego, para ser ejecutados, se traduce al lenguaje de máquina mediante traductores o compiladores.
- **Traductor**
    - Traducen programas escritos en un lenguaje de programación al lenguaje máquina de la computadora y a medida que va siendo traducida, se ejecuta.
- **Compilador**
    - Permite traducir todo un programa de una sola vez, haciendo una ejecución más rápida y puede almacenarse para usarse luego sin volver a hacer la traducción.

PROGRAMACIÓN ORIENTADA A OBJETOS (POO)

- Es un paradigma de la programación, como la programación estructurada o la programación modular.
- Consiste en una forma de programar específica, donde se organiza el código en unidades denominadas clases, de las cuales se crean objetos que se relacionan entre sí para conseguir los objetivos de las aplicaciones.

Características de la POO:

- Existe un acuerdo acerca de qué características contempla la "orientación a objetos". Las **características** siguientes son las más importantes:
- Abstracción:
    - Proceso de interpretación y diseño que implica reconocer y enfocarse en las características importantes de una situación u objeto, y filtrar o ignorar todas las particularidades no esenciales.
    - Hay que dejar a un lado los detalles de un objeto y definir las características específicas de éste, aquellas que lo distingan de los demás tipos de objetos. Hay que centrarse en lo que es y lo que hace un objeto, antes de decidir cómo debería ser implementado.
- Encapsulamiento:
    - Significa reunir todos los elementos que pueden considerarse pertenecientes a una misma entidad, al mismo nivel de abstracción.
- Polimorfismo:
    - Comportamientos diferentes, asociados a objetos distintos, pueden compartir el mismo nombre; al llamarlos por ese nombre se utilizará el comportamiento correspondiente al objeto que se esté usando.
- Herencia:
    - Las clases se relacionan entre sí formando una jerarquía de clasificación. Los objetos heredan las propiedades y el comportamiento de todas las clases a las que pertenecen. La herencia permite a los objetos ser definidos y creados como tipos especializados de objetos preexistentes. Cuando un objeto hereda de más de una clase, se dice que hay herencia múltiple.
- Modularidad:
    - Propiedad que permite subdividir una aplicación en partes más pequeñas (llamadas módulos), cada una de las cuales debe ser tan independiente como sea posible de la aplicación en sí y de las restantes partes.
- Principio de ocultación:
    - Cada objeto está aislado del exterior y muestra una "interfaz" a otros objetos que específica cómo pueden interactuar con los objetos de la clase. El aislamiento protege a las propiedades de un objeto contra su modificación por quien no tenga derecho a acceder a ellas; solamente los propios métodos internos del objeto pueden acceder a su estado. Esto asegura que otros objetos no puedan cambiar el estado interno de un objeto de manera inesperada, eliminando efectos secundarios e interacciones inesperadas.
- Recolección de basura:

    La recolección de basura (*garbage collection*) es la técnica por la cual el entorno de objetos se encarga de destruir automáticamente, y por tanto desvincular la memoria asociada, los objetos que hayan quedado sin ninguna referencia a ellos. Esto significa que el programador no debe preocuparse por la asignación o liberación de memoria, ya que el entorno la asignará al crear un nuevo objeto y la liberará cuando nadie lo esté usando.

Conceptos:

Clase:

- Equivalente a un objeto prototípico en JavaScript.
- Una clase es una abstracción de un objeto.
- Es una especie de molde o plantilla en la que se definen las características (atributos) y el comportamiento (métodos), predeterminado de un tipo de objetos.
- A partir de ese molde o plantilla se pueden crear objetos fácilmente.
- Ejemplos:
    - **Clase Coche**: (marca, modelo, cilindrada, color, precio, acelerar (), frenar (), cambiarVelocidad (),)
    - **Clase Círculo**: (radio, calcularArea(), calcularDiametro(),calcularCircunferencia()).

Objeto:

- En JavaScript se denominan instancias.
- Instancia de una clase.
- Entidad provista de un conjunto de propiedades o atributos (datos) y de comportamiento o funcionalidad (métodos), los mismos que consecuentemente reaccionan a eventos. Se corresponden con los objetos reales del mundo que nos rodea, o con objetos internos de sistema o programa informático.
- Al método de crear nuevos objetos a partir de una clase se le conoce como instanciación.
- Ejemplos:
    - **Objeto1 Coche1:** (Seat, Ibiza, 1800 c.c., Rojo, 12.000,00 euros, instrucciones para acelerar, instrucciones para frenar, …)
    - **Objeto2 Coche2:** (Ford, Fiesta, 1600 c.c., Azul, 10.000,00 euros, instrucciones para acelerar, instrucciones para frenar...)

Resumen:

- Una clase es una abstracción de algún hecho o ente del mundo real, con atributos que representan sus características o propiedades, y métodos que emulan su comportamiento o actividad. Todas las propiedades y métodos comunes a los objetos se encapsulan o agrupan en clases. Una clase es una plantilla, un prototipo para crear objetos; en general, se dice que cada objeto es una instancia o ejemplar de una clase.

Propiedad o atributo:

- Características de una clase u objeto.

Método:

- Conjunto de Instrucciones que permiten realizar una acción.
- Se desencadenan tras la recepción de un mensaje.
- Pueden producir cambios en las propiedades de un objeto o desencadenar otras acciones para otros objetos.

Mensaje:

- Comunicación dirigida a un objeto, que le ordena que ejecute uno de sus métodos con ciertos parámetros asociados al evento que lo generó.

Evento:

- Acción que provoca una respuesta.
- Se pueden producir por la interacción de un usuario, el sistema o un mensaje enviado por otro objeto.

Técnicas de programación:

Proceden de la programación estructurada, pero se utilizan también en la POO.

1. **Secuencia.**
    - Indica que las instrucciones del código se leerán de principio a fin desde la primera línea de código hasta la última, sin excepción.
2. **Iteración.**
    - Indica que, según cierta condición, un número de instrucciones podrían repetirse un número determinado o incluso indeterminado de veces. Las iteraciones son básicamente estructuras cíclicas (bucles), que nos permitirán repetir una cantidad de veces determinada o indeterminada unas instrucciones.
3. **Decisión.**
    - Indica que según unas ciertas condiciones dadas se ejecutarán o no un conjunto de instrucciones.

Variable.

- Espacio reservado en memoria para almacenar un dato.
- A una variable se le asigna un nombre para identificarla y un tipo para especificar qué conjunto de valores puede almacenar.
- Tipos:
    - **Local:**
        - Variables declaradas dentro de bloques de código como, por ejemplo, dentro de un método o un bucle, siendo solo accesibles para ser leídas o modificadas desde el propio bloque.
    - **Global**:
        - Variable declarada fuera de un bloque de código y a la cual pueden acceder todos los métodos, procedimientos y funciones de una aplicación.

Constante.

- Espacio reservado en memoria para almacenar un dato que no cambia, que no puede ser alterado o modificado durante la ejecución de un programa, únicamente puede ser leído.
- A una constante también se le asigna un nombre para identificarla y un tipo para especificar qué conjunto de valores puede almacenar.
- El nombre de las constantes suele escribirse en mayúsculas en la mayoría de lenguajes.

LENGUAJE JAVASCRIPT.

Características.

- **Es Case y Acent sensitive**.
    - Distinción entre mayúsculas y minúsculas, y palabras acentuadas y no.
    - Uso de caracteres Unicode.

- Numero, numero o NUMERO se consideran variables distintas.

- **Declaraciones**.
  - Son las instrucciones en JavaScript.
  - Cada declaración distinta acaba en punto y coma (;).
- **Comentarios**.
  - Comentarios de una línea:

    // Texto del comentario.
  - Comentario de varias líneas:

    /* Texto de un comentario
    de varias líneas */
- **Palabras clave**.
  - Son las palabras reservadas del lenguaje.
  - No se pueden usar para nombrar variables, constantes, funciones, objetos, etc.
  - Algunos ejemplos:
    - break, case, function, export, switch, extends, …
- **Espacio en blanco y saltos de línea.**
  - No se tienen en cuenta.

INTEGRACION DE JAVASCRIPT EN HTML

- Uso de la etiqueta HTML **<script>…</script>.**
- Permite insertar código JavaScript en una página web.
- Formas de uso:
  - **Código JavaScript incluido en la página web.**
    - Sintaxis:

      <script type = "text/javascript">
      Código JavaScript
      </script>
    - Uso habitual cuando se va a usar el script en una sola página o es poco su contenido, así se ahorran peticiones al servidor.
    - Ejemplo:

      <script type = "text/javascript">
      alert("!Hola Mundo¡");
      </script>
  - **Código JavaScript incluido en un archivo .js.**
    - Sintaxis:

      <script type = "text/javascript" src="ruta acceso/archivo.js">
      </script>
    - Ejemplo:
      - Archivo HTML:

        <script type = "text/javascript" src ="holamundo.js">
        </script>
      - Archivo JavaScript:

        alert("!Hola Mundo¡");
    - Uso habitual cuando se va a usar el script en varias páginas o el código es muy extenso.
- En caso de incluir código dentro de las etiquetas <script>, prevalece el código del archivo:

```
<script type = "text/javascript" src="ruta acceso/archivo.js">
    Código JavaScript;
</script>
<script type = "text/javascript" >
```

```
        Código JavaScript;
    </script>
```

- Se pueden cargar varios archivos a la vez con diferentes etiquetas <script>:

```
<script type = “text/javascript” src=”ruta acceso/archivo1.js”> </script>
<script type = “text/javascript” src=”ruta acceso/archivo2.js”> </script>
<script type = “text/javascript” src=”ruta acceso/archivo3.js”> </script>
```

- Ejecución de un script:
  - Al cargarse la página:
    - Una opción es incluirle el evento onLoad.
    - Podría ocurrir que se detenga la carga de la página hasta que no termine la ejecución del script.
    - No se incluye en un evento que tenga que ejecutar el usuario.
  - En respuesta a un evento:
    - Opción más común.
    - La página se carga completa y el script se ejecuta como respuesta a un evento.

<noscript>…</noscript>

- Sirve para incluir un texto de aviso si se detecta que JavaScript está deshabilitado por el usuario o el navegador no lo soporta.
- Ubicación:
  - Se puede incluir en <head> o al principio en <body>.
- Sintaxis:
  - <noscript>Mensaje</noscript>
- Ejemplos:

```
<noscript>
    <p>La página que estás viendo requiere para su funcionamiento el uso de JavaScript. Si lo has deshabilitado intencionadamente, por favor vuelve a activarlo.</p>
</noscript>
```

- Habilitar/Deshabilitar JavaScript en navegador.
  - Firefox:
    - about:config en barra de direcciones.
    - Buscar javascript.enabled.
    - Activarla (true)/Desactivarla (false).
  - Chrome y Opera:
    - Menú/Configuración/Seguridad y privacidad/Configuración del sitio/JavaScript.
    - Activar/Desactivar JavaScript.

VARIABLES.

- Contenedor para almacenar un dato.
- Una variable tiene:
  - Un nombre.
  - Una dirección de memoria.
  - Un dato o no (nulo o undefined).
- Nombre de una variable:
  - Case-Sensitive.
  - Caracteres permitidos:
    - Caracteres alfanuméricos, guion -, subrayado _ y $.
    - No pueden empezar por número.

        - No permitidos caracteres especiales como: */ @ #+;
        - No permitidas las palabras reservadas.
        - No aconsejables palabras acentuadas o con eñes.
        - Evitar espacio en blanco.
    - Usar nomenclatura camel case:
        - nombreDelCliente y no nombre del cliente.
    - También se puede unir palabras usando guiones bajos o de subrayado:
        - nombre_del_cliente
- No puede haber dos variables con el mismo nombre (excepto locales a alguna función).

CREACIÓN DE UNA VARIABLE.

Declaración de la variable:

- Consiste en definir e informar al sistema que se va a usar como variable.
- Formas de declaración:
    - **var**
        - Instrucción que permite crear variables locales y globales.
        - Como puede dar lugar a errores (asignar automáticamente el valor "undefined" a variables no inicializadas, declarar varias veces la misma variable, ...) se aconseja usar let en su lugar.
        - Sintaxis:

            var nombre de la variable;
        - Se pueden declarar varias variables con la misma instrucción.

            var nombre variable 1;
            var nombre variable 2;
            var nombre variable 3;
        - Es equivalente a:

            var nombre variable 1, nombre variable 2, nombre variable 3;

        - Ejemplos:

            var mensajeNumeroUno;
            var mensaje;
            var Mensaje;
            var MENsaje;
            var camion;
            var camión;
            var nombre, apellido1, apellido2;
            var nombreCompleto;
            var numero1, numero2, numero3, valor1;
            var suma;
            var apellidosClientes;
            variable9; //Error, al no utilizar la palabra reservada var para crear la variable.
- **let**

    - Nueva instrucción desde ES6 o ES2015.
    - Permite crear variables locales dentro de bloques de código.
    - Permite también crear variables globales.
    - Tendencia a uso actual en lugar de var.
    - La declaración con **let** afecta a su ámbito, ya que se declara la variable a nivel local.
    - Previene la sobreescritura accidental de variables y aumenta la seguridad.
        - Sintaxis:

let nombre de la variable;

- Se pueden declarar varias variables con la misma instrucción.

  let nombre variable 1;
  let nombre variable 2;
  let nombre variable 3;

- Es equivalente a:

  let nombre variable 1, nombre variable 2, nombre variable 3;

- Ejemplo:

  let suma;

- **const**
  - Permite declara un constante, cuyo valor no puede cambiar a lo largo de la ejecución del programa.
  - Permite también crear objetos con valor inalterable.
    - Sintaxis:
      - const NOMBRE DE LA CONSTANTE = valor;
    - Ejemplo:

      const PI = 3.1416;

Inicialización o asignación de valores a las variables:

- Se hace tras declarar una variable.
- Sintaxis:
  - nombre de la variable = valor;
- Ejemplos:

  - mensaje = "Hola Mundo";
  - numero1 = 28.67; // Variable de tipo numérico real.
  - numero2 = 28; // Variable de tipo numérico entero.
  - numero3 = "45.87"; // Variable de tipo texto con números.
  - nombre ="Ana";
  - apellido1 = "López";
  - letraDNI = 'L';
  - letraDNI = "L";
  - apellido2 = " ";
  - valor1 = true; // Variable de tipo booleano.
  - valor2 = false; // Variable de tipo booleano.
  - n = 5 + 60;
  - suma = numero2 + variable1; // El valor de la variable será el resultado de una operación con otras variables.
  - nombreCompleto = nombre + apellido1 + apellido2; // Concatenación de valores.
  - nombreCompleto = "Ana" + " López";

Declaración + asignación o inicialización:

- Se puede declarar un variable y a la vez asignarle un valor.
- Sintaxis:
  - var o let nombre de la variable = valor;
- Ejemplos:
  - var nombre = "Pepe"; Equivaldría a:

        - var nombre;
        - nombre = “Pepe”;
    - var edad = 20;
    - var suma = numero1 + numero2;

Declaración múltiple:

- Permite declarar varias variables y asignarle su valor en una única instrucción.
- Ejemplo:
    - var apellido1 = ”López”, apellido2 = ”García”;

Declaración implícita:

- Se crea la variable sin la palabra reservada var o let.
- Obligatorio inicializarla o darle valor.
- Sintaxis:
    - nombre de la variable = valor.
- Ejemplo:
    - suma = 45;
    - dirección = “C/ Oviedo, 20”;
    - suma; //Incorrecto por no asignar valor y no haber incluido var o let en la declaración.

Declaración repetida:

- Si se declara una variable más de una vez (explícita o implícitamente), y se le asigna un valor, prevalece el último.
- Ejemplo:
    - var numero = 5;
    - var numero = 20;
    - La variable numero vale 20, no 5, ya que el último valor sobrescribe al previo.

Ámbito de una variable (scope).

- También alcance o visibilidad.
- Zona del programa donde se define una variable y puede ser utilizada.
- Ámbitos:
    - **Local**:
        - La variable se define dentro de una función, método, bucle, etc.
        - Solo están accesibles o se puede acceder a ellas donde se han declarado, es decir, desde la estructura que las incluye.
    - **Global**:
        - Definida en cualquier lugar del programa fuera de estructuras.
        - Están accesible o se puede acceder a ellas desde cualquier lugar del programa o la página web como:
            - Cualquier función o método del script donde está la variable.
            - Otros scripts de la página.
            - Manejadores de eventos.
    - **De bloque**:
        - Solo existen en el bloque donde ha sido definidas.
        - Las variables se crean con let.
        - Podría ser obligatorio incluir la sentencia “use strict” para que los navegadores funcionen en modo estricto.

- Ejemplo:

```
"use strict";
If(true)
{
        let x= 20;
}
        console.log(x);
```

TIPOS DE DATOS.

- En JavaScript no es necesario especificar el tipo de variable o constante a crear.
- El tipo queda determinado por el valor asignado.
- Ventaja:
  - Una variable puede cambiar de tipo a lo largo de la ejecución de un programa.
- **Tipos**:
  - **Números**:
    - Solo hay un tipo numérico que engloba tanto los números enteros como los reales.
    - Ejemplos:
      - var edad = 34;
      - const PI = 3.1416;
      - var seiscientos = 6E+2;
      - const millonYMedio = 1.5E+6;
    - Notación científica:
      - 1.000.000 → $1*10^6$ → Notación científica → 1E+6
      - 0,000001 → $1*10^{-6}$ → Notación científica → 1E-6
    - Otros sistemas de representación numérica.
    - **Octal** (Base 8).
      - Usar dígitos 0 al 7
      - Se escribe un cero delante del número.
    - **Hexadecimal** (Base 16).
      - Dígitos del 0 al 9 y letras desde la a hasta la f.
      - Se escribe 0X delante del número.
    - Ejemplos:
      - Decimal: 5
      - Octal: 05
      - Hexadecimal: 0x5
  - **Lógicos**:
    - También se les denomina booleanos o boolean.
    - Sólo admiten 2 valores: true o false sin comillas
    - Se escriben en minúsculas.
    - Usados en la toma de decisiones, en condiciones.
    - Ejemplo:
      - var prestado = true;
  - **Cadenas**:
    - También se denominan strings.
    - Son conjuntos de carácter alfanuméricos.
    - Se escriben ente comillas dobles o simples.
    - Ejemplos:
      - var nombre = "Luis";
      - var archivo = "código.js";

- var dni = “1234567890L”;
- var texto = “ ”
- var letraDNI = “h”;
- var letraDNI = ‘h’;

- **Valores especiales**:
  - **undefined.**
    - Indefinido.
    - La variable está definida o declarada pero aún no tiene ningún valor.
    - Ejemplo:
      - var a;
      - a = “undefined”;
  - **null**.
    - Nulo.
    - Permite inicializar una variable de la que no se conoce el valor.
    - null es distinto de undefined.
    - Ejemplo:
      - var a = null;
  - **NaN.**
    - Not a Number.
    - Valor no numérico.
    - Se produce en operaciones aritméticas sin sentido.
      - var c = 7 * “hola”;
    - División por cero.
      - var h = 5/0;
  - **infinity.**
    - Infinito.
    - Variable con valor demasiado alto de tipo positivo.
      - var h = 8E500 * 7E34;
  - -**infinity**
    - Infinito negativo.
    - Variable con valor demasiado alto de tipo negativo.
      - var h = -8E500 * 7E34;
  - **Objetos**.

OPERADORES.

Aritméticos.

- Suma y signo positivo: +
- Resta y Signo Negativo: -
- Multiplicación: *
- División: /
- Potencia: **
  - Base**Exponente
  - Ejemplo:
    - 5 al cuadrado se escribiría como 5**2. Equivalente a 5 * 5.
- Módulo o resto: %
- Ejemplos:
  - 6/2 = 3
  - 6%2 = 0
  - 7%2 = 1

- 6%5 = 1
- 125%5 = 0
- 125/5 = 25

Concatenación.

- Operación que permite unir cadenas de caracteres ente si, variables de tipo texto o cadenas con variables.
- Se usa el operador más (+)
- Cuando se suman 2 números se produce una suma, mientras que cuando se hace lo mismo con un número que es en realidad una cadena de caracteres, se produce una concatenación y no una suma.
  - 5 + 5 = 10.
  - "5"+ 5 = 55.

Asignación (=).

- Sirven para asignar un valor a una variable, constante u otro tipo de estructuras de datos.
- Asigna el valor situado a la derecha del igual, al operando situado a la izquierda.
- Sintaxis:
  - operando = valor;
- Ejemplos:
  - nombre = "Luis";

Asignación y cálculo.

- Operadores aritméticos abreviados.
- Formados por un operador aritmético seguido inmediatamente del operador de asignación.
- El primer operando debe ser una variable que almacenara el resultado y como segundo cualquier dato (variables, número, otra operación, etc.).
- Sintaxis:
  - +=, -=, *=, /=, **=, %=
- Ejemplos:
  - a += 5; equivale a = a+5;
  - a += b; equivale a = a+b;
  - a += (5*40); equivale a = a+(5*40);
  - a -= 5; equivale a = a-5;
  - a *= 5; equivale a = a*5;
  - a /= 5; equivale a = a/5;
  - a %= 5; equivale a = a%5;
  - a **= 5; equivale a = a**5;

Incremento/Decremento.

- **Incremento (++).**
  - Suma una unidad al operando.
- Sintaxis:
  - variable ++; equivale a variable = variable + 1 o también a variable += 1;
  - Ejemplo:
    - var numero = 1;
    - numero++; equivale a numero = numero + 1; numero vale 2.
    - numero++; equivale a numero = numero + 1; numero vale 3.
- El operador se puede poner delante o detrás de la variable.
  - **Preincremento**:
    - Primero se incrementa la variable y luego se lee el valor o se hace una operación.
    - Sintaxis:
      - ++variable;
  - **Postincremento**:

        - Primero se lee el valor o se hace una operación, y luego se incrementa la variable.
        - Sintaxis:
            - variable++;
    - Ejemplos:
        - var a = 5;
        - a++; Devuelve 5 y luego pasa su valor a ser 6.
        - ++a; Devuelve 6. Primero incrementa y se muestra el valor.
- **Decremento (--).**
    - Resta una unidad al operando.
- Sintaxis:
    - variable--; equivale a variable = variable -1;
        - var numero = 10;
        - numero--; equivale a numero = numero - 1; numero vale 9.
        - numero--; equivale a numero = numero - 1; numero vale 8.
- El operador se puede poner delante o detrás de la variable.
    - **Predrecremento**:
        - Primero se decrementa la variable y luego se lee el valor o se hace una operación.
        - Sintaxis:
            - --variable;
    - **Postdecremento**:
        - Primero se lee el valor o se hace una operación, y luego se decrementa la variable.
        - Sintaxis:
            - variable--;
    - Ejemplos:
        - var a = 5;
        - a--; Devuelve 5 y luego pasa su valor a ser 4.
        - --a; Devuelve 4. Primero decrementa y se muestra el valor.

Relacionales o de comparación.

- Permite comparar los valores de 2 operandos y devuelven un valor booleano (true - false).
- Se utilizan para crear condiciones simples.
- Operadores:
    - > mayor que.
    - < menor que.
    - >= mayor o igual.
    - <= menor o igual,
    - == igual.
    - != distinto, diferente o lo contario.
    - === igualdad estricta en tipo y valor.
    - !== desigualdad estricta en tipo y valor.
- Ejemplos:

  var a = 23; a > 7; Verdadero o true.
  var b = “f”; var h = 3; b == h; Falso o false.
  var c = “Adiós”; c != “Hola”; Verdadero o true.
- Igualdad estricta (===).
    - Se compara si los datos son iguales o ambos tipos coinciden.
    - Si tipo y dato coinciden devuelven true, sino false.
    - Ejemplo:
        - var a = “1234”;
        - var b = 1234;
        - var c = 1234;
        - a===b // false.

        - a==b // true.
        - b===c // true.
- Desigualdad estricta (!==).
    - Se compara si los datos no son iguales o ambos tipos no coinciden.
    - Si tipo y dato no coinciden devuelven true, sino false.
    - Ejemplo:
        - var a = “1234”;
        - var b = 1234;
        - var c = “234”;
        - a!==b // true.
        - b!==c//true.
        - a!=b // false.

Lógicos.

- Sirve para crear condiciones compuestas.
- Devuelven verdadero o falso (true-false).
- Operadores:
    - && (Y o AND lógico).
        - Para que se cumpla la condición global y se obtenga un valor verdadero, deben cumplirse obligatoriamente todas las parciales, si no, se produce un resultado falso.
        - Ejemplo:
            - (a==5) && (c>r) && (h!=”JavaScript”)
    - || (O o OR lógico).
        - Para que se cumpla la condición global y se obtenga un valor verdadero, debe cumplirse al menos una condición parcial, si no, se produce un resultado falso.
        - Ejemplo:
            - (a==5) || (c>r) || (h!=”JavaScript”)
    - ! (No o NOT negación).
        - Invierte el valor del operando que le precede.
        - Ejemplo:
            - var x = false;
            - var y = !x;

Operador coma (,).

- Sirve para declarar varias variables a la vez.
- Separar para separar varias expresiones en un bucle for.

typeof.

- Permite conocer el tipo de dato del operando que se incluya a continuación.
- El tipo se devuelve como cadena de caracteres.
- Como son caracteres si se usa en una condición deben encerrarse entre comillas.
- Tipos devueltos:
    - number (entero o real).
    - boolean (booleano).
    - string (cadena de caracteres)
    - function (función u objeto predefinido).
    - object (Objeto).
    - undefined (operando no inicializado).
- Sintaxis:
    - typeof(operando);
    - typeof operando;
- Ejemplos:
    - typeof(a);
    - typeof a;

    - typeof(a-6);
    - typeof("hola");
    - typeof(miObjeto);
    - typeof(5.4);
    - typeof(60-43);
- Ejemplo de uso en condiciones:
    - if(typeof(a) == "number")

Operador condicional.

- Operador ternario.
- Sintaxis:
    - var resultado = operando 1? operando 2: operando 3;
    - Operando 1 es una condición.
    - Operando 2 es el valor si verdadero.
    - Operando 3 es el valor si falso.
- Ejemplo:
    - var edad =20;
    - var mayoriaEdad = (edad >=18)? "Si": "No";
    - var mayoriaEdad = (edad >=18)? 80: 60;
    - var mayoriaEdad = (edad >=18)? a*40: "Adiós";

Operadores sobre objetos.

- Usados sólo para objetos.
- Operadores:
    - Punto (.)
        - Obtener o asignar un valor a una propiedad o ejecutar un método.
        - Sintaxis:
            - objeto.propiedad;
            - objeto.metodo();
        - Ejemplo:
            - miObjeto.nombre = "Ana";
            - miObjeto.imprimir();
    - Corchetes[].
        - Permiten acceder al valor a una propiedad o un método.
        - Sintaxis:
            - Objeto["propiedad2"];
            - Objeto["método"]();
        - Ejemplo:
            - miObjeto["nombre"];
            - miObjeto["imprimir"]();
    - new.
        - Operador que sirve para crear objetos.
    - delete.
        - Permite borrar el valor de una propiedad de una instancia.
        - Sintaxis:
            - delete Objeto.propiedad;
        - Ejemplo:
            - delete miObjeto.nombre;
    - in.
        - Permite conocer si una propiedad existe o no en un objeto.
        - Sintaxis:

            - Propiedad in Objeto;
        - Ejemplo:
            - edad in miObjeto;
    - instanceof:
        - Indica si una instancia especificada corresponde a determinada clase.
        - Sintaxis:
            - Objeto instanceof Clase;
        - Ejemplo:
            - miObjeto instanceof unaClase;
    - this.
        - Sirve para referirse a un objeto como sustituto de un nombre.
        - Sintaxis:
            - this.propiedad;
            - this.metodo();
        - Sintaxis:
            - this.nombre;
            - this.imprimir();

Precedencia o prioridad de operadores.

- En una operación en la que estén implicados varios operadores distintos, unos actuarán antes que otros.
- La prioridad se puede modificar usando paréntesis, de modo que, lo que se encierre entre paréntesis tiene la máxima prioridad.
- Ejemplos**:**
    - 7+5**2= 32
    - (7+5)**2=144
    - 7+9+5/3=17,666 // Media incorrecta.
    - (7+9+5)/3=7
- **Prioridades**:

. (punto) y [] (Corchetes) Máxima prioridad.
() y new
! ~ - (signo menos) + (signo más) ++ -- typeof
void delete
* / ** %
+ (suma) - (resta)
<< >> >>>
< <= > >= in instanceof
== != === !==
&
|
&&
||
?: (operador condicional)
= += -= *= **= /= %= <<= >>= >>>= Mínima prioridad.

MÉTODOS DE ENTRADA Y SALIDA DE DATOS.

prompt()

- Método que permite introducir datos desde teclado.
- Los datos introducidos con prompt() se transforman siempre a cadenas de caracteres.
- Sintaxis:

    - variable que recibe el dato = prompt(“mensaje a mostrar al usuario”, ”valor inicial a mostrar”);
- Ejemplos:
    - var edad;
    - edad = prompt(“Escriba aquí su edad”,””);
    - edad = prompt(“Escriba aquí su edad”);
    - edad = prompt(“Escriba aquí su edad”, ”La edad en número no en letra.”);

console.log().

- Muestra en la consola del navegador los resultados de la ejecución de un script.
- Permite consultar error si los ha habido y depurarlos.
- Obligatorio abrir la consola del navegador (F12, Ctrl+Mayus+J (Firefox), , Ctrl+Mayus+C (Chrome), consola del navegador (Firefox), herramientas para desarrolladores(Chrome)).
- Sintaxis:
    - console.log(“cadena de caracteres”);
    - console.log (variable);
- Ejemplos:
    - console.log (“Tú te llamas: ” + nombre);

document.write().

- Para mostrar en la pantalla del navegador texto, valores, contenidos de variables, constantes, objetos, etc.
- Sintaxis:
    - document.write(“cadena de caracteres”);
    - document.write(variable);
- Ejemplos:
    - document.write(“Tú te llamas: ”);
    - document.write(DNI);
    - document.write(“Tú te llamas: ” + nombre + “ y tienes “+ edad +” años”);
    - document.write(“El valor es: “ + numero1 + numero2); Resultado: una concatenación de numero1 y numero2.
    - document.write(“El valor es: “ + (numero1 + numero2)); Resultado: una suma de numero1 y numero2.

document.writeln().

- Para mostrar en la pantalla del navegador texto, valores, contenidos de variables, constantes, objetos, etc., y luego da un salto de línea.
- Sintaxis:
    - document.writeln(“cadena de caracteres”);
    - document.writeln(variable);
- Ejemplos:
    - document.writeln(“Tú te llamas: ”);
    - document.writeln(DNI);
    - document.writeln(“Tú te llamas: ” + nombre + “ y tienes “+ edad +” años”);
    - document.writeln(“El valor es: “ + (numero1 + numero2));

CONVERSIÓN DE TIPOS.

Conversión implícita.

- Conversión automática cuando JavaScript detectar que los operandos tienen distinto tipo al usar determinados operadores.
- Ejemplo:
    - var a =”5”;
    - var b = 4;
    - Concatenación:
        - var c = a+b -> 54
    - Conversión implícita:
        - var d = a*b -> 20

Conversión explicita.

- Se fuerza la conversión de tipos usando métodos o funciones como:
- **Métodos de conversión.**
    - **parseInt().**
        - Permite convertir una cadena de caracteres con un número, a un número de tipo entero sin decimales.
        - Sintaxis:
            - parseInt(dato a convertir a número entero);
        - Ejemplos:

```
var a = “1234”;
a = parseInt(a);

var edad;
edad = parseInt(prompt(“Escriba su edad: “,””));
o también:
    edad = prompt(“Escriba su edad: “,””);
    edad = parseInt(edad);
```

    - **parseFloat().**
        - Permite convertir una cadena de caracteres con un número, a un número de tipo real con decimales.
        - Sintaxis:
            - parseFloat(dato a convertir a número real)
        - Ejemplos:

```
var b = “1234.67”;
b = parseFloat(b);

var sueldo;
sueldo = parseFloat(prompt(“Escriba el Sueldo: “,””));
o también:
    sueldo = prompt(“Escriba el Sueldo: “,””);
    sueldo = parseFloat(sueldo);
```

- **toString()**
    - Convierte un número entero o real en una cadena de caracteres.
    - Sintaxis:
        - variable.toString();
    - Ejemplo:
        - var numero = 30;
        - var texto = numero.toString();

INSTRUCCIONES ALTERNATIVAS, DE BIFURCACIÓN O CONDICIONALES.

- **Condición simple.**

- Se utilizan dos operandos y un operador de comparación.
- Se pueden comparar diversos operandos:
  - a > 8 Variable con valor.
  - a == b Variable con otra variable.
  - a != ”JavaScript” Variable con un texto.

- **Condición compuesta:**
  - Se evalúan varias condiciones simples mediante operadores lógicos.
  - Y lógica (&&)
    - Para que se cumpla la condición global y se realicen las acciones deben cumplirse obligatoriamente todas las condiciones simples.
    - Ejemplo:
      - ((a>b) && (b>c))
      - (a>8 && c==d && e<6)
  - O lógica (||)
    - Para que se cumpla la condición global y se realicen las acciones deben cumplirse al menos una condición simple.
    - Ejemplo:
      - ((a>b) || (b>c))
      - (a>8 || c==d || e<6)
  - Combinación de operadores lógicos.
    - Ejemplo:
      - (a>8 && (c==d || e<6))

Estructuras condicionales:

Condicional simple. (if)

- Sólo se ejecutan las instrucciones si la condición es verdadera, sino no se hace nada.
- Sintaxis:

```
if (condición simple o condición compuesta)
{
        instrucciones;
}
otras instrucciones;
```

- Ejemplo:

```
var a = 7;
var b = 5;
if (a>b)
{
        document.write (“a es mayor que b”);
}
```

Condicional completa. (if…else)

- Se ejecutan unas instrucciones si la condición es verdadera y si no, otras instrucciones.
- Sintaxis:

```
if(condición simple o condición compuesta)
{
        instrucciones;
}
else
```

```
{
        otras instrucciones;
}
```

- Ejemplo:

```
var a = 7;
var b = 5;
if (a>b)
{
        document.write ("a es mayor que b");
}
else
{
        document.write ("a no es mayor que b");

}
```

Condicional anidada. (if…else…if)

- Se evalúan varias condiciones en cadena.
- Si se cumple una condición se realizan unas acciones, si no, se pasa a evaluar la siguiente condición.
- Si no se cumple ninguna condición se pueden o no realizar también acciones.
- Sintaxis:

```
if(condición 1)
{
        instrucciones;
}
else if(condición 2)
{
        instrucciones;
}
else if(condición N)
{
        instrucciones;
}
else
{

        instrucciones si no se cumple ninguna condición.; // (opcional)
}
```

- Ejemplo:

```
var a = 5;
var b = 8;
if(a>b)
{
        document.write ("a es mayor que b");
}
else if(a==b)
{
        document.write ("a es igual que b");
}
else if(a<b)
```

```
{
        document.write ("a es menor que b");
}
```

- Otro ejemplo:

```
var a = 5.6;
var b = 8.9;
if(a>b)
{
        document.write ("a es mayor que b");
}
else if(a==b)
{
        document.write ("a es igual que b");
}
else
{
        document.write ("a es menor que b");
}
```

Condicional múltiple sin anidamiento. (if… …if)

- Se evalúan varias condiciones independientes.
- Si no se cumple ninguna condición se pueden o no realizar también acciones.
- Sintaxis:

```
if(condición 1)
{
        instrucciones;
}
else
{

        instrucciones si no se cumple la condición.; // (opcional)
}

if(condición 2)
{
        instrucciones;
}
else
{

        instrucciones si no se cumple la condición.; // (opcional)
}

if(condición N)
{
        instrucciones;
}
else
{
```

```
        instrucciones si no se cumple la condición.; // (opcional)
    }
```

- Ejemplo:

```
var a = 150;
if(a>=100 && a <=200)
{
    document.write ("a está entre 100 y 200");
}
else
{
    document.write ("a no está en el intervalo");
}

if(a==150)
{
    document.write ("a es igual a 150");
}
if(a>=201 && a <=2000)
{
    document.write ("a está entre 201 y 2000");
}
```

Condicional múltiple. (switch…case)

- Permite evaluar varias condiciones a la vez, que pueden ser valores fijos en una variable, rangos o condiciones.
- Se pueden agrupar varios case cuando con todos ellos se vaya a realizar la misma acción, que se incluye al final del último que forme el grupo.
- Sintaxis:

```
switch (expresión)
{

    case valor 1:
        instrucciones;
        break;

    case valor 2:
        instrucciones;
        break;

    case valor N:
        instrucciones;
        break;

    default:
        instrucciones;
        break;
}
```

- Expresión:
    - Puede ser una variable, una expresión matemática, valor booleano...

- Valor:
  - Puede ser un texto, un número o una condición simple o compuesta entre paréntesis.
  - También se pueden usar operadores relacionales y lógicos:
    - case a > b:
    - case a > b && b != c:
  - Para utilizar operadores relacionales y lógicos como expresión en switch se debe incluir un valor booleano (true o false)
    - switch(true) {}
- break:
  - Opcional.
  - Si se cumple un valor se sale de la instrucción switch.
  - Si no se pone, se continúan evaluando otros valores.
- default:
  - Opcional.
  - Instrucciones que se realizarán si el valor evaluado no coincide con ningún case.
- Ejemplos:
  - Con variable y valores textuales.

```
var color;
color = prompt("¿Cuál es tu color favorito?","");
switch (color)
{

        case "verde":
                document.write("El " + color + " es muy bonito");
                break;

        case "negro":
                document.write("El " + color + " no me gusta");
                break;

        default:
                document.write("Prefiero otro color");
                break;
}
```

  - Con variable y valores textuales agrupados.

```
var color;
color = prompt("¿Cuál es tu color favorito?","");
switch (color)
{

        case "verde":
        case "azul":
        case "rojo":

                document.write("El " + color + " es muy bonito");
                break;

        case "negro":
```

```
        case "marrón":

                document.write("El " + color + " no me gusta");
                break;

        default:
                document.write("Prefiero otro color");
                break;
}
```

- Con valor booleano y condición.

```
var n1 = 67;
var n2 = 56;
switch(true)
{
        case (n1 > n2):
                document.write("El primer número es mayor que el segundo");
                break;
        case (n1 == n2):
                document.write("Ambos números son iguales");
                break;
        case (n1<n2):
                document.write("El segundo número es mayor que el primero");
                break;
 }
```

INSTRUCCIONES REPETITIVAS O BUCLES.
Mientras – While.

- Se ejecutan un conjunto de instrucciones en bucle mientras se cumpla una condición.
- Cada vuelta en el bucle se denomina iteración.
- Si no se cumple inicialmente la condición no se entra en el bucle.
- Puede funcionar con una variable contadora o no.
- Sintaxis:

```
while (condición)
{
        instrucciones;
}
```

- Ejemplos**:**
  - **Con variable contadora.**

```
var a = 0;
while(a<=9)
{
        document.write (a);
        a++  / a=a+1 / a+=1;  // Incrementos de unidad en unidad.
        a=a+5 / a+=5; // incrementos de 5 en 5.
}
document.write("Fin del bucle" + a);
```

  - **Con condición a cumplir.**

```
var b = prompt("Escribe:");
document.write ("Has escrito " + b + "<br>");
while(b != "hola")
{
var b = prompt("Escribe:");
document.write (b);
}
document.write ("Fin del bucle" + b);
```

Hacer-Mientras o Repetir-Hasta o Do-While.

- Se ejecutan un conjunto de instrucciones en bucle mientras se cumpla una condición.
- Cada vuelta en el bucle se denomina iteración.
- Si no se cumple inicialmente la condición al menos se realizan una vez las instrucciones.
- Puede funcionar con una variable contadora o no.
- Sintaxis**:**

```
do
{
    instrucciones;
} while (condición)
```

- Ejemplos**:**
  - **Con variable contadora.**

```
var a = 0;
do
{
    document.write (a);
    a++ ; / a=a+1 / a+=1  // Incrementos de unidad en unidad.
    //a=a+5 / a+=5 // incrementos de 5 en 5.
} while(a<=9)
document.write ("Fin del bucle" + a);
```

  - **Con condición a cumplir.**

```
do
{
    var b = prompt("Escribe:");
    document.write (b);
} while(b != "hola")
document.write ("Fin del bucle" + b);
```

Bucle for.

- Se repiten unas instrucciones un número determinado de veces hasta que deja de cumplirse una condición.
- La condición está determinada por una variable contadora.
- Se pueden usar varias variables contadoras para anidar un bucle for a otro.
- Sintaxis:

```
for(variable contadora = valor inicial; condición; incremento o decremento)
{
    Instrucciones;
}
```

  - **Valor inicial**:

- Valor inicial de la variable contadora que puede crearse antes del bucle o en el momento de asignarle el valor inicial.

- **Condición**:
  - Coincide con el valor final de la variable contadora.
  - Evalúa en cada vuelta si el valor de la variable contadora cumple la condición, si no, se finaliza el bucle.
  - Si la condición es falsa desde el principio no se ejecuta el bucle.
- **Incremento/decremento**:
  - Permite actualizar la variable contadora con el fin de hacer que llegue a ser falsa la condición en algún momento.

- Ejemplos:
  - Variable contadora definida antes del bucle:

```
var x;
for(x = 1; x <=10;x++)
{
        document.write("la variable x vale: " + x);
}
```

  - Variable contadora definida dentro de la instrucción for.

```
for(var x = 1; x <=10;x++)
{
  document.write("la variable x vale: " +x);
}
```

  - Dos bucles anidados.

```
var x;
var y;
for(x = 1; x <=10;x++)
{
        for(y = 1; y <=3;y++)
        {
                document.write("valor de x: " + x+ "valor de y: "+ y);
        }
}
```

- **Sentencias break y continue**.
  - Se usan en bucles y estructuras condicionales para finalizar la repetición de instrucciones o terminar las condiciones.
  - **break**.
    - Sirve para finalizar prematuramente un bucle.
    - Se incluye en una condición dentro del bucle que hace que este termine.
    - Ejemplo:

```
var x;
for(x = 1; x <=10;x++)
{
        if (x==5)
        {
                break;
        }
   document.write("la variable x vale: " + x);
}
```

  - **continue**.

- Permite saltar una iteración de un bucle, ignorando las instrucciones que hubiera a continuación.
- Tras el salto el bucle continúa funcionando hasta que termine de forma normal.
- Ejemplo:

```
var x;
for(x = 1; x <=10;x++)
{
        if (x==5)
        {
            continue;
        }
document.write("la variable x vale: " + x);
}
```

FUNCIONES.

Concepto.

- Conjunto de instrucciones u operaciones agrupadas dentro de un mismo bloque.
- Ahorrar escribir código que se repite con frecuencia.
- Pueden ejecutarse en cualquier parte de un programa.
- Pueden devolver o no valores.
- Funciones miembro:
  - Funciones que forman parte o son miembros de una clase.

Tipos.

- Creadas por el usuario.
- Predefinidas:
  - Incluidas en JavaScript.

FUNCIONES CREADAS POR EL USUARIO.

- **Creación o definición de una función.**
  - Se usa la palabra reservada function**.**
  - Pueden declararse en cualquier parte del código.
  - Pueden o no incluir argumentos**.**
  - Sintaxis básica:

```
function nombreFunción(argumento 1, argumento  2,….,argumento N)
{
        Instrucciones;
}
```

  - Nombre de una función:
    - Se puede usar caracteres alfanuméricos, guiones bajo o de subrayado.
    - Sensible a mayúsculas y minúsculas.
    - No se pueden usar nombres de funciones ya creadas o palabras reservadas.
    - Uso nomenclatura camelCase.
      - calcularArea()
  - Bloque de código:
    - Se encierra entre llaves {}.
    - Puede incluir:

        - Declaración de variables y constantes.
        - Bucles e instrucciones condicionales.
        - Llamadas a otras funciones.
        - Etc.
    - Argumentos:
        - Son opcionales.
        - Se denominan parámetros.
        - Representan datos con los que va a operar la función.

- **Llamada a una función:**
    - Ejecución de una función.
    - Una función se puede ejecutar o llamar varias veces.
    - Desde una función se pueden ejecutar otras.
    - Al llamarla, todas las instrucciones que incluya en su bloque de código se ejecutan.
    - No se puede llamar a una función que no esté definida.
    - Sintaxis:
        - nombre de la función();
    - Ejemplo:
        - calcularSaldo();

- **Lugares de llamada**:

    - **Función definida en el mismo <script> y antes de hacer la llamada**:

```
<script>
        function sumar()
        {
            instrucciones;
        }

        sumar();
</script>
```

    - **Función definida en el mismo <script> y después de hacer la llamada**:

```
<script>
        sumar();

        function sumar()
        {
            instrucciones;
        }

</script>
```

    - **Función definida en un <script> que es llamada desde otro distinto**:

```
<script>
        function sumar()
        {
            instrucciones;
        }

</script>
```

```
<script >
        sumar();

</script>
```

- En este caso es obligatorio que el bloque con la definición de la función este situado antes que del bloque donde se hace la llamada, si no, error.

- **Usando un enlace HTML**:
  - Cuando las funciones son sencillas se puede usar un enlace para llamarlas.
  - El enlace debe incluir el término JavaScript y la función dentro del atributo "href".
  - Si se emplea JavaScript en los enlaces, para abreviar el código, el navegador lo trata como una URL, de modo que no debe haber espacios en blanco y, si son necesarios, se tienen que escapar (%20).
  - Sintaxis:
    - Texto de enlace
  - Ejemplos:
    - Calcular suma
    - Mostrar mensaje
    - Renfe
- **Ejecutar la función al cargar la página web**:
  - Usar el atributo onload dentro de la etiqueta <body>
  - Sintaxis:
    - <body onload = "nombre de la función (argumentos o parámetros si lleva);">
  - Ejemplo:
    - <body onload = "sumar ();">

- **Ejecutar la función al producirse un evento.**
  - Ejemplo:

```
fuction pulsar()
{
        alert("¡Has pulsado el botón!");
}
<input type ="button" name = "boton1" value = "Pulsar" onclick = "pulsar()">
```

- **Recomendación**.
  - Evitar errores en llamadas a función haciendo lo siguiente:
    - Declarándolas todas al principio de la página dentro de <head>.
    - Incluir todas las funciones en un fichero externo, por ejemplo, con el nombre "funciones.js".
      - Ejemplo:

En la página web A.html:

```
<script src = "funciones.js">
        sumar();
        restar();
</script>
```

En la página web B.html:

```
<script src = ”funciones.js”>
        sumar();
        dividir();
        multiplicar();
</script>
<script>
        dividir();
        potencia()
</script>
```

- **Tipos de funciones**.

  - **Sin retorno de datos y sin parámetros.**
    - Incorporan los valores literales con los que hacer las operaciones dentro del bloque de código de la función asignados a variables u otras estructuras de datos.
    - Si se quieren cargar los datos por teclado, la función prompt(), se incluye dentro del bloque de código de la función.
    - No devuelven ningún valor que haya que recoger en una variable fuera de la función**.**
      - Sintaxis:

```
function nombreFunción()
{
        Instrucciones;
}
```

      - Ejemplo:

```
function restar() // Valores literales.
{
        let n1 = 4;
        let n2 = 6;
        document.write("El resultado de la resta es: " + (n1-n2) + "<br>");
}
function restar() // Con función prompt().
{
        let n1 = prompt(“Introduce un número”;””);
        let n2 = prompt(“Introduce otro número”;””);
        document.write("El resultado de la resta es: " + (n1-n2) + "<br>");
}
```

      - Llamada a la función:
        - restar(); // Sin parámetros y sin variable para cargar resultados retornados.
  - **Sin retorno de datos y con parámetros.**

    - **Parámetro:**
      - Dato que recibe la función y que permite realizar las operaciones dentro de la misma.
      - Indican a una función los valores con los que tiene que operar.
      - Si no se incluyen, la función utiliza los valores fijos que se especifican en su bloque de instrucciones, y siempre hace lo mismo.

- Se especifican dentro de los paréntesis de la función.
- Se pueden incluir varios separados por comas.
- Sintaxis:

```
function nombreFunción(nombre parámetro 1, nombre parámetro 2,…,nombre parámetro N)
{
    Instrucciones;
}
```

- Ejemplo:

```
function sumar3(numero1, numero2, numero3)
{
    document.write("El resultado de la suma es: " + (numero1+numero2+numero3) + "<br>");
}
```

- Paso de parámetros.
  - Para pasar parámetros a una función hay que incluir en los paréntesis de ésta un valor para cada uno de los parámetros separado por una coma.
  - Si se omite algún parámetro se inicializa con el valor “undefined”, a no ser que haya variables con valores ya dentro de la función.
  - Los valores deben incluirse en el mismo orden en el que están definidos los parámetros.
  - Valores como parámetros:
    - Valores literales, variables, expresiones aritméticas, funciones anidadas, objetos, etc.,
    - El parámetro se inicializa con dichos valores.
  - Carga de parámetros con el método prompt().
    - Se usa como valores en la llamada a una función las variables que ha cargado los datos por teclado.
  - Ejemplos:
    - Con valores literales:

```
function cliente(DNI, nombre)
{
document.write("Te llamas : “ + nombre + “ y tu DNI es el “ + DNI + "<br>");
}
cliente(“73423234V”, ”Pepe”); //Datos literales. Se pasan 2 cadenas de texto a la función que los mostrará concatenados.
…….
function sumar3(numero1, numero2, numero3)
{
document.write("El resultado de la suma es: " + (numero1+numero2+numero3) + "<br>");
}
sumar3(7,9,5); //Datos literales. Se pasan 3 números a la función que los utilizará para sumarlo.
sumar3(17,90,55); // Ídem.
```

    - Con valores cargados por teclado con prompt():

```
let a = parseInt(prompt("Intro un número",""));
```

```
let b = parseInt(prompt("Intro otro número",""));
let c = parseInt(prompt("Intro otro más número",""));
sumar3(a,b,c); // Datos dentro de variables cargadas por teclado.
```

- **Parámetros opcionales y valores por defecto**.
  - Para que un parámetro sea opcional no puede ser el primero.
  - Para que sea opcionales hay que asignarle un valor por defecto.
  - Sólo pueden ser opcionales aquellos parámetros situados a la derecha del último que sea necesario para realizar las acciones de la función.
  - Asignación de valores por defecto:
    - **Dentro del bloque de código**:
      - Se asigna el valor ente las llaves del bloque de código.
      - Sintaxis:

```
function nombre de la función(parámetro 1, parámetro 2, …, parámetro N)
{
        if( typeof parametro2 == “undefined”)
        {
                parámetro 2 = valor por defecto;
        }
        instrucciones;
}
```

      - Al llamar la función se pueden o no incluir los parámetros opcionales. Si se incluyen sus valores, son estos los que utiliza la función, si no, se utilizan los valores definidos como valores por defecto.
        - Ejemplo:

```
function calcular (n1, n2, n3, n4)
{
    if( typeof n3 == “undefined”)
    {
        n3 = 20;
    }
    if( typeof n4 == “undefined”)
    {
        n4 = 120;
    }
    return n1 + n2 + n3 + n4;
}
var a = calcular (20,50,70,30);

var b = calcular (30,50); Equivale a calcular (30,50,20,120);
```

    - **Al definir los parámetros**.
      - Se incluye el valor por defecto en la definición del parámetro dentro de los paréntesis de la función.
      - Cualquier parámetro puede ser opcional.
      - Sintaxis:

```
function nombre de la función(parámetro 1 = valor, parámetro 2= valor,…, parámetro N)
{
```

```
        instrucciones;
    }
```

- Al llamar la función se pueden o no incluir los parámetros opcionales. Si se incluyen sus valores, son estos los que utiliza la función, si no, se utilizan los valores definidos como valores por defecto.
- Se usan como parámetros los valores en el mismo orden en que se han incluido en la llamada a la función.
- Ejemplos:

```
function calcular (n1=23, n2=4, n3=4, n4=8)
{
    return n1 + n2 + n3 + n4;
}
var a = calcular (20,50,70,30);

var b = calcular (30,50); Equivale a calcular (30,50,4,8);

var c = calcular (); Equivale a calcular (23,4,4,8);
```

- El orden de los valores es importante, porque si no se respeta pueden ocurrir errores:

```
function calcular (n1=23, n2, n3=4, n4)
{
    return n1 + n2 + n3 + n4;
}
var a = calcular (20,50,70,30);

var b = calcular (30,50); //Error porque el cuarto parámetro se queda sin valor, al usar los valores en el mismo orden en el que se han especificado en la función.
```

- **Con retorno de datos y sin parámetros.**
  - Si la función produce un resultado y se quiere utilizar después, hay que almacenar éste en una variable, constante, …
  - Para devolver el valor se utiliza la instrucción **return**.
  - Sintaxis**:**

```
function nombreFunción()
{
    Instrucciones;
    return valor a devolver;
}
```

  - Llamada a la función**:**
    - Hay que crear una variable y usar una operación de asignación para que se cargue en ella el valor devuelto.
    - Sintaxis:
      - [var o let] variable que almacenara el valor devuelto = función();
    - Ejemplo:

```
function sumar()
{
    var a = 7;
    var b = 5;
    return a+b;
}
var suma;
suma = sumar();
document.write(suma);
```

- **Con retorno de datos y con parámetros.**
  - Si la función produce un resultado y se quiere utilizar después, hay que almacenar éste en una variable, constante, …
  - Para devolver el valor se utiliza la instrucción **return**.
  - Incluyen parámetros para pasar datos a la función.
  - Sintaxis**:**

```
function nombreFunción(nombre parámetro 1, nombre parámetro 2,…)
{
    instrucciones;
    return valor a devolver;
}
```

  - Llamada a la función**:**
    - Hay que crear una variable y usar una operación de asignación para que se cargue en ella el valor devuelto.
    - La función debe incluir obligatoriamente los valores para los parámetros con los que se ha definido.
    - Sintaxis:
      - var variable que almacenara el valor devuelto = función (valores para los parámetros);
    - Ejemplos:

```
function sumar3(a,b,c)
{
    var d = a+b+c;
    return d;
    //return a+b+c;
}
```

Opción con datos literales:

```
var resultado1 = sumar3(7,9,5);
```

Opción con datos dentro de variables:

```
var a1 = parseInt(prompt("Intro un número",""));
var b1 = parseInt(prompt("Intro otro número",""));
var c1 = parseInt(prompt("Intro otro número más",""));

var resultado2 = sumar3(a1, b1, c1);
```

Retorno de varios datos.

- **Con Arrays.**

  - En una función que devuelve datos cuando es ejecutada no se puede usar más de una instrucción *return*.
  - Si se quisiera devolver más de un resultado calculado dentro una función, estos resultados deben incluirse en un array.
  - Los resultados incluidos en el array pueden ser variables, datos literales, etc.
  - Sintaxis:
    - Creando una variable que reciba los datos retornados:
      - var nombre del array = [resultado 1, resultado 2, … resultado N];
      - return nombre del array;
    - Sin crea variable que reciba los datos retornados:
      - return [resultado 1, resultado 2, … resultado N];
  - Ejemplo:

```
function calcular(n1, n2)
{
        let suma = n1 + n2;
        let resta = n1- n2;
        let resultados = [suma, resta];
        return resultados;
        //return [suma, resta];
}
let calculos = calcular(5,7);
document.write("La suma es: " + calculos[0]);
document.write("La resta es: " + calculos[1]);
```

- **Con return alternativos.**

  - En este caso, solo se devuelve un resultado con return, pero puede haber varios de ellos que se ejecutan según se cumplan unas u otras condiciones.
  - Ejemplo:

```
function comparar(n1, n2)
{
        If (n1 >n2)
        {
                return "n1 es mayor";
        }
        else If (n1 <n2)
        {
                return "n2 es mayor";
        }
        else If (n1 == n2)
        {
                return n1 + n2;
        }
}
```

Resumen de llamadas a una función.

- sumar(); // Llamada a función sin parámetros y sin retorno.
- sumar(8,4,"pepe"); // Llamada a función con parámetros y sin retorno.
- var a = sumar(); // Llamada a función sin parámetros y con retorno.
- var b = sumar(7,40,3,8,7,6,4,3,6,7,8,4,); // Llamada a función con parámetros y con retorno.

Funciones anónimas - Variables como función.

- Las funciones son un tipo más al declarar una variable.
- Se puede definir y almacenar el bloque de instrucciones dentro una variable y después ejecutarlo.
- El nombre de la variable será el nombre de la función.
- Las funciones creadas así tienen las mismas características que las creadas de otras formas (pueden tener o no parámetros y pueden o no retornar datos.).
- Uso frecuente como parámetro de otra función o dato a pasar como parámetro.
- Sintaxis:

```
var nombre = function (Con o sin parámetros)
{
    instrucciones;
}
```

- Ejemplos:

```
var sumar = function (n1, n2)
{
    return n1 + n2;
}
var restar = function ()
{
    var n1 = 4;
    var n2 = 2;
    var n3 = n1 – n2;
    document.write("Resultado resta: " + n3);
}
```

- Ejecución o llamada de una función anónima:
  - Igual que cualquier otra función.
  - Sintaxis:
    - nombre de la función (Con o sin parámetros);
    - var variable = nombre de la función (Con o sin parámetros);
  - Ejemplos:

```
restar();

var resultado;
resultado = sumar(4,2);
document.write("Resultado suma: " + resultado);
```

Funciones flecha (arrow).

- Forma abreviada de definir una función.
- Se elimina la palabra function y se añade = > antes de abrir las llaves.
- Las funciones flecha son siempre anónimas, por lo que hay que cargarlas en una variable.
- Consideraciones:

- Si el cuerpo de la función sólo tiene una línea pueden omitirse las llaves {}.
- Si el cuerpo de la función sólo tiene una línea puede omitirse return.
- Si la función no tiene parámetros, se indica () = >
- Si solo tiene un parámetro, se indica sólo el nombre de este sin paréntesis:
  - nombre parámetro = >
- Si tiene 2 o más parámetros, se indican entre paréntesis:
  - (nombre parámetro 1, nombre parámetro 2, …, nombre parámetro N) = >

- Sintaxis de declaración:

```
var nombre = (Con o sin parámetros) =>
{
        instrucciones;
}
```

- Ejemplos:

```
var sumar1 =  (n1, n2) =>
{
        return n1 + n2; // Al tener sólo una línea de código, return se puede quitar.
}

var sumar2 = (n1, n2) => n1 + n2 ;

var sumar3 =  () =>
{
        var n1 = 4;
        var n2 = 3;
        return n1 + n2;
}

var restar = () =>
{
        var n1 = 4;
        var n2 = 2;
        var n3 = n1 - n2;
        document.write("Resultado resta: " + n3);
}

var cuadrado = numero =>
{
        var n1 = numero **2;
        document.write("Resultado potencia: " + n1);
}
```

- Ejecución o llamada de una función flecha:
  - Igual que cualquier otra función.
  - Sintaxis:
    - nombre de la función (Con o sin parámetros);
    - var variable = nombre de la función (Con o sin parámetros);
  - Ejemplos:

```
restar();
```

```
var resultado;
resultado = sumar1(4,2);
document.write("Resultado suma: " + resultado);

var suma3;
suma3 = sumar3();
document.write("Resultado suma: " + suma3);
```

Funciones Callback (Llamada de nuevo o retro llamada).

- Son funciones que se incluyen como argumento de otras funciones.
- La función que incluye en sus argumentos a una o varias funciones callback, se la denomina de *orden-superior* (*high-order*), y contiene las instrucciones que determinan cuando se ejecutará la función callback.
- Si la función callback no tiene parámetros, cuando se pasa como parámetro no se incluyen los paréntesis vacíos.
- Si la función callback tiene parámetros, cuando se pasa como parámetro, se incluyen éstos de la misma forma en otro tipo de funciones.
- Uso habitual en operaciones asíncronas.
- Sintaxis:
    - Nombre de la función de primer orden (función/es callback, otros argumentos si son necesarios);
- Ejemplo:

```
function soyUnaFuncionCallback()
 {
      documento.write("Función ejecutada transcurridos 2 segundos");
}
setTimeout(soyUnaFuncionCallback, 2000);
```

Funciones predefinidas de JavaScript.

- **isNaN(valor).**
    - Comprueba si un valor no es un número.
    - Resultados:
        - ***false***:
            - Si el valor es un número.
        - ***true***:
            - Si el valor no es un número.
    - Antes de comprobar el valor, se aplica la conversión automática implícita de tipos.
    - Sintaxis:
        - isNaN(valor)
            - El valor booleano se puede cargar en una variable.
            - valor:
                - Números.
                - "Textos".
                - Variables.
                - Otros.
    - Ejemplos:
        - var a = 5;

        - var b = “1.8”;
        - var c = “hola”;
        - isNaN(a) // falso.
        - isNaN(60) // falso.
        - isNaN(b) // verdadero.
        - isNaN(c) // verdadero.
        - isNaN(“adiós”) // verdadero.
    - Se puede usar como operando en una condición para preguntar por el valor resultante.
        - (isNaN(valor) == ***false***) // Indica que valor es un número.
        - (isNaN(valor) == ***true***) // Indica que valor no es un número.
- **isInfinite(valor).**
- **decodeURI()**
    - Función complementaria a encodeURI().
    - Permite decodificar una URL previamente codificada con encodeURI() o que por defecto se muestra ya codificada.
    - Sintaxis:
        - var urldecodificada = decodeURI(“URL codificada”);
    - Ejemplos:
        - var miURL = decodeURI(“https://www.sitio.com/enviar%20correo%20web.html”);
        - var miURL = document.write("Direccion decodificada: " + decodeURI(fotos[2].src) + "
");
- **encodeURI()**
    - Función complementaria a decodeURI().
    - Permite codificar una URL previamente decodificada con decodeURI() o que por defecto se muestra no codificada.
    - Caracteres de codificación:
        - Ver en https://es.wikipedia.org/wiki/C%C3%B3digo_porciento
    - Sintaxis:
        - var urlcodificada = encodeURI(“URL codificada”);
    - Ejemplos:
        - var miURL = encodeURI(“https://www.sitio.com/enviar correo web.html”);
        - var miURL = document.write("Direccion codificada: " + encodeURI(fotos[2].src) + "
");

CLASES U OBJETOS PROTOTÍPICOS EN JAVASCRIPT.

Tipos.

- Creados por el usuario.
- Predefinidos:
    - Son los objetos nativos incluidos en JavaScript.
    - Tipos:
        - **Envoltorios o Wrappers**.
            - Se usan para los tipos predefinidos.
            - Boolean, String, Number, …
        - **Utilidades**.
            - Date, Math, RegExp, JSON, Object, Set, Map, …
        - **Vectores**.
            - Array.

- **Características del navegador**.
    - navigator.
- **Objetos DOM.**
    - window, document, frame, location, history, screen, …
    - **Objetos DOM de document**
    - Genéricos:
        - image, applet, link, form, …
        - Objetos de form - Formularios:
            - radio, button, textarea, ….

CREACIÓN Y DEFINICIÓN DE OBJETOS (PROTOTÍPICOS O CLASES).

Con función constructora.

- Tener claro que atributos o propiedades y funciones o métodos formaran parte del objeto.
- Definir un constructor.
- **Operadores para Objetos**:
    - **this**:
        - Se usa para hacer referencia los miembros o elementos de un objeto.
        - Se usa también para hacer referencia a un objeto.
        - Permite diferenciar entre parámetros o valores de parámetros y miembros (propiedades o métodos) del objeto.

- Sintaxis:
  - Objeto sin parámetros en su constructor.

```
function nombre del Objeto()
{
        Definición de propiedades;
        Definición de métodos;
}
```

  - Ejemplo:

```
function Coche()
{
        this.marca ;
        this.modelo;
        this.color;
        this.precio;
        this.cilindrada;

        this.acelerar = function()
        {
                document.write("Estoy acelerando")
        }
        this.frenar = function()
        {
                document.write("Estoy frenando")
        }
        this.cambiarMarcha = function(marcha)
        {
                document.write("Cambiando de marcha")
                //return algo;
        }

}
```

  - Objeto con parámetros en su constructor.
    - También se crea automáticamente un constructor sin parámetros.

```
function nombre del Objeto(parámetro 1 o propiedad 1, parámetro2  o propiedad
2,…,parámetro N o propiedad N)
{
        this.propiedad 1 = parámetro 1;
        this.propiedad 2 = parámetro 2;
        this.propiedad N = parámetro N;

        this.metodo 1 = function (Pueden incluir o no parámetros)
        {
                instrucciones;
        }
        this.metodo 2 = function (Pueden incluir o no parámetros)
        {
                instrucciones;
        }
```

```
        this.metodo N = function (Pueden incluir o no parámetros)
        {
            instrucciones;
        }

    }
```

Ejemplo:

```
    function Coche(marca, modelo, color, precio, cilindrada)

    {
        this.marca = marca;
        this.modelo = modelo;
        this.color = color;
        this.precio = precio;
        this.cilindrada = cilindrada;

        this.acelerar = function()
        {
            document.write("Estoy acelerando")
        }
        this.frenar = function()
        {
            document.write("Estoy frenando")
        }
        this.cambiarMarcha = function(marcha)
        {
            document.write("Cambiando de marcha")
            //return algo;
        }

    }
```

Creación directa de una instancia – Objeto literal.

- No es un constructor, ya que no se pueden crear instancias a partir de esta estructura.
- Es una especie de constructor de un solo uso.
- Sintaxis:
    - **Genérica**.

```
    var nombre de la instancia u objeto =
    {
        Propiedad 1;
        Propiedad 2;
        Propiedad N;

        método 1 = function (Pueden incluir o no parámetros)
        {
            instrucciones;
        },
        método 2 = function (Pueden incluir o no parámetros)
        {
```

```
        instrucciones;
    },

    método N = function (Pueden incluir o no parámetros)
    {
        instrucciones;
    }

};
```

- **Específica**.

```
var nombre de la instancia u objeto =
{
    Propiedad 1: valor,
    Propiedad 2: valor,
    Propiedad N: valor,

    método 1: function (Pueden incluir o no parámetros)
    {
        instrucciones;
    },
    método 2: function (Pueden incluir o no parámetros)
    {
        instrucciones;
    },

    método N: function (Pueden incluir o no parámetros)
    {
        instrucciones;
    }

};
```

- **Vacío**.
  - Si se crea un objeto vacío después hay que asignar valores a propiedades y crear los métodos usando el operador punto.
  - Sintaxis:
    - **Creación del objeto**:
      - var nombre de la instancia u objeto = {};
    - **Asignar valores**:
      - nombre objeto.propiedad = valor;
    - **Crear métodos**:

```
nombre objeto.metodo = function()
{
        instrucciones;
}
```

  - Ejemplo:

```
var persona2 = {};

persona2.nombre = "Juan";
persona2.apellidos = "Fernández García";
persona2.direccion = "C/ Toledo 12";
persona2.telefono = "123456789";
```

```
persona2.aficiones = "Bonsais";
persona2.mostrarDatos = function()
{
        document.write(persona2.nombre + " " + persona2.aficiones+ "<br>");
        document.write(persona2["apellidos"] + " " + persona2["telefono"]);
}

persona2.mostrarCodigo = function(n)
{
        document.write("El código de " + persona2.nombre + " es el " + n + "<br>");
}
```

Creación de una instancia usando el objeto Object.

- Se crea un objeto vacío al que después hay que asignar valores a propiedades y crear los métodos usando el operador punto.
- Sintaxis:
    - **Creación del objeto**:
        - var nombre de la instancia u objeto = new Object();
    - **Asignar valores**:
        - nombre objeto.propiedad = valor;
    - **Crear métodos**:

```
nombre objeto.metodo = function()
{
        instrucciones;
}
```

- Ejemplo:

```
var persona2 = new Object();

persona2.nombre = "Juan";
persona2.apellidos = "Fernández García";
persona2.direccion = "C/ Toledo 12";
persona2.telefono = "123456789";
persona2.aficiones = "Bonsais";
persona2.mostrarDatos = function()
{
        document.write(persona2.nombre + " " + persona2.aficiones+ "<br>");
        document.write(persona2["apellidos"] + " " + persona2["telefono"]);
}

persona2.mostrarCodigo = function(n)
{
        document.write("El código de " + persona2.nombre + " es el " + n + "<br>");
}
```

Creación de una instancia.

- Se usa el operador new.
- Se crea una copia de un objeto con todas sus propiedades y métodos listos para usarse.
- Sintaxis:
    - var/let/const nombre de la instancia u objeto = new nombre del Objeto o Clase();

    - var/let/const nombre de la instancia u objeto = new nombre del Objeto o Clase (valor parámetro 1, valor parametro2,…, valor parámetro N);
    - var es opcional, y también puede usarse let, const para crear los objetos o instancias.
- Ejemplo:
    - var coche1 = new Coche();
    - var coche2 = new Coche("Ford", "Fiesta", "Azul", 18000.00, "1800 c.c.");
    - coche3 = new Coche();
    - const coche4 = new Coche();

Acceso a los elementos de un objeto o instancia.

- Se accede una propiedad para mostrar su valor o para asignárselo.
- Se accede a un método para ejecutarlo.

Uso del operador punto (.)

- Sintaxis:
    - nombre del objeto o instancia.propiedad;
    - nombre del objeto o instancia.metodo();
- Ejemplos:
    - Acceso a las propiedades para mostrar su contenido.
        - document.write("la marca del coche es :" + coches1.marca);
        - document.write("la marca del coche es :" + coches2.marca);
    - Acceso a las propiedades para asigna un valor.
        - coches1.marca = "Seat";
        - coches2.precio = 20000.50;
    - Ejecutar o llamar a un método.
        - coches1.acelerar();
        - coches2.cambiarMarcha("5ª");
        - var marchaIntroducida = coches2.cambiarMarcha("5ª");

Uso del operador corchetes[]

- Sintaxis:
    - nombre del objeto o instancia["propiedad"];
    - nombre del objeto o instancia["método"]();
- Ejemplos:
    - Acceso a las propiedades para mostrar su contenido.
        - document.write("la marca del coche es :" + coches1["marca"]);
        - document.write("la marca del coche es :" + coches2["marca"]);
    - Acceso a las propiedades para asigna un valor.
        - coches1["marca"]= "Seat";
        - coches2["precio"] = 20000.50;
    - Ejecutar o llamar a un método.
        - coches1["acelerar"]();
        - coches2["cambiarMarcha"]("5ª");
        - var marchaIntroducida = coches2["cambiarMarcha"]("5ª");

Otros operadores para objetos.

delete.

- Elimina el dato de una propiedad o atributo.
- El dato es sustituido por el valor “undefined”.
- Sintaxis:
    - delete nombre objeto.propiedad;
- Ejemplo:
    - delete coche1.modelo;

instanceof.

- Indica si una instancia u objeto pertenece a una clase u objeto prototípico determinado.
- Para preguntar si una instancia forma parte de un objeto, ésta debe existir, aunque sea de un objeto distinto.
- Si se incluye concatenado a una cadena, deben ir entre paréntesis la instancia, el operador y el objeto.
- Devuelve true o false.
- Sintaxis:
    - Nombre objeto o instancia instanceof nombre clase u objeto prototípico.
    - “Cadena” + (Nombre objeto o instancia instanceof nombre clase u objeto prototípico) + “Cadena”.
- Ejemplo:
    - document.write(coche1 instanceof Coches); mostrará true si coche1 existe como instancia y es del tipo Coches.
    - document.write(coche1 instanceof Viajes); mostrará false si coche1 existe como instancia pero es de otro tipo diferente.
    - document.write(“El coche7 es una instancia de Viajes” + (coche7 instanceof Viajes)); mostrará false si coche7 existe como instancia.

in.

- Indica si una propiedad existe o no en la instancia u objeto.
- Devuelve true o false.
- También se puede buscar una propiedad en los objetos o clases predefinidos de Javascript.
- Sintaxis:
    - “propiedad” in nombre de la instancia u objeto.
- Ejemplo:
    - document.write(“marca” in coches1); mostrará true.
    - document.write(“marca” in viajes1); mostrará false.
    - document.write(“length” in String); mostrará true.

Uso del Bucle for…in para recorrer un objeto.

- Se usa la variable contadora para dar vueltas en el bucle y también como elemento miembro del objeto.
- Los elementos miembros (propiedades y métodos), se cargan en la variable contadora en el mismo orden en el que están declarados dentro de la definición del objeto.
- Sirve para recorrer cualquier objeto independientemente de cómo esté creado (constructor, objeto literal, etc.).
- Sintaxis:

```
var contadora o índice;
for (variable contadora o índice in nombre del objeto)
{
        Instrucciones;
}
```

- Ejemplo**:**

```
function Datos()
{
        this.nombre;
        this.correo;
        this.verDatos = function()
        {
        document.write("nombre: " +this.nombre + " tu correo es " + this.correo);
        }
}

var cliente1 = new Datos("Pepe", "pepe@gmail.com");
var cliente2 = new Datos("Juan", "juan@gmail.com");

var i;
for (i in cliente1)
{
        document.write("En la propiedad " + i + " está el dato " + cliente1[i]);
}
```

CLASES U OBJETOS PROTOTÍPICOS DE JAVASCRIPT.

ARRAY

- Estructura de datos que permite almacenar un conjunto de datos.
- Otros Nombre:
  - Vector, matriz, tabla o matriz unidimensional, lista o arreglo.
- Composición:
  - **Elementos**.
    - Cada dato del array.
  - **Índices**.
    - Hace referencia a la posición en que hay un dato en el array o un hueco donde almacenar un dato.
    - Primer índice = 0.
    - Último índice = N-1.
- **Constructores**:
  - **new Array();**
    - Crea un array vacío sin longitud.
    - Útil cuando no se conoce la longitud inicial del array.
    - Sintaxis:
      - var nombre del objeto array = new Array();
    - Ejemplo:
      - var menu = new Array();
  - **new Array(número);**

        - Crea un array vacío del número de posiciones especificadas, es decir, con la longitud indicada, pero con huecos o posiciones vacías.
        - Sintaxis:
            - var nombre del objeto array = new Array(número);
        - Ejemplo:
            - var menu = new Array(3);
    - **new Array(valor1, valor2,…,valor N);**
        - Crea un array con tantas posiciones como valores se hayan incluido como parámetros.
        - Cada posición o hueco se rellenará con el valor especificado en el mismo orden.
        - Sintaxis:
            - var nombre del objeto array = new Array(valor1, valor2,…,valor N);
        - Ejemplo:
            - var menu = new Array("Paella", "Pollo al ajillo", "Flan");
    - **[]** o [**valor1, valor2, … , valor N].**
        - No es un constructor, equivale a crear directamente una instancia literal,
        - Sintaxis:
            - var nombre del objeto array = [];
            - var nombre del objeto array = [valor1, valor2, … , valor N];
        - Ejemplos:
            - var menu = [];
            - var menu = ["Paella", "Pollo al ajillo", "Flan"];
            - var números = [10, 45.78,78, 90, 34.23, 89];

Introducir elementos en un Array.

- Uso del operador [].
- Si en una posición hay un dato
- Sintaxis:
    - nombre del array[índice] = valor;
- Ejemplos:

```
var menu = new Array(3);
menu[0] = "Paella";
menu[1] = "Pollo al ajillo";
menu[2] = "Flan";

var menu = new Array(); // También valdría: var menu = [];
menu[0] = "Paella";
menu[1] = "Pollo al ajillo";
menu[2] = "Flan";
menu[3] = "Natillas";
menu[4] = "Arroz con leche";
menú[5]= 22.50;
```

Eliminar elementos de un Array.

- Se asigna un valor nulo a su contenido.
- Se añade una cadena vacía.
- Sintaxis:

```
nombre del array[índice] = null;
```

nombre del array [índice] = “”;

Acceso a elementos de un Array.

- Para mostrar los elementos de un array u operar con su contenido.
- Sintaxis:
  - nombre del array[índice];
- Ejemplo:
  - document.write(“El primer plato de hoy es: “ + menu[0]);
  - var plato1 = menu[0];
  - document.write(“El primer plato de hoy es: “ + plato1);
  - var calculo = numeros[6] + numeros[20];

Propiedades de Arrays.

length.

- Única propiedad para los arrays.
- Muestra la longitud o número de posiciones de un array.
- Sintaxis:
  - nombre del array.length;
- Ejemplo:
  - document.write(“En el array menú hay “ + menu.length + “posiciones”);

Métodos.

sort().

- Muestra, ordenado alfabéticamente, el contenido de un array.
- Modifica el array al ordenarlo.
- Sintaxis.
  - nombre del array.sort();
- Ejemplo.
  - document.write(datos5.sort());

join().

- Muestra un listado de los valores que hay en el array intercalando el carácter separador que se le especifique.
- Si no se incluye un separador, se utiliza por defecto como separador la coma.
- Sintaxis:
  - nombre del array.join(“separador”);
- Ejemplo:
  - numeros.join(“/”);

reverse();

- Invierte el orden de los valores de un array.
- Modifica el array al invertir los valores.
- Sintaxis:
  - nombre del array.reverse();
- Ejemplo:
  - numeros.reverse();

push();

- Añade 1 o más valores al final de un array.

- Si se incluye push() dentro de document.write muestra el número total de elementos que tiene el array incluidos los nuevos.
- Sintaxis:
    - nombre del array.push(valor);
    - nombre del array.push(valor 1, valor 2,…, valor, N);
- Ejemplo:
    - var numeros = new Array(34,67,45);
    - numeros.push(20); Resultado → 34, 67, 45, 20
    - numeros.push(20,10,50); Resultado → 34, 67, 45, 20, 10, 50

pop().

- Elimina el último elemento de un array.
- Si se incluye pop() dentro de document.write muestra el valor del elemento a eliminar.
- Sintaxis:
    - nombre del array.pop();
- Ejemplo:
    - var numeros = new Array(34,67,45);
    - numeros.pop(); Resultado → 34, 67

unshift().

- Añade 1 o más valores al principio de un array.
- Si se incluye unshift() dentro de document.write muestra el número total de elementos que tiene el array incluidos los nuevos.
- Sintaxis:
    - nombre del array.unshift(valor);
    - nombre del array.unshift(valor 1, valor 2,…, valor, N);
- Ejemplo:
    - var numeros = new Array(34,67,45);
    - numeros.unshift(20); Resultado → 20, 34, 67, 45
    - numeros.unshift(20,10,50); Resultado → 20, 10, 50, 34, 67, 45

shift()

- Elimina el primer elemento de un array.
- Si se incluye shift() dentro de document.write muestra el valor del elemento a eliminar.
- Sintaxis:
    - nombre del array.shift();
- Ejemplo:
    - var numeros = new Array(34,67,45);
    - numeros.shift(); Resultado → 67,45

indexOf().

- Muestra el índice del elemento especificado.
- Si los datos son cadenas de caracteres se incluyen en el método con comillas.
- Si no existe, muestra el valor -1.
- Sintaxis:
    - nombre del array.indexOf(“valor”);
- Ejemplo:
    - var numeros = new Array(34,67,45);
    - numeros.indexOf(34); Resultado → 0
    - numeros.indexOf(45); Resultado → 2
    - numeros.indexOf(89); Resultado → -1

splice().

- Permite insertar, borrar o modificar los elementos de un array en una posición especifica.
- Sintaxis:
  - Eliminar:
    - nombre del array.splice(posición primer elemento a eliminar inclusive, número de elementos a eliminar);
  - Insertar eliminando elementos: (modifica sustituyendo).
    - nombre del array.splice(posición de inserción, número de elementos a eliminar, valor a insertar 1, valor a insertar 2,..valor a insertar N);
  - Insertar sin eliminar elementos: (Insertar desplazando y añadiendo delante los nuevos elementos).
    - nombre del array.splice(posición delante de la cual se insertarán los nuevos elementos, 0, valor a insertar 1, valor a insertar 2,..valor a insertar N);
- Ejemplo:

```
var colores = new Array("Rojo", "Verde", "Azul", "Violeta", "Naranja", "Amarillo", "Marrón");
document.write(colores.join()+ "<br>");
document.write(colores.splice(1,2,"Rosa","Blanco","Negro")+ "<br>"); //Se insertan los colores “Rosa","Blanco","Negro y se eliminan los colores "Verde", "Azul" (2 elementos a partir de la posición 1 a eliminar).
document.write(colores.splice(1,0,"Rosa","Blanco","Negro")+ "<br>"); //Se insertan los colores “Rosa","Blanco","Negro y no se eliminan otros colores, desplazándose los existente para dejar paso a los nuevos.
document.write(colores.splice(2,1)+ "<br>");    // Se elimina un color a partir de la posición 2 inclusive.
```

slice().

- Permite copiar un array o mostrar los elementos que se especifiquen.
- Sintaxis:
  - Copiar un array a una variable:
    - nombre del array.slice();
  - Mostar los valores de un array entre una posición inicial y otra final -1.
    - nombre del array.slice(posición inicial inclusive, posición final no incluida);
  - Mostar todos los valores de un array a partir de la posición especificada.
    - nombre del array.slice(posición inicial inclusive);

concat().

- Permite unir o concatenar un array a otro u otros.
- Al array que utilice la función, se le añaden a continuación de sus datos, los datos de los arrays a concatenar.
- Sintaxis:
  - nombre del array.concat(nombre array 1, nombre array 2,…..);
- Ejemplo:
  - let numeros1 = new Array(34,67,45);
  - let numeros2 = new Array(80,32,27);
  - let palabras = new Array(“Hola”, “Adiós”);
  - numeros1.concat(palabras); -> Resultado: 34,67,45, “Hola”, “Adiós”.
  - palabras.concat(numeros1); -> Resultado: “Hola”, “Adiós”, 34,67,45.
  - numeros1.concat(palabras, numeros2, “hola”,89); -> Resultado: 34,67,45, “Hola”, “Adiós”, 80, 32, 27,”hola”,89.

includes().

- Determina si un dato o elemento está incluido en el array especificado.
- Devuelve un valor booleano:
    - **true**.
        - Elemento incluido.
    - **false**.
        - Elemento no incluido.
- Sintaxis:
    - nombre del array.includes(valor o dato);
- Ejemplo:
    - let palabras = new Array("Hola", "Adiós");
    - palabras.includes("Hola"); -> Resultado: ***true***.
    - palabras.includes("Buenos días"); -> Resultado: ***false***.

forEach().

- Llama a la función pasada como parámetro para cada elemento del array.
- Funciona como un bucle.
- Sintaxis:
    - nombre del array.forEach(función a ejecutar por cada elemento);

- **Función pasada como parámetro**.
    - Sintaxis:

```
function nombre de la función (valor, índice, array)
        {
            ínstrucciones;
        }
```

    - Parámetros**:**
        - **valor.**
            - Representa a cada valor o elemento contenido en el array.
            - Parámetro obligatorio.
        - **índice.**
            - Valor representa a los índices de los elementos contenidos en el array.
            - Parámetro opcional.
        - **array.**
            - Representa al array que está siendo recorrido.
            - Parámetro opcional.

    - Ejemplo:

```
var nombres = new Array ("Ana", "Juan", "Pepe");
nombres.forEach(funcionArray);
function funcionArray(valor, índice)
        {
            document.write("En la posición " + indice + " está el nombre " + valor + "<br>");
        }
```

Recorrer un Array.

Mostrar todos los elementos de un Array.

- Se puede recorrer un array y mostrar todos sus elementos utilizando su nombre sin índices.
- El contenido se muestra en una línea separándose cada elemento con una coma.
- Sintaxis:
    - document.write(nombre del array);
- Ejemplo:
    - var colores = new Array (“Rojo”, ”Verde”, ”Azul”);
    - document.write(“ el contenido del array colores es: “ + colores);

Mostrar todos los elementos de un Array usando bucles.

Uso del Bucle for.

- Se usa la variable contadora para dar vueltas en el bucle y también como índice para el array.
- Muestra el valor “undefined” en posiciones sin nada o vacías.
- Sintaxis:
    - Con propiedad length:

```
var contadora o índice;
for (indice = 0; índice < nombre del array.length; índice++)
{
        document.write(nombre del array[índice]);
}
```

    - Con valor conocido para la condición del bucle:

```
var contadora o índice;
for (indice = 0; índice < longitud del array; índice++)
{
        document.write(nombre del array[índice]);
}
```

Uso del Bucle for…in.

- Se usa la variable contadora para dar vueltas en el bucle y también como índice para el array.
- No muestra posiciones vacías con valor “undefined”.
- Sintaxis:

```
var contadora o índice;
for (variable contadora o índice in nombre del array)
{
        Instrucciones;
}
```

- Ejemplo**:**

```
let rios = new Array();
rios[0]="Ebro";
rios[2]="Guadalquivir";
var i;
for (i in rios)
{
        document.write(“Rios:” + rios[i]);
}
```

Este bucle equivaldría a:

```
var i;
for (i = 0; i < rios.length; i++)
{
        document.write("Rios:"+ rios[i]);
}
```

Uso del Bucle for…of.

- Se usa la variable contadora para dar vueltas en el bucle y también como índice para el array.
- Muestra posiciones vacías con valor "undefined".
- Los valores se muestran especificando sólo el nombre de la variable contadora.
- Los índices se muestran usando la función indexOf().
- Sintaxis:

```
var contadora o índice;
for (variable contadora o índice of nombre del array)
{
        Instrucciones;
}
```

- Ejemplo:

```
var i;
for (i of numeros)
{
        document.write(i);  // Muestra datos contenidos en el  array.
        document.write(numeros.indexOf(i));  // Muestra índices del array.

}
```

Uso de la función o método forEach().

- Visto en la sección anterior de métodos de los arrays.

Cargar datos en un array usando el método prompt.

- Directamente:
  - nombre array [índice] = prompt("Mensaje","");
  - Ejemplo:
    - colores[3] = prompt("Intro un color","");
    - colores[10] = prompt("Intro otro color","");
- Con bucle for:

```
var contadora o índice;
for (indice = 0; índice < nombre del array.length; índice++)
{
        var nombre de variable que carga dato = prompt("Mensaje","");
        nombre del array[índice] = nombre de variable que carga dato;
        //nombre del array[índice] = prompt("Mensaje",""); //Alternativa para no
        crear una variable.
}
        document.write(nombre del array[índice]);
Ejemplo:
        var datos4 = new Array (3);
        var i, valor;
        for (i=0; i<datos4.length; i++)
```

```
{
        //valor = prompt("Introduce valor para el array","");
        //datos4[i] = valor;
        datos4[i] = prompt("Introduce valor para el array","");
document.write("En la posicion "+ i + " el valor del array es " + datos4[i] +"<br>");
}
```

- Con bucle for…of:
  - Carga los datos de forma temporal, éstos no se mantienen en el array.
  - Los datos introducidos sólo se muestran mientras se ejecuta le bucle.

```
var contadora o índice;
for (indice of nombre del array)
{
        var nombre de variable que carga dato = prompt("Mensaje","");
        nombre del array[índice] = nombre de variable que carga dato;
        //nombre del array[índice] = prompt("Mensaje",""); //Alternativa para no crear una variable.
        document.write(nombre del array[índice]);

}
```

Ejemplo:

```
        var datos4 = new Array (3);
        var i, valor;
        for (i of datos4)
{
        //valor = prompt("Introduce valor para el array","");
        //datos4[i] = valor;
        datos4[i] = prompt("Introduce valor para el array","");
        document.write("Elementos contenidos en el array " + datos4[i] +"<br>");
}
```

UF1306. PRUEBAS DE FUNCIONALIDADES Y OPTIMIZACIÓN DE PÁGINAS WEB.

EXPRESIONES REGULARES.

Concepto.

- Conjunto de caracteres que permiten construir un patrón.

Utilidad.

- Buscar patrones o coincidencias dentro de un texto o cadena.
- Comprobar si una cadena tiene un formato concreto. (DNI).
- Realizar una misma operación sobre un grupo determinado de palabras.
- Sustitución de caracteres.
- Crear funciones de validación.

Formas de crear una expresión regular.

- Definir un patrón literal que se puede o no almacenar en una variable.
  - var nombre = /patrón/;

- Utilizar la clase u objeto prototípico RegExp.
    - var nombre = new RegExp("patrón");
- Sitio web para probar expresiones regulares:
    - https://regexr.com/

CREACIÓN DE UNA EXPRESIÓN REGULAR CON PATRÓN LITERAL

- Se construye como una cadena de caracteres encerrada entre barras inclinadas de las siguientes formas:
- **Patrón Simple.**
    - Texto o conjunto de caracteres para el que se buscan coincidencias directas en otro texto.
    - Ejemplo:
        - Texto:
            - "El pájaro saltó del nido"
        - Patrón:
            - /el/
        - Coincidencias:
            - Una porque los patrones distinguen mayúsculas de minúscula.
- **Patrones Complejos**.
    - Para crear patrones complejos y así ampliar o restringir su ámbito de acción se usan estructuras y caracteres especiales.
    - **Tipos según funcionalidad**.
    - **Caracteres de repetición o cuantificadores**.
        - Permiten que en el patrón algunos caracteres pueden repetirse un número determinado de veces.
        - Caracteres:
            - **Asterisco (*).**
                - Indica que el carácter que le precede puede aparecer cero, una o más veces.
                - Ejemplo:
                    - /mira*/
                    - Coincidencias:
                        - mir, mira, miraa, miraaaaaa.
            - **Más(+).**
                - Indica que el carácter que le precede puede aparecer una o más veces.
                - Ejemplo:
                    - /mira+/
                    - Coincidencias:
                        - mira, miraa, miraaaaaa.
            - **Interrogación (?).**
                - Indica que el carácter que le precede puede aparecer cero o una sola vez.
                - Ejemplo:
                    - /mira?/
                    - Coincidencias:
                        - mir, mira.
            - **{número entero positivo}**

        - Indica que el carácter que le precede se debe repetir el número de veces especificado.
        - Ejemplo:
            - /mira{2}/
            - Coincidencias:
                - miraa.
    - **{número entero positivo,}**
        - Indica que el carácter que le precede se debe repetir el número de veces especificado o más veces.
        - Ejemplo:
            - /mira{2,}/
            - Coincidencias:
                - mraa, mraaaaaaaaaa.
    - **{número entero positivo 1, número entero positivo 2}**
        - Indica que el carácter que le precede se debe repetir el número de veces definidos entre ambos números inclusive.
        - Ejemplo:
            - /mira{2,4}/
            - Coincidencias:
                - miraa, miraaa, miraaaa.
    - **Combinación de caracteres.**
        - En un único patrón o expresión se pueden incluir más de un carácter de repetición.
        - Ejemplo:
            - /mi+ra*/
                - La i puede repetirse 1 o más veces.
                - La “a” puede repetirse ninguna, 1 o más veces.
            - Coincidencias:
                - mir, miir, mira, miraaaa
- **Caracteres especiales.**
    - Para que JavaScript puede reconocerlos se deben incluir con un carácter de escape (\), escrito antes que el carácter a escapar.
    - Caracteres:
    - **\n**
        - Salto de línea.
        - Ejemplo:
            - /hola\n adiós/
    - **\t**
        - Tabulador.
        - Ejemplo:
            - /hola\t adiós/
    - **\r**
        - Retorno de carro.
        - Salto a la primera posición de una línea.
        - Ejemplo:
            - /hola\r adiós/
    - **\v**

- Tabulador vertical.
- Ejemplo:
  - /\vhola /

- **\uxxxx**
  - Coincidencia con el carácter Unicode definido por código de 4 dígitos hexadecimales representados por xxxx.
  - Página con todos los valores Unicode:
    - https://unicode-table.com/es/
  - Ejemplo:
    - /\u00f1/ representa al carácter ñ.
- **\b**
  - Separador de palabra que puede ser un espacio en blanco o un salto de línea.
  - Ejemplo:
    - /Java\b Script/
    - Java seguido de un espacio en blanco o salto de línea y Script.
  - También se puede utilizar para indicar que un texto empiece o termine por los caracteres especificados.
  - Ejemplo:
    - var patron5A = /or\b/; Texto que termine por "or".
    - var patron5B = /\borde/; Texto que empiece por "orde".
    - var texto5bis = "ordenador"; //Coincidencia con patron5A y con el patron5B.
    - var texto5bis = "elordenador"; //Sin coincidencias con patron5B y si con patron5A.
- **\B**
  - Separador de palabra con un carácter que no sea un espacio en blanco o un salto de línea.
  - Ejemplo:
    - /Java\BScript/
    - Coincidencias:
      - JavaScript , Java8Script, Java*Script,…
- **Punto(.)**
  - Comodín.
  - Cualquier carácter excepto un salto de línea.
  - Ejemplo:
    - /.s/
    - Coincidencias:
      - as, es, 3s, -s
- **\s**
  - Carácter de separación que representa a tabulador, espacio o salto de línea.
  - Ejemplo:
    - /Java\sScript/
    - Coincidencias:
      - Java Script, Java Script, Java [Salto Línea] Script

\S

- Cualquier carácter de separación que no sea un tabulador, espacio o salto de línea.
- Ejemplo:
  - /Java\SScript/
  - Coincidencias:
  - Java9Script, Java/Script, Java*Script, JavazScript.

- **\d**
  - Digito entre 0 y 9.
  - Ejemplo:
    - /Java\dScript/
    - Coincidencias:
      - Java9Script, Java7Script, Java6Script.
- **\D**
  - Carácter separador que no sea un digito.
  - Ejemplo:
    - /Java\DScript/
    - Coincidencias:
      - Java_Script, Java-Script, Java,Script.
- **\w**
  - Cualquier carácter alfanumérico (letras, números y guion de subrayado), como separador.
  - Ejemplo:
    - /Java\wScript/
    - Coincidencias:
      - JavanScript, JavaMScript, Java7Script, Java_Script.
- **\W**
  - Cualquier carácter que no sea un alfanumérico o un guion de subrayado.
  - Ejemplo:
    - /Java\WScript/
    - Coincidencias:
      - Java-Script, Java/Script, Java Script.
- **Combinaciones de varios caracteres especiales**.
  - Se pueden combinar o repetir varios caracteres especiales en un patrón.
  - Ejemplo:
    - /\d\d-\d\d-\d\d\d\d/
    - /\w\w\w\w/
  - Coincidencia:
    - Una fecha como 23-07-1999.
    - java, Java, JAVA, ja7a, Luis, Pep_

- **Agrupación de valores.**
  - Permiten encontrar coincidencias con palabras que son muy similares, pero que se diferencia en una pequeña parte.
  - Caracteres:
  - **[xxx]**
    - Coincidencia con uno solo de los caracteres especificados entre corchetes.
    - Se pueden usar guiones para separar los extremos de un rango de caracteres contiguos

- Ejemplos:
    - /[abc]a/ equivale a /[a-c]a/
    - /[pmr]alo/
    - /[m-z]aes/
    - /[a-zA-Z0-9]ma/
    - /gat[ao]/
    - /pa[lt]o/
    - /[a-z][0-3][f-t]mod/
    - /[a-z]{7}/ -> Equivale a [a-z] [a-z] [a-z] [a-z] [a-z] [a-z] [a-z]
- Coincidencias:
    - aa, ba, ca.
    - malo, palo, ralo.
    - maes, zaes, taes,
    - ama, Ama, 0ma, 6ma, Zma.
    - gato, gata.
    - palo, pato.
    - t2gmod es válido, a9tmod no es válido.
    - djhassf es válido, dfdj no es válido, dfr342k no es válido.

- **[^xxx]**
    - Coincidencia con caracteres que no sean los especificados entre corchetes.
    - Se pueden usar guiones para separar los extremos de un rango de caracteres contiguos
    - Ejemplos:
        - /[^abc]a/ equivale a /[^a-c]a/
        - /[^pmr]alo/
        - /[^m-z]aes/
        - /[^a-zA-Z0-9]ma/
        - /gat[^ao]/
        - /pa[^lt]o/
    - Coincidencias:
        - ta, va, na.
        - valo, aalo, talo.
        - caes.
        - +ma, *ma, /ma.
        - gati, gatu.
        - paro, paso.
- **[x|y]**
    - Equivale al operador O lógico.
    - Coincidencia con x o y, pero no con los dos**.**
    - Ejemplos:
        - /gat[a|o]/
        - /gata|gato/
    - Coincidencias:
        - gata, gato.

- **Caracteres de posición.**
    - Permiten especificar en qué posición debe existir la coincidencia con el patrón.
    - Caracteres:

- **Acento circunflejo (^).**
  - Hace que el patrón tenga que coincidir desde el inicio de la palabra o línea.
  - Se coloca al inicio del patrón fuera de corchetes.
  - Ejemplo:
    - /^b/
    - /^bu/
    - /^[buú]/
  - Coincidencias:
    - buenos, buenos días, borrador (primer ejemplo).
    - buenos, buenos días (segundo ejemplo).
    - buenos, únicos, unidos (tercer ejemplo).
- **Dólar ($).**
  - Hace que el patrón tenga que coincidir desde el final de la palabra o línea.
  - Se coloca al final del patrón fuera de corchetes.
  - Ejemplo:
    - /a$/
    - /as$/
    - /[er]$/
  - Coincidencias:
    - casa, terraza (primer ejemplo).
    - buenas, personas (segundo ejemplo).
    - presidente, leer, comer (tercer ejemplo).
- **(?=n)**
  - Hace que el patrón coincida con cualquier palabra que tenga la cadena **n** a continuación.
  - **n** puede ser una palabra un carácter u otro patrón.
  - Siempre va encerrado en te paréntesis.
  - Ejemplo:
    - /gat(?=o)/
    - / gato(?= bonito)/
    - /gat(?=[ao])/
  - Coincidencias:
    - gato (primer ejemplo).
    - Gato bonito.
    - gato, gata.
- **(?!n)**
  - Hace que el patrón no coincida con cualquier palabra que tenga la cadena **n** a continuación.
  - **n** puede ser una palabra un carácter u otro patrón.
  - Siempre va encerrado entre paréntesis.
  - Ejemplo:
    - /gat(?!o)/
    - / gato(?! bonito)/
    - /gat(?![ao])/
  - Coincidencias:
    - gata (primer ejemplo).

        - gato blanco.
        - gatu.
- **Modificadores (flags o banderas).**
    - Modifican algunas características al buscar coincidencias.
    - Se escriben al final después de /.
    - Se pueden usar.
    - Caracteres:
        - **g**
            - Fuerza a que se sigan buscando coincidencias después de encontrar la primera.
            - Ejemplo:
                - /ja/g
            - Coincidencias:
                - ja, jajeja, Jajeja, jamón, jarra.
        - **i**
            - Elimina la distinción entre mayúsculas y minúsculas.
            - Ejemplo:
                - /ja/i
            - Coincidencias:
                - ja, jajeja, Jajeja, jamón, Jarra, alhaJa.
        - **s**
            - Permite que el carácter especial punto (.) pueda ser un salto de línea.
            - Ejemplo:
                - /orde.nador/s
            - Coincidencias:
                - orde
                - nador
        - **x**
            - Fuerza a que los espacios en blanco sean ignorados.
            - Ejemplo:
                - /buenos días/x
                - /buenos días/x
                - /buenos días/x
                - /buenosdías/x
            - Coincidencias:
                - buenosdías
        - **m**
            - Permite usar el patrón en varias líneas.
            - Busca en cadenas con retorno de carro.
            - Ejemplo:
                - /buenos/m
            - Coincidencias:
                - buenos (línea 1).
                - buenos (línea 3).
- **Paréntesis**.
    - Permiten agrupar caracteres que serán tratados como un bloque.

        - Con el objeto RegExp sirven también para recuperar el valor de cada bloque.
        - Se pueden crear varios bloques usando varios paréntesis.
        - Otros caracteres especiales se pueden aplicar a los bloques para modificar la funcionalidad del conjunto.
        - Utilidad:
            - Usar un patrón para reemplazar texto.
        - Ejemplo:
            - /sum(ar)+/
            - /sum(ar)?/
            - /(sum)+(ar)+/
            - /ja+/
            - /(ja)+/
        - Coincidencias.
            - sumar, sumarar, sumararar.
            - sum, sumar.
            - sumar, sumsumar,sumsumsumarar.
            - ja, jaa, jaaa.
            - ja, jaja, jajaja.
- **Expresiones regulares útiles.**
    - **/\d{9}/** Número de teléfono**.**
    - **/\d{8}[a-zA-Z]/** DNI.
    - **/\d{2}-\d{2}-\d{4}/** Fecha (dd-mm-aaaa / mm-dd-aaaa)**.**
    - **/[0-3]\d-[0|1]\d-\d{4}/** Fecha (dd-mm-aaaa)**.**
    - **/^(0[0-9]|1[0-9]|2[0-3]):([0-5]\d)(:([0-5]\d))?$/** Hora (hh:mm:ss formato 24 horas con y sin segundos)**. 23:12 23:12:33**
    - **/^(0[0-9]|1[0-2]):([0-5]\d)(:([0-5]\d))?$/** Hora (hh:mm:ss formato 12 horas con o sin segundos)**.**

CREACIÓN DE UNA EXPRESIÓN REGULAR CON EL OBJETO RegExp.

- Se usa para trabajar con expresiones regulares.
- Constructor.
    - Permite crear una instancia que contendrá la expresión regular o patrón a aplicar a un texto.
    - Sintaxis:
        - var nombre instancia patron = new RegExp(“patrón”, “modificadores”);
        - Patrón.
            - Cadena que representa la expresión regular sin barras (/), y entre comillas.
            - También puede ser una variable que contenga la expresión.
        - Modificadores:
            - Son opcionales.
            - Son g, i, s, m, x.
        - Si se usan caracteres especiales en la cadena, hay que introducirlos con doble barra inclinada como, por ejemplo, \\d en lugar de \d.
    - Ejemplos con patrón literal:
        - var patron1 = new RegExp(“hola+”); hola,holaa,holaa.
        - var patron2 = new RegExp(“hola+”,”i”);Hola, HOLA, holAAA.
        - var patron3 =new RegExp(“^A”); Que empiecen por A.

        - var patron4 =new RegExp("\\d\\d"); Números de 2 dígitos: 22, 67, 45.
        - var patron5 =new RegExp("\\d{2}-\\d{2}-\\d{4}"); Fecha: 23-09-2021.
    - Ejemplos con patrón literal en una variable:
        - var exp;
        - exp = /[a-z]t/;
        - var patron6 =new RegExp(exp); Palabras de 2 letras minúsculas: at, rt, mt.
        - var patron7 =new RegExp(exp,"i"); Palabras de 2 letras sin distinguir mayúsculas-minúsculas: at, rt, mt, AT, aT.
- Propiedades.
    - La mayor parte se centrar en comprobar si los modificadores especificados en el constructor están o no activados.
    - Sintaxis:
        - nombre instancia.propiedad;
    - Propiedades:
        - **global**.
            - Indica si el modificador "g" está activo en el patrón.
        - **ignoreCase**.
            - Indica si el modificador "i" está activo en el patrón.
        - **multiline.**
            - Indica si el modificador "m" está activo en el patrón.
        - **flags.**
            - Devuelve una cadena con los modificadores presentes en el constructor.
        - **source**.
            - Devuelve la cadena correspondiente al patrón escrito.
        - **index.**
            - Representa la posición (índice), dentro del texto, donde empieza el primer carácter de la coincidencia encontrada.
            - Por defecto valor 0.
            - Se actualiza automáticamente tras usar un método de búsqueda de RegExp si se tiene activado el modificador "g".
            - También se puede asignar un valor manualmente.
            - Debe incluirse junto con el método exec y el patrón de búsqueda:
                - patron.exec(texto).index;
        - **lastIndex.**
            - Representa la posición (índice), dentro del texto, del último carácter de la coincidencia encontrada.
            - Por defecto valor 0.
            - Se actualiza automáticamente tras usar un método de búsqueda de RegExp si se tiene activado el modificador "g".
            - También se puede asignar un valor manualmente.
        - **dotAll**.
            - Indica si el carácter de punto (.) del patrón de una expresión normal coincide con un salto de línea (\n).
            - Hay que incluir el modificador "s" en la expresión para que se obtenga un valor verdadero.
- Métodos.
    - Permiten probar si existen coincidencias de un patrón dentro de un texto.

- Sintaxis:
  - nombre instancia.metodo();
- Métodos:
  - **test().**
    - Permite aplicar el patrón sobre el parámetro “texto” o la variable que lo contenga.
    - Muestra verdadero o falso según si hay o no coincidencia.
    - Sintaxis:
      - patron.test(“texto”);
      - patron.test(variable con texto);
    - Ejemplos con variables y sin objeto:
      - var palabras = ”Hola”;
      - var patron = /la/gi;
      - document.write(“¿Hay coincidencia?” + patron.test(“Hola”));
      - document.write(“¿Hay coincidencia?” + patron.test(palabas));
    - Ejemplos con variables y con objeto:
      - var palabras = ”Hola”;
      - var patron = /la/gi;
      - var expresion = new RegExp(patron);
      - document.write(“¿Hay coincidencia?” + expresion.test(palabras));
    - Ejemplos sin variables y con objeto:
      - var expresion = new RegExp(“la”,”gi”);
      - //document.write(“¿Hay coincidencia?” + expresion.test(“Hola”));
    - Ejemplos con if..else, variables y sin objeto:

```
var patron = /la/;
var texto = "la cigala";
if (patron.test(texto))

{
        document.write("Si");
}
else
{
        document.write("No");
}
```

    - Ejemplos con if..else, variables y con objeto:

```
var patron = /la/;
var texto = "la cigala";
var expresion = new RegExp(patron);
if (expresion.test(texto))

{
        document.write("Si");
}
else
{
        document.write("No");
}
```

- **exec().**
    - Permite aplicar el patrón sobre el parámetro “texto” o la variable que lo contenga para mostrar:
        - null, si no hay coincidencia.
        - La primera coincidencia, si la hay.
        - Para mostrar más coincidencias hay que usar el modificador “g”.
        - Si “g” está activo, actualiza la propiedad lastIndex y deja todo preparado para que la siguiente ejecución del método exec() comience en esa posición y así, a partir de ella busque una nueva coincidencia.
    - Sintaxis:
        - patron.exec(“texto”);
        - patron.exec(variable con texto);
    - Ejemplo:

```
var patron = new RegExp(“A”,”gi”);
document.write(patron.exec(“Hola Adiós”)); Encuentra la letra a.
document.write(patron.lastIndex); Muestra índice 4.
document.write(patron.exec(“Hola Adiós”)); Encuentra la letra A.
document.write(patron.lastIndex); Muestra índice 6.
document.write(patron.exec(“Hola Adiós”)); Sin coincidencias, muestra null.
document.write(patron.lastIndex); Muestra índice 0.
```

CLASE U OBJETO PROTOTÍPICO String.

- Permite realizar diversas operaciones sobre cadenas de caracteres.
- Constructor.
    - Permite crear una instancia que contendrá una cadena de caracteres.
    - Sintaxis:
        - var nombre del objeto u instancia = new String(valor);
        - Valor:
            - Cadena de caracteres (texto, patrón, expresión regular, etc.) que se pasará al objeto creado.
            - Si se omite el valor, se crea una cadena vacía.
    - Ejemplos:
        - var texto1 = new String (); Cadena vacía.
        - var texto2 = new String (hola); “hola”.
        - var texto3 = new String (20); “20”.
        - var texto4 = new String (false); “false”.
- Propiedades.
    - **length**.
        - Muestra el número de caracteres que tiene la cadena, incluidos espacios en blanco y caracteres especiales.
        - Sintaxis:
            - nombre del objeto u instancia.length;
            - variable con cadena de caracteres.length;

        - “texto literal”.length;
    - Ejemplos:
        - document.write(texto1.length); Muestra 0.
        - texto1 = “adiós”;
        - document.write(texto1.length); Muestra 5.
        - document.write(texto2.length); Muestra 4.
        - document.write(“hola”.length); Muestra 4.

- Métodos.
    - **match():**
        - Busca las coincidencias de un patrón o expresión regular dentro de un texto y las devuelve en forma de array o con el valor **null** si no hay ninguna.
        - Sintaxis:
            - texto.match(patrón);
            - texto.match(variable con el patrón);
            - var variable o array con coincidencias = texto.match(patrón);
            - var variable o array con coincidencias = texto.match(variable con el patrón);
        - Ejemplos:
            - var texto = new String (“Coche”);
            - var patron=/a/;
            - document.write (texto.match(/a/)); Devuelve null.
            - document.write(texto.match(/c/)); Devuelve c minúscula que se puede almacenar en una variable o en un array.
            - document.write(texto.match(/c/gi)); Devuelve C y c que se pueden almacenar en un array.
            - var coincidencias = texto.match(/c/gi);
            - document.write (coincidencias); Devuelve C, c.
            - document.write (coincidencias[1]); Devuelve c.
            - document.write (coincidencias.length); Devuelve 2.
            - texto = “Coche de carreras”;
            - document.write (coincidencias.length); Devuelve 3.
            - for(var i = 0; i < 2; i++)// Correcto si no hay cambios en los textos que muestren más o menos coincidencias.
            - for(var i = 0; i < coincidencias.length; i++)

```
{
        document.write (coincidencias[i]);
}
```

- **search().**
    - Devuelve el índice o posición donde está la primera ocurrencia del patrón, tras comprobar si existe en el texto.
    - Primer índice es el 0.
    - Si no hay coincidencias devuelve -1.
    - No permite búsquedas globales (g).
    - Sintaxis:
        - texto.search(patrón);

        - texto.search(variable con el patrón);
        - var variable para el índice = texto.search(patrón);
        - var variable para el índice = texto.search(variable con el patrón);

    - Ejemplos:
        - var texto = "Esta Mancha no mancha";
        - var patron1 = /mancha/;
        - var patron2 = /Mancha/;
        - var patron3 = /mancha/gi;
        - var patron4 = /manta/;
        - var busqueda1 = texto.search(patron1); Devuelve 15.
        - var busqueda2 = texto.search(patron2); Devuelve 5.
        - var busqueda3 = texto.search(patron3); Devuelve 5.
        - var busqueda4 = texto.search(patron4); Devuelve -1.

- **replace().**
    - Realiza una búsqueda y si encuentra una coincidencia se produce un reemplazo.
    - Si no hay coincidencias no modifica el texto.
    - Si se usa el modificador g se reemplazan todas las coincidencias que haya en el texto.
    - Si no se usa el modificador g se reemplaza sólo la primera coincidencia encontrada en el texto.
    - Sintaxis:
        - texto original.replace(búsqueda, reemplazo).
        - var variable con texto modificado = texto original.replace(búsqueda, reemplazo).
        -
        - Parámetros:
            - Búsqueda.
                - Texto, expresión regular o patrón.
            - Reemplazo.
                - Texto literal, variable con texto literal, objeto de tipo String o función.

- **split().**
    - Divide una cadena en un subconjunto de cadenas (subcadenas).
    - Las subcadenas se guardan en un array.
    - Sintaxis:
        - texto.split (separador, límite);
        - var nombre de un array = texto.split (separador, límite);
    - parámetros:
        - **separador**.
            - Carácter, espacio en blanco, expresión regular o patrón, o texto que actúa como delimitador de las subcadenas.
            - Si no hay separador o delimitador en la cadena, se devuelve un array con la cadena completa.
        - **límite**.
            - Opcional.
            - Es un numero entero.
            - Si se incluye se devuelven tantas subcadenas como se indique con el número.

- **toLowerCase().**
    - Muestra un texto en minúsculas.
    - Sintaxis:

        - nombre del objeto u instancia.toLowerCase();
        - variable con cadena de caracteres.toLowerCase();
        - “texto literal”. toLowerCase();
        - función().toLowerCase();
    - Ejemplos:
        - var texto = “La catedral de Burgos”;
        - document.write (texto.toLowerCase()); Devuelve “la catedral de burgos”.
        - var frase = prompt(“Escribe un texto”,””). toLowerCase();
- **toUpperCase().**
    - Muestra un texto en mayúsculas.
    - Sintaxis:
        - nombre del objeto u instancia.toUpperCase();
        - variable con cadena de caracteres. toUpperCase();
        - “texto literal”. toUpperCase();
        - función().toUpperCase();
    - Ejemplos:
        - var texto = “La catedral de Burgos”;
        - document.write (texto.toUpperCase()); Devuelve “LA CATEDRAL DE BURGOS”.
        - var frase = prompt(“Escribe un texto”,””). toUpperCase();
- **charAt()**.
    - Muestra el carácter especificado como parámetro.
    - El parámetro representa la posición que se especifica mediante un numero entero empezando por 0.
    - Si no se especifica muestra el carácter de la posición 0.
    - Si la posición tiene un valor no válido muestra una cadena vacía.
    - Sintaxis:
        - nombre del objeto u instancia.charAt(posición);
        - variable con cadena de caracteres. charAt(posición);
        - “texto literal”. charAt(posición);
    - Ejemplos:
        - var texto = “La catedral de Burgos”;
        - document.write (texto.charAt(0)); Devuelve “L”.
        - document.write (texto.charAt(8)); Devuelve “r”.
        -
- **concat().**
    - Une las cadenas pasadas como parámetro.
    - Equivale al operador de concatenación +.
    - Sintaxis:
        - nombre del objeto u instancia.concat(cadena1, cadena2,…,cadena N);
        - variable con cadena de caracteres. concat(cadena1, cadena2,…,cadena N);
        - “texto literal”. concat(cadena1, cadena2,…,cadena N);
    - Ejemplos:
        - var texto = “La catedral de Burgos”;
        - document.write (texto.concat(“ y la de Cuenca”));
- **includes().**
    - Devuelve un valor booleano (true, false).
    - Indica si un texto está incluido en otro texto.
    - Sintaxis:
        - nombre del objeto u instancia.includes(texto a buscar, posición);

        - variable con cadena de caracteres. includes(texto a buscar, posición);
        - “texto literal”. includes(texto a buscar, posición);
        - Posición:
            - Valor de posición a parir de la cual se iniciará la búsqueda.
            - Es opcional.
            - Valor por defecto el 0.
    - Ejemplos:
        - var texto = “La catedral de Burgos”;
        - document.write(texto.includes(“catedral”)); Devuelve true.
        - document.write(texto.includes(“catedral”,10)); Devuelve false.
- **indexOf() / lastIndexOf().**
    - Muestra la posición de la primera coincidencia(indexOF()), o última coincidencia (lastIndexOf()), de un texto dentro de una cadena.
    - Sintaxis:
        - nombre del objeto u instancia.indexOf(texto, inicio);
        - variable con cadena de caracteres. indexOf(texto, inicio);
        - “texto literal”. indexOf(texto, inicio);
        - nombre del objeto u instancia.lastindexOf(texto, hasta posición indicada);
        - variable con cadena de caracteres.lastindexOf(texto, hasta posición indicada);
        - “texto literal”. lastindexOf(texto, hasta posición indicada);
        - Texto:
            - Carácter o conjunto de caracteres.
            - Si no se encuentra devuelve -1.
            - Primer carácter de la cadena el 0.
        - Inicio:
            - Valor opcional, por defecto un 0.
            - Indica posición por la que empezar a buscar.
    - Ejemplos:
        - var texto = “JavaScript”;
        - texto.indexOf(“a”); Devuelve 1.
        - texto.indexOf(“a”,2); Devuelve 3.
        - texto.indexOf(“Script”); Devuelve 4.
        - texto.indexOf(“T”); Devuelve -1.
        - texto.lastIndexOf(“a”); Devuelve 3.
        - texto.lastIndexOf(“Script”); Devuelve 4.
        - texto.lastIndexOf(“R”); Devuelve -1.
- **substring().**
    - Muestra la cadena de caracteres ubicada ente una posición inicial y otra final.
    - Sintaxis:
        - nombre del objeto u instancia.substring(inicio inclusive, fin exclusive);
        - variable con cadena de caracteres.substring(inicio inclusive, fin exclusive);
        - “texto literal”. substring (inicio inclusive, fin exclusive);
        - Si no se incluye el valor final, muestra la cadena entera o la cadena entera a partir del inicio especificado.
        - Si no se incluye ningún valor como parámetro equivale a la posición inicial cero.
    - Ejemplos:
        - var texto = “JavaScript”;

        - texto.substring(0); Devuelve JavaScript.
        - texto.substring(0,2); Devuelve Ja.
- **repeat().**
    - Crea una cadena repitiendo, el número de veces que se indique, el contenido del objeto o variable con el texto.
    - Sintaxis:
        - nombre del objeto u instancia.repeat(número de repeticiones);
        - variable con cadena de caracteres.repeat(número de repeticiones); "texto literal". repeat(número de repeticiones);
        - Si el valor es cero o no se incluye ningún valor, muestra una cadena vacía.
    - Ejemplos:
        - var texto = "JavaScript";
        - texto.repeat(); Devuelve " ".
        - texto.repeat(2); Devuelve JavaScript JavaScript
- **trim().**
    - Elimina espacios en blanco en una cadena ubicados al principio o al final de ésta.
    - Sintaxis:
        - nombre del objeto u instancia.trim();
        - variable con cadena de caracteres.trim();
        - "texto literal". trim();
    - Ejemplos:
        - var texto = " JavaScript ";
        - texto.trim(); Devuelve "JavaScript ".

FUNCIONES DE VALIDACIÓN.

- Funciones que incluyen patrones o expresiones regulares.
- Pueden usarse bajo cualquier condición, pero es común que se llamen o ejecuten en respuesta a un evento.
- Se crean normalmente para validar formularios.
- Objetivos:
    - Evitar errores en los datos enviados.
    - Evitar modificaciones del código JavaScript malintencionadamente cuando la validación se realiza en el lado cliente.
- Sintaxis:
    - Hay varias formas construir una función de validación, todo dependerá de qué deberá hacer ésta, por ejemplo:
        - opción 1:

```
function nombre de la función (Con o sin parámetros)
{
        var variable = valor; // var variable = métodos get();
        var patron = expresión regular; // var patron = new regExp(expresión regular);
        return  patron.test(variable);
}
```

        - opción 2:

```
function nombre de la función (Con o sin parámetros)
```

```
{
        let variable = valor;  // let variable = métodos get();
        let patron = expresión regular; // let patron = new regExp(expresión regular);
        if(condición o condiciones a evaluar)
        {
                instrucciones;
        }
        else if (condición o condiciones a evaluar) (opcional)
        {
                instrucciones;
        }
        .
        .
        else (opcional)
        {
                instrucciones;
        }
}
```

- **Variable**.
  - La variable permite cargar un valor literal, un valor introducido por teclado, un valor producido por la ejecución de una función, un valor extraído del contenido de la página web, etc.
  - El contenido de dicha variable será el evaluado usando una expresión regular.
- **Patrón**.
  - Variable u objeto regExp donde se carga la expresión regular.
  - La expresión regular también puede utilizarse de forma literal.
- **Método test()** .
  - Este método en la función permite comprobar la validez del contenido de la variable respecto de la expresión regular.
  - Se puede retornar el resultado del método (verdadero o falso), para que con una expresión condicional se establezcan las acciones a realizar.
- **Métodos get().**
  - Conjunto de métodos que sirven para acceder al DOM y cargar contenido web en variables, arrays, etc.
- **Condiciones**.
  - Si se usan expresiones condicionales simples o compuestas con if…else, se pueden incluir dentro de la función las acciones a realizar.
- **Acceso a un campo de texto en un formulario.**
  - Para acceder al contenido de un campo *input* de tipo texto y cargar su contenido en una variable se utiliza la propiedad *value*.
  - Sintaxis:
    - let/var/const variable = document.getElementById("identificador id").value;
- **Vaciar uno o todos los campos de formulario.**
  - Si un dato introducido en un campo no es correcto y se avisa al usuario de tal circunstancia, se puede, además, eliminar el contenido escrito para volver a incluirlo de forma correcta.

- Para ello se utiliza la propiedad *value* de JavaScript que permite acceder al atributo *value* de los campos input y así eliminar el contenido cargando en ella el valor *null* o una cadena vacía.
- Para vaciar o resetear el formulario completo se usa el método reset() aplicado al formulario.
- Sintaxis:
    - document.getElementById("identificador id del campo").value = null;
    - document.getElementById("identificador id del campo").value = “”;
    - document.getElementById("identificador id del formulario").reset();

- Ejemplos:

```
function valEdad1()

        {
                var edad = document.getElementById("edad").value;
                var patron = /^\d\d$/;

                if(patron.test(edad)==false)
                {
                        alert("Edad incorrecta. Escríbela de nuevo.");
                        document.getElementById("edad").value = null;

                }
        }

function valEdad2()

        {
                var edad = document.getElementById("edad").value;
                var patron = /^\d\d$/;
                return patron.test(edad);

        }
function valEdad3()

        {
                var edad = parseInt(prompt(“Cúantos años tienes?;””));
                if(edad <=0 || edad > 120)
                {
                        alert("Edad incorrecta. Escríbela de nuevo.");
                        document.getElementById("edad”).value = null;

                }
        }
```

ACCESO A FORMULARIOS.

- **Array de formularios.**
    - Cuando se carga la página web en el navegador, el DOM crea un array con todos los formularios.
    - El array se llama forms.

- A éstos se puede acceder de diferentes formas a través del objeto *document.*

- **Acceso al array de formularios.**
  - **Objeto forms.**
    - En el array de formulario se cargan todos los formularios de la página, en el mismo orden en el que están incluidos en el código.
    - Para acceder a ellos, por tanto, se necesita conocer el índice de la posición donde un formulario concreto está almacenado.
    - Sintaxis:
      - document.forms[índice de posición en el array];
    - Ejemplos:
      - document.forms[0]; // Acceso al primer formulario.
      - document.forms[2]; // Acceso al tercer formulario.
  - **Atributo *name* y corchetes**.
    - Se puede acceder a un formulario incluyendo el nombre de su atributo *name* entre comillas dentro de los corchetes del array.
    - Sintaxis:
      - document.forms[“nombre en atributo *name*”];
    - Ejemplos:
      - <form name="for1" action="" method="post">
      - document.forms[“for1”]; // Acceso al formulario for1.
  - **Atributo *name* como propiedad**.
    - Se puede también acceder al formulario usando el valor del atributo *name* como propiedad, sin comillas y sin corchetes.
    - Sintaxis:
      - document.forms.nombre en atributo *name*;
    - Ejemplos:
      - <form name="for2" action="" method="post">
      - document.forms.for2; // Acceso al formulario for2.
  - **Atributo *name* como propiedad y sin objeto forms**.
    - También se puede incluir el valor del atributo name como una propiedad de document, de modo que, no es necesario usar el objeto form.
    - Sintaxis:
      - document.nombre en atributo *name*;
    - Ejemplos:
      - <form name="for3" action="" method="post">
      - document.for3; // Acceso al formulario for3.

ACCESO A LOS ELEMENTOS DE UN FORMULARIO.

- **Array de elementos.**
  - Dentro de cada formulario, el DOM crea un array con todos los elementos de éste.
  - El array se llama elements.
  - A éstos se puede acceder de diferentes formas a través del objeto *document.*
- **Acceso al array de elementos.**
  - **Objeto elements.**
    - En el array de elementos se cargan todos los elementos y campos del formulario, en el mismo orden en el que están incluidos en el código.

        - Para acceder a ellos, por tanto, se necesita conocer el índice de la posición donde un elemento concreto ha sido almacenado.
        - Sintaxis:
            - document.nombre del formulario.elements[índice de posición en el array];
        - Ejemplos:
            - document.for1.elements[0]; // Acceso al primer elemento del formulario for1.
            - document.for1.elements[2]; // Acceso al tercer elemento del formulario for1.
    - **Atributo *name* y corchetes**.
        - Se puede acceder a un elemento incluyendo el nombre de su atributo *name* entre comillas dentro de los corchetes del array.
        - Sintaxis:
            - document.nombre del formulario.elements[“nombre en atributo *name*”];
        - Ejemplos:
            - <label>Edad<input type="number" name="edad"></label>
            - document.for1.elements[“edad”]; // Acceso al campo edad del formulario for1.
    - **Atributo *name* como propiedad**.
        - Se puede también acceder al formulario usando el valor del atributo *name* como propiedad, sin comillas y sin corchetes.
        - Sintaxis:
            - document.nombre en atributo *name*;
        - Ejemplos:
            - <label>Nombre<input type = "text" name = "nombre"></label>
            - document.for2.nombre; // Acceso al campo nombre del formulario for2.
    - **Atributo *name* como propiedad y sin objeto forms**.
        - También se puede incluir el valor del atributo name como una propiedad de document, de modo que, no es necesario usar el objeto form.
        - Sintaxis:
            - document.nombre en atributo *name*;
        - Ejemplos:
            - <form name="for3" action="" method="post">

ACCESO A LAS PROPIEDADES DE LOS ELEMENTOS.

- Se puede acceder a las propiedades de un elemento para conocer su contenido o para modificarlo (si no son de sólo lectura).
- **Propiedades comunes a todos los elementos.**
    - **type.**
        - Indica de qué tipo de elementos se trata:
            - Elementos tipo input:
                - Muestra el valor del atributo *type* que tengan éstos, es decir, *text, password, radio*, etc.

- Listas desplegables select:
    - Muestra los valores *select-one*, para las listas de selección única o, *select-multiple,* para las de selección múltiple.
- Áreas de texto:
    - Muestra el valor *textarea*.

- Dichos valores pueden guardarse en una variable.
- Sintaxis:
    - document.nombreformulario.nombre elemento.type;
    - document.getElementById("Id del elemento").type;
- Ejemplo:
    - <label>Contraseña<input type = "password" name = "pass"></label>
    - let tipo = document.for1.pass.type; // Acceso al campo *contraseña* del formulario *for1* para guardar en la variable *tipo* el valor *password*.
    - let tipo = document.getElementById("nom").form; // Acceso con método *get* al campo *contraseña* del formulario *for1* para guardar en la variable *tipo* el valor *password*.

- **form.**
    - Permite conocer a qué formulario pertenece el elemento en cuestión.
    - Esta información puede guardarse en una variable.
    - Si no funciona, usar getAttribute("form").
    - Sintaxis:
        - document.nombreformulario.nombre elemento.form;
        - document.getElementById("Id del elemento").form;
    - Ejemplo:
        - <form name="for1" action="" method="post" id ="f1">
        - <label>Nombre<input type = "text" name = "nombre" form ="f1" id="nom"></label>
        - let formulario = document.for1.nombre.form; // Acceso al campo *nombre* del formulario *for1* para guardar en la variable *formulario* su nombre identificador.
        - let formulario = document.getElementById("nom").form; // Acceso con método *get* al campo *nombre* del formulario *for1* para guardar en la variable *formulario* su nombre identificador.
- **name.**
    - Permite obtener el valor del atributo *name* de un elemento.
    - Esta propiedad es de sólo lectura, por lo que no se puede modificar su contenido.
    - Su valor puede guardarse en una variable.
    - Sintaxis:
        - document.nombreformulario.nombre elemento.name;
        - document.getElementById("Id del elemento").name;
    - Ejemplo:
        - <label>Nombre<input type = "text" name = "nombre" form ="f1" id="n"></label>

- let valor = document.for1.nombre.name; // Acceso al campo *nombre* del formulario *for1* para guardar, en la variable *valor,* el valor de su atributo *name*.
- let valor = document. getElementById("n").name; // Acceso con método get al campo *nombre* del formulario *for1* para guardar, en la variable *valor,* el valor de su atributo *name*.

- **value.**
  - Permite acceder a la propiedad value de un elemento para cargar su valor en una variable o modificarlo.
  - Según el elemento, se obtiene o cargan distintos contenidos:
    - Campos *input* de tipo *text y password,* así como, áreas de texto *textarea*.
      - Permite obtener el texto escrito por el usuario o escribir texto en ellos.
    - Botones de acción o comando.
      - Muestran el texto que tiene el botón o permiten cambiarlo por otro.
    - Botones de opción(*radio*) y casillas de verificación (*checkbox*):
      - Muestran el valor del atributo *value,* que permite conocer que valor tiene la opción seleccionada.
      - Este valor también puede ser cambiado.
  - Sintaxis:
    - **Carga de valor**:
      - variable = document.nombreformulario.nombre elemento.value;
      - variable = document.getElementById("identificador").value; // Mediante método get.
    - **Cambio de valor**:
      - document.nombreformulario.nombre elemento.value = nuevo valor;
      - document.getElementById("identificador").value = nuevo valor;
  - Ejemplos:
    - **Carga de valor:**
      - <label>Dirección<input type = "text" name = "dir" id="d"></label>
      - let direccion = document.for1.dir.value; // Acceso al campo dirección del formulario *for1* para guardar, en la variable direccion*,* lo que el usuario haya escrito en este campo.
      - let direccion = document.getElementById("d").value; // Acceso al campo dirección del formulario *for1* mediante un método get para guardar, en la variable direccion*,* lo que el usuario haya escrito en este campo.
    - **Cambio de valor**:
      - <label>Comentarios</label><textarea name = "coment" id="com"></textarea>

        - document.for1bre elemento.coment.value = “Nuevo comentario”; //Acceso al área de texto del formulario *for1* para escribir en ella un nuevo comentario.
        - document.getElementById(“com”).value = “Nuevo comentario”; //Acceso al área de texto del formulario *for1* usando un método get, para escribir en ella un nuevo comentario.

- **checked.**

  Botones de opción (radio).

  - En los botones de opción, *checked* permite conocer la única opción que ha sido seleccionada.
  - Devuelve un valor booleano, true o false, por lo que se pueden crear condiciones para comprobar que opción ha sido pulsada.
  - Al tener todas las opciones el mismo valor en la propiedad *name*, para que funcionen sincronizadas, cuando se accede a ellas se crea un array de opciones que puede recorrerse usando un bucle for, para así, con *checked* preguntar que opción es la elegida.
  - Sintaxis:
    - **Creación del array de opciones**:
      - nombre del array de opciones = document.nombreformulario.nombre elemento en name;
      - nombre del array de opciones = document.método get para varias opciones(“argumento”).name; // Mediante método get.
    - **Acceso a los elementos del array con checked**.
      - Array [índice de posición].checked;
  - Ejemplo:

```
let opciones= document.for.botones;
let seleccionada;
for (var i=0; i < opciones.length; i++)
 {
        seleccionada = botones[i].checked;
         if (seleccionada == true)
        {
                alert("Opción elegida" + botones[i].value);
        }
   }
```

  Casillas de verificación (checkbox).

  - En las casillas de verificación, *checked* permite conocer que opciones han sido seleccionadas, que pueden ser ninguna, una, varias o todas.
  - Devuelve un valor booleano, true o false, por lo que se pueden crear condiciones para comprobar que opción/es han sido pulsadas.
  - A las distintas opciones se puede acceder de forma individual por sus valores en los atributos *id o name* usando los métodos *get* correspondientes y así, cargar en una variable cada opción seleccionada.

- También se puede crear un array con todas las opciones disponibles y, posteriormente, con un bucle *for* recorrer cada una de ellas para con *checked,* ir preguntado si están o no seleccionadas.
- El array puede crearse con el método querySelectorAll(), usando el atributo *type* y el valor *checkbox* para cargar todas las opciones.
- Sintaxis:
  - **Creación de una variable para cargar una sola opción**:
    - variable = document.nombreformulario.nombre elemento en name; // Carga en una variable de la casilla u opción en una variable usando *name*. También se puede usar un método get.
    - variable.checked; // Devuelve verdadero o falso si la opción está o no seleccionada.
    - if(variable.checked==true){instrucciones;} // Para crear condiciones usando checked.
  - **Creación de un array de opciones**:
    - nombre del array de opciones = document.querysSelectorAll([“type=’checkbox’]”)
  - **Acceso a los elementos del array con checked**.
    - Array [índice de posición].checked;
- Ejemplo:

```
let casillas= document.querysSelectorAll ([“type=’checkbox’]”);
let opciones=””;
for (var i=0; i < casillas.length; i++)
 {
        if (casillas[i].checked == true)
        {
                opciones = opciones + “ “+ casillas[i].value; // Los distintos valores de las casillas de verificación seleccionadas se van concatenando en la variable acumuladora opciones.
        }
  }
```

ACCESO A LISTAS DESPLEGABLES. (SELECT)

- **Array de opciones.**
  - Dentro de cada lista desplegable creada con *select*, el DOM crea un array con todas sus opciones.
  - El array se llama options.
  - A éstas se puede acceder de diferentes formas a través del objeto *document.*
- **Acceso al array de opciones.**
  - **Objeto options.**
    - En el array de opciones se cargan todas las opciones presentes en la lista, en el mismo orden en el que están incluidas en el código.
    - Para acceder a ellas, por tanto, se necesita conocer el índice de la posición donde una opción concreta ha sido almacenada.

        - Sintaxis:
            - Opción 1:
                - Variable = document.nombre del formulario.nombre de la lista *select*.options;
            - Opción 2:
                - variable = document.nombre del formulario.nombre de la lista *select*;
                - otra_variable = variable.options;
        - Ejemplos:
            - let menu = document.for1.lista.options; // Acceso y creación el array menú cargando todos los valores de la lista cuyo nombre en el atributo *name* es lista.
            - let menu = document.for1.lista; // Acceso a la lista que, en el formulario for1, tiene como valor en su atributo *name* el nombre de lista.
            - let lista = menu.options; // Creación del array lista con todas las opciones del elemento select cargadas en *menu*.
- **Propiedades para el array de opciones.**
    - **length.**
        - Permite conocer cuántas opciones u elementos se han cargado en el array creado a partir de *select*.
        - Sintaxis:
            - nombre del array de opciones.length;
            - document.nombre del formulario.nombre de la lista select.length; // Opción sin crear un array.
        - Ejemplos:
            - let elementos = menu.length // Carga, en la variable elementos, del número de opciones o elementos incluidos en el array menu.
            - let numElementos = document.for2.opciones.length; // Carga, en la variable numElementos, del número de opciones o elementos incluidos en una lista cuyo valor en el atributo *name* es *opciones*.
    - **selectedIndex**.
        - Permite conocer el índice de la posición que ocupa, en el array, el elemento seleccionado.
        - Sintaxis:
            - nombre del array de opciones.selectedIndex;
            - document.nombre del formulario.nombre de la lista select.selectedIndex; // Opción sin crear un array.
        - Ejemplos:
            - let index = menu. selectedIndex // Carga en la variable *index*, el índice de posición de la opción seleccionada en el array menu.
            - let indice = document.for2.opciones.selectedIndex; // Carga, en la variable *indice* el número de posición de la opción u elemento seleccionado en una lista cuyo valor en el atributo *name* es *opciones*.
    - **value**.

- Carga en una variable el valor del atributo *value* del elemento del array de opciones que se indique a través de su índice.
- Sintaxis:
  - variable = nombre del array de opciones[indice].value;
  - variable = document.nombre del formulario. Nombre de la lista.options[indice].value; // Equivalente a la anterior usando el array *options* sin crear un array nuevo.
- Ejemplos:
  - let valor = menu[1].value // Carga en la variable *valor*, el valor del atributo *value* del segundo elemento del array de opciones *menu*.
  - let valor = document.for3.opciones.options[1].value // Carga en la variable *valor*, el valor del atributo *value* del segundo elemento del array *options* incluido en la lista *opciones* del formulario *for3*.

- **text.**
  - Carga en una variable el texto que representa cada valor del elemento del array de opciones que se indique a través de su índice, es decir, permite cargar el nombre de la opción seleccionada que está incluido entre las etiquetas <option></option>.
  - Sintaxis:
    - variable = nombre del array de opciones[indice].text;
    - variable = document.nombre del formulario. Nombre de la lista.options[indice].text; // Equivalente a la anterior usando el array *options* sin crear un array nuevo.
  - Ejemplos:
    - let texto = menu[0].text // Carga en la variable *texto*, el texto incluido en la etiqueta <option> del primer elemento del array de opciones *menu*.
    - let texto = document.for1.opciones.options[0].text // Carga en la variable *texto*, el texto incluido en la etiqueta <option> del primer elemento del array *options* incluido en la lista *opciones* del formulario *for1*.
- **selected.**
  - Permite conocer que opción u opciones de una lista está seleccionada.
  - Devuelve un valor booleano, true o false, por lo que se pueden crear condiciones para comprobar que opción ha sido pulsada.
  - El array de opciones puede recorrerse usando un bucle for, para así, con *selected* preguntar que opción es la elegida.
  - En caso de selecciones múltiples, se puede crear otro array con los valores o texto de las opciones seleccionadas.
  - Sintaxis:
    -
      - nombre del array de opciones[índice de la opción].selected; // Devuelve true o false.
      - if(nombre del array de opciones[índice de la opción].selected==true){instrucciones;} // Para crear condiciones usando *selected*.
  - Ejemplo:

```
let lista = document.for1.opciones;
let menu=lista.options;
let selecciones;
for (let i = 0; i < menu.length; i++)
{
        if (menu[i].selected == true )
        {
                selecciones = menu[i].text; // Se cargan en la variable selecciones, y en cada vuelta del bucle, los textos de las opciones seleccionadas de la lista opciones del formulario for1, cuya totalidad de elementos previamente se incluyeron en el array menu.
                También se pueden crear variables acumuladoras o arrays para cargar todas opciones seleccionadas.
        }
}
```

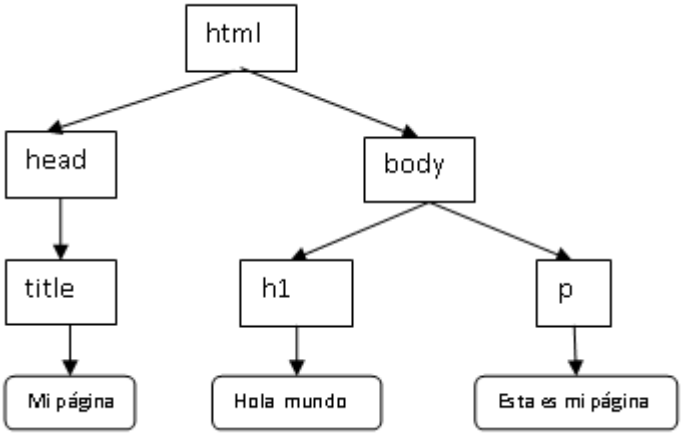
BOM.

- **Concepto.**
  - Modelo de Objetos del Navegador - Browser Object Model.
  - Permite a JavaScript comunicarse con el navegador y acceder a todas sus áreas.
  - JavaScript incluye un conjunto de objetos y funcionalidades que permiten acceder y modificar las propiedades y elementos de la ventana del navegador.
  - El modelo BOM es específico de cada navegador.
  - Objetos del BOM:
    - *window, navigator, screen, history, location* y *frames*.

DOM.

- **Concepto:**
  - Modelo de objetos del documento - Document Object Model.
  - Documento que contiene toda la estructura de un documento HTML.
  - Es una interfaz de programación que permite crear, cambiar, modificar o eliminar elementos del documento web.
  - Describe el contenido del documento como un conjunto de objetos, sobre los que JavaScript puede interactuar.
  - También se pueden añadir eventos a dichos elementos para crear páginas web más dinámicas e interactivas.
  - El DOM es independiente del navegador y se corresponde con el estándar definido por la W3C.
  - Objetos del DOM:
    - *document* y sus objetos hijos como *anchors, forms, images, links, layers, element* y otros.
- **Árbol DOM:**

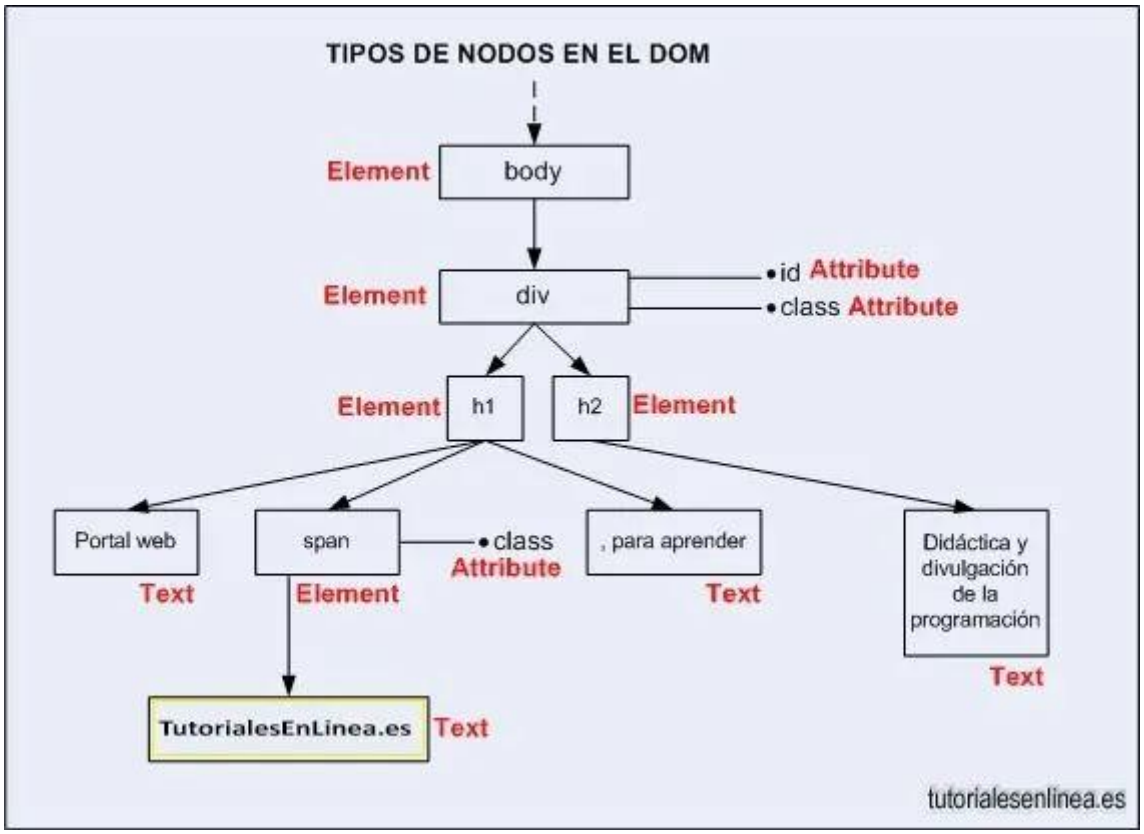
- Es una estructura jerárquica y ramificada que muestra los distintos nodos y relaciones que forman una página web.



- Página Web correspondiente al anterior árbol DOM.

```
<html>
        <head>
            <title>
                        mi página
            </title>
        </head>
        <body>
            <h1 id = “titulo1”>
                        Hola mundo
            </h1>
            <p>
                        Esta es mi página
            </p>
        </body>
</html>
```

- **Otro ejemplo de árbol DOM.**



- **Nodo:**
  - Cada uno de los componentes de una página web.
  - Hay 12 diferentes.
  - **Tipos:**
    - **Raíz o documento (Document).**
      - Etiqueta <HTML>.
      - Nodo raíz, a partir del cual derivan el resto de nodos.
    - **Etiqueta (Element).**
      - Son el resto de etiquetas HTML, es decir, los nodos definidos por etiquetas HTML.
      - Dentro de un nodo Element puede haber nodos hijos, por ejemplo, etiquetas dentro de .
    - **Texto (Text).**
      - Contenido textual entre etiquetas, por ejemplo, Hola .
      - Son nodos hijos de tipo texto dentro de un nodo Element.
      - Los navegadores pueden crear nodos de tipo texto sin contenido para representar saltos de línea o espacios vacíos.
    - **Atributo (Attr o Attribute).**
      - Se corresponden con los atributos de las etiquetas HTML.
      - Aunque son nodos, se consideran más bien información asociada a nodos de tipo Element.
    - **Comentarios (Comment).**
      - Son los comentarios incluidos en la página. <!- -texto del comentario - - >

        - **Otros tipos.**
            - Declaraciones doctype en la cabecera de los documentos HTML generan nodos, como también, CDatasection, DocumentType, DocumentFragment, Entity, EntityReference, ProcessingInstrution, Notation.

ACCESO A DOM. ACCESO A PÁGINAS WEB DESDE JAVASCRIPT.

- Hay varios objetos prototípicos o clase en JavaScript que permiten acceder al DOM como window, document, frame, location, history, screen, node, element, etc, …

OBJETO WINDOW.

- Objeto relacionado con la ventana del navegador y sus características.
- Es un objeto de primer nivel o nivel superior del cual derivan otros.
- **Objetos contenidos.**
    - document, frame, history, navigator, location, screen, …
- **PROPIEDADES**.
- **MÉTODOS**.
    - **alert().**
        - Muestra un cuadro modal de alerta o aviso.
        - Incluye sólo un botón para aceptar las indicaciones del cuadro.
        - Sintaxis:
            - window.alert(“mensaje”);
            - alert(“mensaje”);
        - Ejemplos:
            - alert(“Hola”);
            - alert(“La variable no tiene un dato numérico”);
    - **confirm().**
        - Muestra un cuadro modal de confirmación.
        - Incluye un botón para aceptar o confirmar una acción, y otro para cancelarla.
        - Devuelve true si se pulsa en aceptar.
        - Devuelve false si se pulsa en cancelar.
        - Con una instrucción if…else, se puede realizar una u otra acción según el botón que se pulse.
        - Sintaxis:
            - window.confirm(“mensaje”);
            - confirm(“mensaje”);
        - Ejemplo:
            - confirm(“¿Es correcto el nombre?”);

OBJETO DOCUMENT.

- Permite acceder a todos los elementos (textos, imágenes, enlaces, formularios, etc.), de una página web para modificarlos o añadirlos nuevos.
- Incluye otros objetos, métodos y propiedades.
- **OBJETOS.**
    - La mayoría sirve para obtener información sobre el contenido de un documento web, con fines estadísticos, creación de informes.
- Sintaxis.

    - document.objeto;
    - document.objeto.propiedades;
- Objetos del objeto prototípico document.
    - **styleSheets.**
        - Devuelve un array con todas las hojas de estilo CSS que se están usando en la página.
    - **anchors**.
        - Devuelve un array con todas las anclas del documento.
    - **links.**
        - Devuelve un array con todos los enlaces del documento.
            - nombre array de enlaces[indice];
                - var enlaces = document.links;
                - enlaces[0];
                - document.links[0];
        - Se pueden mostrar los atributos de los enlaces con:
            - nombre array de enlaces[indice].atributo;
                - enlaces[0].href;
                - enlaces[2].target;
            - document.links[indice].atributo;
                - document.links[0].href;
                - document.links[2].target;
    - **images**.
        - Devuelve un array con todas las imágenes del documento.
            - var nombre del array = document.images;
                - var fotos = document.images;
        - Se pueden mostrar los atributos de las imágenes con:
            - nombre array de imágenes[indice].atributo;
                - fotos[0].src;
                - fotos[2].height;
                - fotos[5].alt;
            - document.links[indice].atributo;
                - document.images[0].src;
                - document.images[2].width;
                - document.images[5].alt;
    - **scripts.**
        - Devuelve un array con todos los scripts que hay en el documento.
    - **forms.**
        - Devuelve un array con todas las referencias a formularios que hay en el documento.
    - **URL**.
        - Devuelve un variable con la dirección del documento actual.
- Acceso a cada elemento individual por posición en el array o con el atributo name**.**
- Sintaxis**:**
    - document**.objeto[“atributo name”];**
    - document**.objeto[número de posición en array];**
- Ejemplos:
    - var miFormulario = document.forms[“formularioRegistro”]);

- document.write(document.forms[“formularioRegistro”]);
- var miImagen = document.images[“logo”];
- var miEnlace = document.links[3];

- **PROPIEDADES**.
- **MÉTODOS.**
  - Sirven para acceder directamente a los nodos DOM.
  - Empleados para manejar el DOM.
  - Sintaxis genérica:
    - document.método(parámetros);
- **métodos:**
  - **write().**
    - Muestra texto en el documento web justo en el sitio donde se incluya el script.
    - Muestra texto HTML en el documento.
    - Se pueden incluir dentro del método como parámetros etiquetas HTML y reglas CSS:
      - Uso de las etiquetas HTML encerradas entre <> y con los atributos correspondientes.
      - Uso del atributo global style de HTML para especificar los estilos CSS:
        - style = “propiedad1 css: valor; propiedad2 css: valor; …”
    -
    - Sintaxis:
      - document.write(“texto” o variables o arrays u otros métodos o expresiones aritmeticas o booleanas o HTML o CSS.);
    - Ejemplos:
      - var saludo = “Hola”;
      - document.write("Hola ");
      - document.write(saludo);
      - document.write("El valor de la posición 3 del array es: “ + numeros[3]);
      - document.write("La suma es: “ + (7 + 20));
      - document.write("La suma es: “ + (n1 + n2));
    - Ejemplos con HTML:
      - document.write("<h1>Hola</h1> ");
      - document.write(" <h1>Hola</h1> Párrafo1 Párrafo2 ");
      - document.write('Wikipedia');
    - Ejemplos con CSS:
      - document.write('<h1 style = "color:blue; font-family: comic sans ms">Hola</h1>');
- **getElementById().**
  - Devuelve el elemento del documento con atributo id que coincide con el parámetro.
  - No se incluye #.
  - El contenido o el nodo devuelto se puede cargar en una variable.
  - Sintaxis:
    - var nombre de la variable para el id = document.getElementById(“identificador”);
  - Ejemplo:
    - var division = document.getElementById(“d1”).innerHTML;
- **getElementsByName().**

- Devuelve el/los element/os del documento cuyo atributo name coincida con el parámetro.
- El contenido o el nodo devuelto se puede cargar en un array.
- Sintaxis:
  - var nombre del array para name = document.getElementsByName("nombre");
- Ejemplo:
  - var parrafos = document.getElementsByName("html");
-

- **getElementsByTagName().**
  - Devuelve el/los element/os del documento cuya etiqueta HTML coincida con el parámetro.
  - El contenido o el nodo devuelto se puede cargar en un array.
  - Sintaxis:
    - var nombre del array para name = document.getElementsByTagName("etiqueta");
  - Ejemplo:
    - var parrafos = document.getElementsByTagName("p");
  -
- **getElementsByClassName().**
  - Devuelve el/los element/os del documento cuyo nombre de clase coincide con el parámetro.
  - El contenido o el nodo devuelto se puede cargar en un array.
  - Sintaxis:
    - var nombre del array para name = document.getElementsByClassName("nombre de clase");
  - Ejemplo:
    - var parrafos = document.getElementsByClassName("parrafos");
- **querySelector().**
  - Devuelve el primer elemento que coincide con el selector CSS incluido como parámetro.
  - No tiene por qué haber reglas CSS en la página o en un archivo externo para poder usar el método querySelector().
  - El contenido o el nodo devuelto se puede cargar en una variable.
  - Se incluye puntos (.), #, ~, …
  - Sintaxis:
    - var nombre de la variable = document.querySelector("selector");
  - Ejemplo:
    - var parrafos = document.querySelector ("p");
    - var parrafos = document. querySelector ("#div1 p");
    - var parrafos = document. querySelector ("[class $= "ma"]);
    -
- **querySelectorAll().**
  - Devuelve todos los elementos que coinciden con el selector CSS incluido como parámetro.
  - No tiene por qué haber reglas CSS en la página o en un archivo externo para poder usar el método querySelectorAll().
  - El contenido o el nodo devuelto se puede cargar en una variable.
  - Se puede incluir más de un selector como argumentos del método.
  - Se incluye puntos (.), #, ~, …
  - Sintaxis:
    - var nombre de la variable = document.querySelectorAll("selector");
  - Ejemplo:
    - var parrafos = document. querySelectorAll ("p");

        - var parrafos = document. querySelectorAll (“#div1 p”);
        - var parrafos = document. querySelectorAll (“[class $= “ma”]);
        - var cosas = document. querySelectorAll (“#div1 p, a, .fotos”);
    - **Nota**:
        - Los métodos *get* cargan los nodos completos. Para utilizar algún contenido concreto de nodo hay que usar la propiedad correspondiente y sus resultados cargarlos en variables, arrays, etc.
        - Por ejemplo, para poder usar los métodos del objeto Array hay que crear uno adicional para cargar en él, el contenido textual que se haya extraído con las propiedades innerHTML, textContent o innerText.

- **createElement().**
    - Método de document.
    - Crea un nuevo elemento del tipo especificado como parámetro.
    - Tras crear un elemento hay que insertarlo en el elemento padre que lo contendrá, así como, incluirle texto o atributos si son necesarios, por lo que serán necesarios usar también los métodos appendChild(), createTextNode() o la propiedad innerHTML y createAttribute(),
    - Sintaxis:
        - document.createElement("Elemento a crear");
    - Ejemplos:

```
let lista = document.getElementById("lista1");
let elementoNuevo= document.createElement("li"); //Equivale a <li>...</li>
elementoNuevo.innerHTML = "Texto opción"; // Equivale a <li>Texto opción</li>
lista.appendChild(elementoNuevo);
```

- **createTextNode().**
    - Método de document.
    - Crea un nodo de texto para, posteriormente, añadirlo o asociarlo a un elemento que pueda incluir texto.
    - Sintaxis:
        - document.createTextNode("Texto");
    - Ejemplos:

```
let lista = document.getElementById("lista1");
let elementoNuevo = document.createElement("li");
let elementoTexto = document.createTextNode("Texto opción");
elementoNuevo.appendChild(elementoTexto);
lista.appendChild(elementoNuevo);
```

OBJETO ELEMENT.

- Objeto que representa cualquier elemento contenido en el DOM de una página web (párrafos, tablas, listas, títulos, enlaces, etc.).

OBJETOS Y PROPIEDADES.

- **children.**
    - Propiedad que muestra o devuelve todos los elementos anidados (hijos) de un elemento determinado.

- Si hay varios elementos se cargan en un array.
- Si un elemento no tiene elementos hijos, la propiedad length devolverá 0.
- Se puede usar un bucle *for* para mostrar todos los tipos de objetos que son los hijos de un elemento dado.
- Sintaxis:

  document.getElementById("valor atributo id").children;
- Ejemplo:

```
let elementosHijos = document.getElementById("division1").children;
document.write(elementosHijos.length); // Muestra cuántos hijos hay.
document.write(elementosHijos[2]); // Muestra el tipo de objeto que es el tercer hijo.
```

MÉTODOS.

- **appendChild().**
  - Permite insertar un nodo o elemento al final de todos los hijos de un mismo nivel de anidamiento, es decir, el nuevo nodo se incluye inmediatamente después de los hijos ya existentes, si hay alguno.
  - Sintaxis:
    - Nodo padre.appendChild(Nuevo nodo);
      - **Nodo padre**:
        - Padre o nodo de nivel superior dentro del cual se insertará el nuevo nodo.
      - **Nuevo nodo**:
        - Nodo o elemento nuevo a insertar.
  - Ejemplos:
    - Añadir elemento a una lista ya existente:

```
let lista = document.getElemtnById(“lista1”);
let elemento = document.createElement(“li”);
elemento.innerHTML = ”Nuevo elemento al final de una lista existente”;
lista.appendChild(elemento);
```

    - Añadir elemento a una lista nueva:

```
let lista = document.createElement(“ul”);
let elemento = document.createElement(“li”);
elemento.innerHTML = ”Nuevo elemento al final de una lista nueva”;
lista.appendChild(elemento);
```

    - Añadir un párrafo al final de una división:

```
division1.appendChild(ultimoParrafo);
```

- **insertBefore().**
  - Permite insertar un nodo o elemento antes que otro elemento que esté en el mismo nivel de anidamiento.
  - Para ello, necesitamos conocer u obtener la referencia del nodo antes del cual se insertará el nuevo y la del elemento padre dentro del cual se producirá la inserción.
  - Sintaxis:
    - Nodo padre.insertBefore(Nuevo nodo, Nodo de referencia);
      - **Nodo padre**:
        - Padre o nodo de nivel superior dentro del cual se insertará el nuevo nodo.
      - **Nuevo nodo**:
        - Nodo o elemento nuevo a insertar.

        - **Nodo de referencia**:
            - Nodo antes del cual se insertará el nuevo.
            - Si el nodo de referencia es *null* o no se especifica, el nuevo se añadirá al final de la lista de hijos del nodo padre especificado, como con appendChild().
    - Ejemplo:
        - Añadir un elemento antes del quinto elemento de una división denominada *division1*:

```
division1.insertBefore(elemento4, elemento5);
```

- **replaceChild().**
    - Permite reemplazar un nodo por otro.
    - Sintaxis:
        - Nodo padre.replaceChild(Nuevo nodo, Nodo a reemplazar);
            - **Nodo padre**:
                - Padre o nodo de nivel superior dentro del cual se insertará el nuevo nodo.
            - **Nuevo nodo**:
                - Nodo o elemento nuevo a insertar.
            - **Nodo a reemplazar**:
                - Nodo que será reemplazado o sustituido.
    - Ejemplo:
        - Sustituir un párrafo por otro nuevo en una división denominada *division1*:

```
division1.insertBefore(parrafoNuevo, parrafo2);
```

- **removeChild().**
    - Permite eliminar un nodo.
    - Sintaxis:
        - Nodo padre.removeChild(Nodo a eliminar);
            - **Nodo padre**:
                - Padre o nodo de nivel superior dentro del cual está el nodo a eliminar.
            - **Nuevo a eliminar**:
                - Nodo o elemento hijo a eliminar.
    - Ejemplos:
        - Eliminar un elemento de una división:

```
let division1 = document.getElementById("d1");
let parrafo4 = document.getElementById("p4");
division1.removeChild(parrafo4);
```

        - Eliminar un elemento de una lista:

```
let opcion2 = document.getElementById("op2");
document.getElementById("lista1").removeChild(opcion2);
```

NODOS DE TIPO ATRIBUTO.

- Los atributos son modificadores que controlan o proporcionan funcionalidad adicional a las etiquetas o elementos HTML.
- Son un nodo más del DOM.

PROPIEDADES PARA AÑADIR ATRIBUTOS.

- Se pueden usar algunos atributos HTML como propiedades JavaScript para así, o asignárselos a elementos que no los tengan, o cambiar los valores de los existentes.
- No todos los atributos se pueden usar como propiedad.
- Los que no se pueden usar, se asignan con el método setAttribute().
- Algunos de los que si se pueden usar son: id, className, lang, accessKey, tabIndex o style, entre otros.
- Sintaxis.
    - elemento.propiedad =”valor”;
- Ejemplos:

```
let texto = document.getElementById(“parrafo1”);
texto.id=”p1”;
texto.className=”párrafos”;
texto.tabIndex =”2”;
```

MÉTODOS PARA CREAR, ACCEDER O ESTABLECER NODOS DE TIPO ATRIBUTO.

- **getAttribute().**
    - Método del objeto Element.
    - Devuelve el valor del atributo especificado como parámetro en el método.
    - Este se puede cargar en una variable o array dependiendo del método de obtención de nodos empleado.
    - Sintaxis:
        - var variable/array = document.método(parámetros).getAttribute(“atributo”);
        - var variable1/array1 = document.método(parámetros);
        - var variable2/array2 = variable1. getAttribute(“atributo”);
    - Ejemplos:
        - var parrafo = document.getElementById("p4"). getAttribute(“class”);
        - var alternativo = document.getElementById("imagen1"). getAttribute(“alt”);
        - var a = document.getElementById("p4");
        - var alineacion = a. getAttribute(“align”);
- **getAttributeNames().**
    - Carga todos los atributos de un elemento en un array.
    - Sintaxis:
        - document.método(parámetros).getAttributeNames();
        - var variable1 = document.método(parámetros);
        - variable1. getAttributeNames();
    - Ejemplo:
        - Párrafo 1
        - let atributos = document.getElementById("p1").getAttributeNames(); // Carga, en el array atributos, los atributos id, class y name.
- **setAttribute().**
    - Método del objeto Element.
    - Permite añadir o modificar los atributos de un elemento.
    - Establece el valor del atributo especificado como parámetro en el método.
    - Sintaxis:

- document.método(parámetros).setAttribute(“nombre del atributo”,”valor”);
- var variable1 = document.método(parámetros);
- variable1. setAttribute(“nombre del atributo”,”valor”);

▪ Ejemplos:
- var parrafo = document.getElementById("p4");
- parrafo.setAttribute(“id”,”p1”);
- var division = document.getElementById("div1");
- division.setAttribute(“name”,”d1”);

o **createAttribute().**
▪ Método de document.
▪ Crea un nuevo nodo de tipo atributo para posteriormente asociarlo a un elemento determinado.
▪ Para asignar el valor para el atributo se utiliza la propiedad value.
o Sintaxis:
- **Creación del nodo**:
▪ document.createAttribute(“Atributo”);
- **Atributo:**
o Tipo de atributo a crear (id, class, alt, href, type, etc.).
- **Asignación de valor**:
▪ variable.value =”valor”;
- **Variable**:
o Variable que contiene el nodo creado.
- **Valor:**
o Contenido asociado al atributo.
- **Mostrar el valor del atributo:**
▪ elemento.getAttribute(“Atributo”);
- **Elemento:**
o Nodo que contiene el atributo cuyo valor se quiere mostrar.
- **Atributo:**
o Tipo de atributo cuyo valor se quiere mostrar (id, class, alt, href, type, etc.).

o Ejemplo:
▪ let division = document.getElementById("div1");
▪ let atributo = document.createAttribute("class");
▪ atributo.value = "contenedores";
▪ division.setAttributeNode(atributo);
▪ console.log(division.getAttribute("atributo")); // Mostrará *contenedores*.

o **setAttributeNode().**
▪ Método de document.
▪ Añade un nuevo nodo de tipo atributo al elemento que se especifique.
▪ Sintaxis:
- **elemento. setAttributeNode(atributo).**
▪ **Atributo:**

- Variable que contiene el nodo de tipo atributo incluyendo su valor.

- Ejemplo: (En el ejemplo anterior la línea azul).

```
let division = document.getElementById("div1");
let atributo = document.createAttribute("class");
atributo.value = "contenedor";
division.setAttributeNode(atributo);
console.log(division.getAttribute("atributo")); // Mostrará contenedor.
```

PROPIEDADES PARA ACCEDER AL CONTENIDO TEXTUAL DE LOS NODOS.

- **innerHTML.**
  - Propiedad del objeto Element.
  - Devuelve o establece un texto de un elemento o nodo en formato html.
  - Muestra saltos de tabulación, línea o espacios en blanco.
  - Muestra etiquetas HTML con el método alert().
  - Uso:
    - Recuperar o escribir texto en un elemento.
    - Permite insertar etiquetas HTML.
  - **Devolver contenido.**
    - Sintaxis:
      - var variable = document.método(parámetros).innerHTML;
      - var variable1 = document.método(parámetros);
      - var variable2 = variable1.innerHTML;
    - Ejemplo:
      - var parrafo = document.getElementById("p4").innerHTML;
      - document.write("Contenido del cuarto párrafo : " + parrafo + "
");
  - **Establecer contenido**.
    - **Añadir contenido sustituyendo**.
      - Sintaxis:

        ```
        document.metodo(parámetros).innerHTML = "Contenido HTML a insertar";
        ```

      - Ejemplo:

        ```
        document.getElementById("p4").innerHTML = "<p>Párrafo nuevo sustituto</p>";
        document.body.innerHTML = ""; // Vacía de contenido la página web.
        ```

    - **Añadir contenido manteniendo el existente.**
      - El contenido se añade al final del contenido ya existente.
      - Sintaxis:

        ```
        document.metodo(parámetros).innerHTML += "Contenido HTML a insertar";
        ```

      - Ejemplo:

        ```
        document.getElementById("div1").innerHTML += "<p><strong>Párrafo nuevo </strong ></p>";
        ```

document.body.innerHTML + = “<h1>Hola</h1> Otro párrafo ”;
document.getElementById("p1").innerHTML += “ Párrafo añadido”;
document.getElementById("p1").innerHTML += “ Párrafo añadido ”;

- **innerText.**
  - Propiedad del objeto HTMLElement.
  - Devuelve o establece un texto de un elemento o nodo sin formato (texto plano).
  - No muestra saltos de tabulación, línea o espacios en blanco.
  - No muestra etiquetas HTML.
  - Uso:
    - Recuperar o escribir texto en un elemento.
  - Devolver contenido.
    - Sintaxis:
      - var variable = document.método(parámetros).innerText;
      - var variable1 = document.método(parámetros);
      - var variable2 = variable1.innerText;
    - Ejemplo:
      - var parrafo = document.getElementById("p4").innerText;
      - document.write("Contenido del cuarto párrafo : " + parrafo + "
");
  - **Establecer contenido**.
    - No reconoce etiquetas HTML ni formato de la página.
- **textContent.**
  - Propiedad del objeto Node.
  - Devuelve o establece un texto de un elemento o nodo sin formato (texto plano).
  - Si muestra saltos de tabulación, línea o espacios en blanco con alert().
  - No muestra etiquetas HTML.
  - Lee las etiquetas <script> y <style>.
  - Muestra el texto oculto CSS.
  - Uso:
    - Recuperar o escribir texto en un elemento.
  - Devolver contenido.
    - Sintaxis:
      - var variable = document.método(parámetros).textContent;
      - var variable1 = document.método(parámetros);
      - var variable2 = variable1.textContent;
    - Ejemplo:
      - var parrafo = document.getElementById("p4").textContent;
      - document.write("Contenido del cuarto párrafo : " + parrafo + "
");
  - **Establecer contenido**.
    - No reconoce etiquetas HTML, ni formato de la página.

APLICAR ESTILOS CSS DESDE JAVASCRIPT.

STYLE.

- Propiedad u objeto de *Element*.
- Se usa para aplicar un determinado estilo o formato a cualquier elemento HTML.
- Si las propiedades CSS son compuestas, se elimina el guion medio y se pone la primera letra de la segunda palabra en mayúsculas:

- Ejemplos:
  - font-family -> fontFamily
  - font-size -> fontSize
  - background-color -> backgroundColor

- Sintaxis:
  - Opción 1:
    - document.método(parámetros).style.propiedadCSS = “valor”;
  - Opción 2:
    - var variable1 = document.método(parámetros);
    - variable1. style.propiedadCSS = “valor”;

- Ejemplos:

  - Opción 1:

```
<p id = “p1”>Hola</p>
document.getElementById(“p1”).style.color = “#f00”;
document.getElementById(“p1”).style.fontSize = “20px”;
```

  - Opción 2:

```
<p id = “p1”>Hola</p>
var parrafo = document.getElementById(“p1”);
parrafo. style.color = “#f00”;
parrafo.style.fontSize = “20px”;
```

- **Atributo cssText.**
  - La propiedad *style* incluye el atributo cssText, que permite asignar varias propiedades CSS al mismo tiempo.
  - En este caso las propiedades se escriben igual que en CSS, es decir, usando guiones para aquellas con nombres compuestos.
  - Sintaxis:

    - Opción 1:

```
document.método(parámetros).style.cssText = “propiedad1: valor; propiedad2: valor; ….”;
```

    - Opción 2:

```
var variable2 = document.método(parámetros);
variable2. style. cssText = “propiedad1: valor; propiedad2: valor; …”;
```

  - Ejemplos:

    - Opción 1:

```
<p id = “p1”>Hola</p>
document.getElementById(“p1”).style.cssText = “color: #f00; font-size: 20px”;
```

    - Opción 2:

```
<p id = “p1”>Hola</p>
var parrafo = document.getElementById(“p1”);
parrafo. style.cssText = “color: #f00; font-size: 20px”;
```

Añadir carga de estilos con prompt() y propiedad className.

SPRITES.

- Una página web puede necesitar cargar muchísimas imágenes, tanto para usarlas como fondo, como para usarlas como objetos o parte del contenido.
- Las imágenes como fondo se manejan con propiedades como *backgroung-image*, *background-size*, *background-repeat y background-position.*
- **Concepto.**
    - Un *sprite* es un conjunto de imágenes en una sola imagen, de modo que, cargando una sola imagen, se tienen disponibles todas las que la conforman y van a usarse en el documento web sin necesidad de descargarlas del servidor cuando se necesiten. Así, se puede mejorar el tiempo de carga de una página web al reducir el número de peticiones a un servidor web.
    - Cargar una imagen implica una petición al servidor, una respuesta de éste, la carga de la imagen en el navegador, etc. Aunque una imagen sea pequeña y pese poco, todas las peticiones que se realicen consumen tiempo y ralentizan la carga de una página.
- **Utilidad**.
    - Accediendo con CSS a la posición de cada imagen, se pueden visualizar todas las que forman el *sprite* como si fueran imágenes únicas e independientes y estuvieran guardadas en archivos diferentes.
    - Usando pseudoclases CSS como *hover* o scripts de JavaScript se pueden conseguir cambios de estado o de posición de la imagen.
    - También pueden crearse efectos de movimiento y animaciones usando JavaScript u otros lenguajes de programación.
- **Creación de un Sprite.**
    - Se pueden usar programas de edición imagen como Photoshop o Gimp, o crearlas en línea subiendo las imágenes a webs que permiten generar la imagen única con todas ellas.
        - https://www.toptal.com/developers/css/sprite-generator/

TRANSICIONES.

- Permiten cambiar de un estado a otro de forma gradual o suave al realizarse una acción.

Propiedades de transición.

- **transition-property.**
    - Permite especificar que propiedad CSS se verá afectada por la transición.
    - No todas las propiedades CSS son “animables” mediante esta propiedad.
    - Sintaxis:
        - transition-property: valor;
        - valores:
            - **all.**
                - Aplica la transición a todas las propiedades CSS.
            - **none.**
                - No se aplica la transición, el cambio de estado se produce de golpe.
            - **propiedad.**
                - Aplica la transición solo a la propiedad CSS especificada.
    - Ejemplos:

- Transición con una sola propiedad:

```
#enlace:hover{
font-size:30px;
transition-duration: 0.5s;
transition-timing-function: linear;
```

- En el ejemplo, el texto del enlace aumentará en 5 décimas de segundo de forma lineal desde el tamaño por defecto hasta los 30 píxeles.
- Transición con una sola propiedad:

```
#enlace:hover{
font-size:30px;
font-family: Verdana;
transition-property:all;
transition-duration: 1s;
transition-timing-function: linear;
```

- En el ejemplo, el texto del enlace aumentará en un segundo de forma constante desde el tamaño por defecto hasta los 30 píxeles. También cambiará la fuente por defecto, a Verdana en ese mismo tiempo.

- **transition-duration.**
  - Especifica la duración de la transición de principio a fin.
  - El tiempo se mide en segundos, incluyendo la letra "s" al valor numérico.
  - A mayor valor, transición más lenta.
  - Sintaxis:
    - transition-duration: tiempo en segundos;
  - Ejemplos:
    - transition-duration: 2s; // La transición durara 2 segundos.
    - transition-duration: 0.3s; // La transición durará 3 décimas de segundo.
- **transition-timing-function.**
  - Establece el ritmo al que avanza la transición.
  - Sintaxis:
    - transition-timing-function: valor;
    - valores:
      - **ease.**
        - La transición comienza lentamente, continua rápido y termina de nuevo lentamente.
        - Valor por defecto.
      - **linear.**
        - La transición avanza a ritmo constante.
      - **ease-in.**
        - La transición comienza lentamente y continua y termina a ritmo constante.
      - **ease-out.**
        - La transición comienza y continua a ritmo normal constante y termina lentamente.
      - **ease-in-out.**

        - La transición comienza lentamente, continua a ritmo normal constante y termina lentamente.
    - **cubic-bezier(valor1, valor2, valor3, valor4).**
        - Permite establecer un ritmo de transición personalizado usando curvas de Bézier.
        - Los valores, que están comprendidos entre 0 y 1 con decimales y pueden ser positivos o negativos, representan los siguientes puntos:
            - **valor 1.**
                - Valor del eje X del primer punto de control de la curva.
            - **valor 2.**
                - Valor del eje Y del primer punto de control de la curva.
            - **valor 3.**
                - Valor del eje X del segundo punto de control de la curva.
            - **valor 4.**
                - Valor del eje Y del segundo punto de control de la curva.
        - En la web https://cubic-bezier.com se puede obtener los valores de forma gráfica.
- Ejemplos:
    - transition-timing-function: linear;
    - transition-timing-function: ease-out;
    - transition-timing-function: cubic-bezier(0.59,0.27,0.67,1);

- **transition-delay.**
    - **Permite retrasar el inicio de la transición** un tiempo determinado.
    - El tiempo se mide en segundos, incluyendo la letra “s” al valor numérico.
    - Si no se incluye, la transición empieza inmediatamente.
    - Sintaxis:
        - transition-delay: tiempo en segundos;
    - Ejemplos:
        - transition-delay: 0,5s; // La transición comenzara transcurridas 2 décimas de segundo.
- **Notación abreviada.**
    - Se pueden usar algunas o todas las propiedades de transición usando la propiedad *transition*.
    - Los valores que no se quieran usar, se pueden no incluir.
    - Si se quieren usar varias propiedades con distintos tiempos, se separa las características de cada una con una coma.
    - Sintaxis:
        - transition: property duration timing-function delay;
    - Ejemplos:
        - transition: all 1s linear; // Se animan todas las propiedades a ritmo constante durante un segundo.
        - transition: color 0.5s linear 1s; // Efecto de transición que comienza con un segundo de retraso, dura medio segundo y sigue un ritmo normal continuo.

- transition: color 3s, font-size 0.5s ease-in, background-color 1s; // Se animan todas las propiedades durando cada una, un tiempo distinto y a ritmos diferentes.

EVENTOS.

Concepto.

- Acción del usuario sobre un elemento HTML que provoca una respuesta.
- Mecanismo principal de interacción usuario-página Web.

Manejador de eventos.

- A los eventos hay que asociarles una función o el código JavaScript que se ejecutará al producirse el evento.
- Dichas funciones o código son los manejadores de eventos.
- Para diferenciar el manejador del evento del evento en sí, se antepone el prefijo "on" al manejador.
  - Evento: click.
  - Manejador: onclick.

Tipos de eventos:

- **click**:
  - Pulsar con el botón izquierdo del ratón sobre un elemento.
  - HTML: Elementos de selección de datos, botones de formulario, <a> e <img>.
- **dblclick:**
  - Se hace doble clic con el botón izquierdo del ratón sobre un elemento.
  - HTML: Elementos de selección de datos, botones de formulario, <a> e <img>.
- **mouseover**:
  - El puntero del ratón entra en el área que ocupa un elemento, pasa por encima.
  - HTML: Todos elementos HTML y en especial, elementos de formulario, <a> e <img>.
- **mouseout:**
  - El puntero del ratón sale del área que ocupa un elemento, se quita de encima.
  - HTML: Todos elementos HTML y en especial, elementos de formulario, <a> e <img>.
- **load**:
  - Se produce cuando el navegador termina de cargar la página, incluyendo la carga recursos como hojas de estilo, archivos con script o imágenes.
  - HTML: <body>, <window>, <img> y <script>.
- **unload**:
  - Se produce cuando se abandona la página porque se cierra el navegador o se cambia de página.
  - HTML: <body>, <window>
- **focus**:
  - Un elemento de la página recibe el foco.
  - HTML: <body>, <input>, <select>, <textarea>.
- **blur**:
  - Un elemento de la página pierde el foco.
  - HTML: <body>, <input>, <select>, <textarea>.

Evento	Descripción	Elementos para los que está definido
onblur	Deseleccionar el elemento	<button>, <input>, <label>, <select>, <textarea>, <body>
onchange	Deseleccionar un elemento que se ha modificado	<input>, <select>, <textarea>
onclick	Pulsar y soltar el ratón	Todos los elementos
ondblclick	Pulsar dos veces seguidas con el ratón	Todos los elementos
onfocus	Seleccionar un elemento	<button>, <input>, <label>, <select>, <textarea>, <body>
onkeydown	Pulsar una tecla (sin soltar)	Elementos de formulario y <body>
onkeypress	Pulsar una tecla	Elementos de formulario y <body>
onkeyup	Soltar una tecla pulsada	Elementos de formulario y <body>
onload	La página se ha cargado completamente	<body>
onmousedown	Pulsar (sin soltar) un botón del ratón	Todos los elementos
onmousemove	Mover el ratón	Todos los elementos
onmouseout	El ratón *"sale"* del elemento (pasa por encima de otro elemento)	Todos los elementos
onmouseover	El ratón *"entra"* en el elemento (pasa por encima del elemento)	Todos los elementos
onmouseup	Soltar el botón que estaba pulsado en el ratón	Todos los elementos
onwheel	Se produce al girar la rueda del ratón sobre un elemento.	
onreset	Inicializar el formulario (borrar todos sus datos)	<form>
onresize	Se ha modificado el tamaño de la ventana del navegador	<body>
onselect	Seleccionar un texto	<input>, <textarea>
onsubmit	Enviar el formulario	<form>
onunload	Se abandona la página (por ejemplo, al cerrar el navegador)	<body>
oncontextmenu	Se produce al pulsar el botón derecho del ratón en cualquier elemento.	Todos los elementos

FORMAS DE ASOCIAR UN MANEJADOR A EVENTO.

A. Como atributo HTML del elemento.

- Sintaxis:
  - manejador = “código a ejecutar”
  - manejador = “función()”
- Ejemplos:
  - Con función:

```
fuction pulsar()
{
        alert(“¡Has pulsado el botón!”);
}
<input type =”button” name = “boton1” value = “Pulsar” onclick = “pulsar()”>
```

  - Con código directo:

<input type =”button” name = “boton1” value = “Pulsar” onclick = “alert(‘¡Has pulsado el botón!’)”>

- Si se quieren ejecutar varias funciones al producirse el evento, se pueden realizar 2 acciones distintas:
  - **Opción1**:
    - Crear una función que englobe a todas las demás.

```
function varias()
{
   funcion1();
   funcion2();
   funcion3();
 }

function funcion1(){
   alert("funcion uno");
}
function funcion2(){
   alert("funcion dos");
}
function funcion3(){
   alert("funcion tres");
}
<input type =”button” name = “boton2” value = “Pulsar” onclick = “varias()”>
```

  - **Opción2**:
    - Separar cada función con un punto y coma (;).
    - <input type =”button” name = “boton2” value = “Pulsar” onclick = “funcion1();funcion2();funcion3()”>

B. Con un manejador semántico.

- Añadir un manejador a una variable, constante, array, objeto, etc.
- No es necesario incluir elementos adicionales en el HTML.
- Se asocia el manejador a una función para ejecutar cuando se produzca el evento.
- Nombre del evento se escribe siempre en minúscula.
- El nombre de la función se escribe sin comillas y sin paréntesis.
- Sintaxis:
  - elemento.manejador = función;
- Ejemplo:

```
<p id = “p1”>Párrafo1</p>
function pulsar()
{
        alert(“¡Has pulsado dentro del párrafo!”);
}
 var pa =document.getElementById(“p1”);
 pa.onclick = pulsar;
 document.getElementById(“p1”).onclick = pulsar; // Alternativa a crear una variable para cargar el nodo.
```

C. Con un manejador de eventos escuchador añadido a un elemento.

- **Añadir un manejador.**
    - Se usa el método addEventListener().
    - Permite agregar un manejador de eventos a un elemento DOM para que al producirse un evento sobre el elemento se ejecute una función.
    - Sintaxis:
        - elemento.addEventListener("evento", función sin paréntesis, flujo) ;
        - Elemento:
            - variable, array, etiqueta HTML.
        - Evento:
            - Se trata del evento, no del manejador, luego se escribe sin "on".
        - Función:
            - Función que contiene las acciones que se ejecutarán si se produce el evento.
            - Se escribe el nombre de la función sin sus paréntesis.
            - Para llamar a los métodos o propiedades necesarios para realizar las acciones, se antepone el nombre de la variable si sólo se ha cargado un elemento con un método get, si no, si se han obtenido varios elementos, se antepone el operador "this" para referirse a arrays u objetos que contienen múltiples nodos.
            - Varias funciones pueden ejecutarse si crea una que las contenga a todas las demás.
        - Flujo:
            - Valor booleano (true o false).
                - **false**:
                    - Ejecución desde el más bajo nivel al más alto. (Event bubbling).
                - **true**:
                    - Ejecución desde el más alto nivel al más bajo. (Event capturing).
                    - Valor por defecto.
    - Utilización:
        - Crear la función o funciones que se serán ejecutadas al producirse un determinado evento.
        - Cargar el elemento o elementos en una variable u array usando métodos get.

        - Añadir el manejador de eventos (addEventListener()), al elemento u elementos.
    - Ejemplos:
        - Con un solo elemento:

```
<h1 id = ”titulo”>Título de la web</h1>
function cambiarColor()
 {
        titulo.style.color = "yellow";
 }
 var texto = document.getElementById("titulo");
 texto.addEventListener("dblclick",cambiarColor);
```

        - Con varios elementos:

```
<p id = ”p1”>Párrafo 1</p>
<p id = ”p2”>Párrafo 2</p>

<p id = ”p3”>Párrafo 3</p>
function cambiarColor()
 {
        this.color = "yellow"; // this se refiere al array párrafos.
 }
 var parrafos = document.getElementsByTagName("p");
 parrafos.addEventListener("dblclick",cambiarColor);
```

- **Quitar un manejador.**
    - Se usa el método removeEventListener().
    - Elimina el manejador de eventos que se ha aplicado a un elemento DOM.
    - Sintaxis:
        - elemento.removeEventListener(“evento”, función sin paréntesis, flujo) ;

EVITAR LA ACCIÓN DE UN EVENTO.

- **preventDefault().**
    - Permite cancelar la acción de un evento, si ésta se puede cancelar.
    - La acción cancelada depende del evento sobre el que se aplique este método, por ejemplo:
        - Si se hace clic sobre un botón no ocurriría nada.
        - Si se pulsa en el botón de enviar en un formulario, éste no se enviará.
        - Si se hace clic con el botón derecho sobre la ventana no se abrirá el menú contextual.
    - No todos los eventos son cancelables.
    - Este método se utiliza dentro de un método manejador de eventos, como addEventListener().
    - Sintaxis:
        - Objeto de tipo evento. preventDefault();
            - **Objeto de tipo evento**.

        - Todos los manejadores de eventos reciben como parámetro el objeto evento que se ha generado.
        - Éste se va a utilizar para llamar al método preventDefault().
    - Ejemplo:
        - document.getElementById("form1").addEventListener("submit", function(e) {e.preventDefault();})
        - **Parámetro *e* en el ejemplo.**
            - Representa al objeto que se genera.
- **cancelable.**
    - Propiedad que indica si un evento es o no cancelable y, por tanto, si se puede, o no, usar el método preventDefault() con él.
    - Devuelve un valor booleano (true o false) y, obviamente, puede ser cancelado cuando devuelve al valor verdadero.
    - Sintaxis.
        - e.cancelable;
        - **Objeto e**:
            - Objeto que se genera al producirse un evento.
    - Ejemplo:
        - let esCancelable =e.cancelable;
        - alert("¿Este evento es cancelable? " + esCancelable);
        - e.preventDefault();

EVENTOS DE TIEMPO.

- Son temporizadores que se usan para retrasar o repetir la ejecución de código (una función, por ejemplo), un determinado intervalo de tiempo.
- Ejecutar una función transcurrido un tiempo determinado, se denomina *planificar una llamada*.
- Existen 2 **métodos** nativos de JavaScript que se utilizan en los eventos de tiempo.
    - **setTimeout**.
        - Permite ejecutar una función pasado un intervalo de tiempo determinado.
        - Sintaxis:
            - setTimeout(Función o código, tiempo);
            - Función.
                - Función o código que se llamará o ejecutará transcurrido el tiempo especificado como segundo parámetro.
            - Tiempo.
                - Tiempo expresado en milisegundos que debe transcurrir antes de ejecutarse la función incluida como parámetro.
                - 1000 milisegundos = 1 segundo.
        - Ejemplos:

```
function saludar()
{
    alert("¡Buenos días!");
}
setTimeout(saludar,2000); // Transcurridos 2 segundos se muestra el saludo.
```

    - **setInterval.**
        - Permite ejecutar una función de forma repetida pasado un intervalo de tiempo.
        - Sintaxis:
            - setInterval(Función o código, tiempo);

- Función.
  - Función o código que se llamará o ejecutará transcurrido el tiempo especificado como segundo parámetro y volverá a repetirse cada vez que el intervalo temporal se cumpla.
- Tiempo.
  - Tiempo expresado en milisegundos que debe transcurrir antes de ejecutarse cada repetición de la función o código incluido como parámetro.

- Ejemplos:

```
function saludar()
{
        alert("¡Buenos días!");
}
setInterval(saludar,2000); // Cada 2 segundos, se repite el saludo.
```

Identificador de temporizador. (intervalID).

- Cuando setTimeout() o setInterval(), se ejecutan, se les asigna un número o identificador único (intervalID), que puede ser guardado en una variable.
- A través de dicha variable, podemos, por ejemplo, conocer qué temporizador está ejecutándose.
- También sirve para detener los temporizadores.
- Sintaxis:
  - variable = setTimeout(Función o código, tiempo);
  - variable = setInterval(Función o código, tiempo);
  -
- Ejemplo:
  - let identificador = setTimeout(calcular, 2000);
  - let intervalo = setInterval(saludar, 5000);

Detener eventos de tiempo.

- Para detener un temporizador, se crea una función para borrarlo a la que se le pasa la variable que contiene el identificador del temporizador que se quiere parar.
- **clearInterval().**
  - Emparejado con setInterval()
  - Sintaxis:

```
function detenerTemporizador()
 {
  clearInterval(variable con identificador);
 }
```

- **clearTimeout()**
  - Emparejado con setTimeout().
  - Sintaxis:

```
function detenerTemporizador()
 {
  clearTimeout (variable con identificador);
 }
```