Universidad Nacional de Rosario

Argentina programa 4.0

Programación
Web Full Stack PHP

Introducción a JavaScript

Conceptos generales del lenguaje

¿Qué es JavaScript?

JavaScript es el lenguaje de programación más popular del mundo.
Dentro de la programación de páginas web, JavaScript es uno de los 3 lenguajes que todos los desarrolladores web deben aprender. Los 3 lenguajes que son bases y pilares de la programación web son:

1. HTML utilizado para definir el contenido de las páginas web
2. CSS utilizado para especificar el diseño de las páginas web
3. JavaScript utilizado para programar el comportamiento de las páginas web

JavaScript es un lenguaje de programación que se utiliza principalmente para crear páginas web dinámicas. Una página web dinámica es aquella que incorpora efectos como texto que aparece y desaparece, animaciones, acciones que se activan al pulsar botones y ventanas con mensajes de aviso al usuario, entre otras cosas.
JavaScript es un lenguaje interpretado que se embebe en una página web HTML y permite incorporar lógica y funcionalidad.
Es un lenguaje interpretado, esto significa que las instrucciones las analiza y procesa el navegador en el momento que deben ser ejecutadas. Cada navegador tiene un intérprete Javascript, que varía en función del navegador.
JavaScript y Java son lenguajes completamente diferentes, tanto en concepto como en diseño.
JavaScript fue inventado por Brendan Eich en 1995 y se convirtió en un estándar ECMA en 1997. ECMA-262 es el nombre oficial del estándar.
ECMAScript es el nombre oficial del lenguaje.
Con Javascript se programan scripts, pero...

¿Qué es un script?

Un script de JavaScript es un conjunto de instrucciones o líneas de código que se escriben en el lenguaje de programación JavaScript. Estas instrucciones le indican al navegador web qué acciones debe realizar en una página web determinada.

Un script permite interactuar con elementos de una página web, realizar cálculos, modificar contenido, responder a eventos y muchas otras acciones.

Anteriormente dijimos que JavaScript se embebe dentro del código HTML pero...

¿Dónde se escribe el código JavaScript dentro de un archivo HTML?

Para utilizar un script de JavaScript, primero debes incluirlo en una página web. Esto se hace mediante una etiqueta '<script>' dentro del código HTML de la página. Por lo general, se coloca en la sección '<head>' o antes de cerrar la etiqueta '</body>'.

En este ejemplo, el código de JavaScript se encuentra en un archivo llamado "script.js". La etiqueta '<script>' con el atributo 'src' se utiliza para especificar la ubicación del archivo.

```html
<html>
    <head>
        <title>Mi página web</title>
    </head>
    <body>
        <!-- Contenido de la página -->
        <script src="script.js"></script>
    </body>
</html>
```

Con un script de JavaScript, se pueden realizar una gran cantidad de acciones en una página web, como:

- Manipular el contenido HTML y los estilos de los elementos de la página.
- Realizar cálculos matemáticos y manipulaciones de datos.
- Obtener y modificar valores de formularios.
- Responder a eventos, como hacer clic en un botón o desplazar el mouse.
- Realizar solicitudes a servidores para obtener o enviar datos.

Ejemplos de uso común de scripts de JavaScript:

- Validación de formularios: se puede utilizar JavaScript para verificar si los campos de un formulario están completos antes de enviarlos.
- Creación de interactividad: se puede agregar interactividad a una página web, como mostrar y ocultar elementos, cambiar el contenido dinámicamente, crear animaciones, etc.
- Manipulación del DOM: se puede utilizar JavaScript para acceder y modificar elementos de la página, como cambiar el color de un texto, agregar nuevos elementos o cambiar el tamaño de una imagen.

El contenido dentro de la etiqueta <script> y el código HTML lo veremos más adelante, en este momento el objetivo es entender que es un script, donde se incorporan y para qué sirven.

Los scripts también se pueden colocar en archivos externos al HTML. Los scripts externos son prácticos cuando se usa el mismo código en muchas páginas web diferentes.
Para usar una secuencia de comandos externa, coloque el nombre del archivo de secuencia de comandos en el src atributo (fuente) de una <script>etiqueta:
Ejemplo:

```
9
10      <script src="myScript.js"></script>
11
```

Los archivos JavaScript tienen la extensión de archivo .js →

Ventajas de archivos de JavaScript externos:

Colocar scripts en archivos externos tiene algunas ventajas:

- Separa HTML del código JavaScript.
- Hace que HTML y JavaScript sean más fáciles de leer y mantener.
- Los archivos JavaScript almacenados en caché pueden acelerar la carga de la página.

Para agregar varios archivos de script a una página, use varias etiquetas de script. Por ejemplo:

```
12      <script src="myScript1.js"></script>
13      <script src="myScript2.js"></script>
```

Se puede hacer referencia a un script externo de 3 maneras diferentes:

- Con una URL completa (una dirección web completa)
- Con una ruta de archivo (como /js/)
- Haciendo referencia al nombre del archivo

Este ejemplo usa una URL completa para vincular a myScript.js:

```
15      <script src="https://www.w3schools.com/js/myScript.js"></script>
```

Este ejemplo utiliza una ruta de archivo para vincular a myScript.js:

```
18      <script src="/js/myScript.js"></script>
```

Este ejemplo no usa una ruta para vincular a myScript.js:

```
10      <script src="myScript.js"></script>
```

¿Qué es una función (function)?

En JavaScript, una función es un bloque de código reutilizable que realiza una tarea específica. Las funciones nos permiten agrupar un conjunto de instrucciones y ejecutarlas cuando las necesitamos. Usamos funciones para evitar repetir el mismo código una y otra vez, y para organizar y estructurar nuestro programa de manera más clara.

Para declarar una función en JavaScript, utilizamos la palabra clave 'function', seguida del nombre de la función y un par de paréntesis. A continuación, se utiliza un bloque de código entre llaves '{}' para definir las instrucciones que la función ejecutará.

En este ejemplo, hemos declarado una función llamada 'saludar()' que imprime el mensaje "Hola, bienvenido" en la consola.

```
// declaro la funcion saludar
function saludar() {
  console.log("Hola, bienvenido");
}
// llamo a la funcion saludar
saludar();
```

Una función puede aceptar parámetros, que son valores que se pasan a la función al momento de llamarla. **Los parámetros son variables locales que la función puede utilizar en su bloque de código.**

En este ejemplo, hemos agregado un parámetro 'nombre' a la función 'saludar()'. Al llamar a la función y pasarle el parámetro, este valor se asignará a la variable 'nombre' dentro de la función.

```
// declaro la funcion saludar
function saludar(nombre) {
  console.log("Hola " + nombre);
}
/* llamo a la funcion saludar
   con el parametro 'Mateo' */
saludar("Mateo");
```

Al llamar a la función saludar con el parámetro 'NOMBRE_PERSONA' el parámetro nombre de la función pasa a tener el valor de 'NOMBRE_PERSONA', por lo que en nuestra consola se vería lo siguiente:

```
¡Hola, NOMBRE_PERSONA!                                arg-programa.js:2
```

Una función puede devolver un valor utilizando la palabra clave 'return'. El valor que se devuelve se puede utilizar en otros lugares del programa.

En este caso, la función 'sumar()' recibe dos parámetros 'a' y 'b', y devuelve la suma de estos dos valores mediante la palabra clave 'return'. En este caso la variable resultado es igual a 30.

```
function sumar(a, b) {
  return a + b;
}
/* declaro una variable llamada resultado donde
guardaré el valor que retorna la funcion sumar */
let resultado;
/* guardo en resultado el valor del
return de la funcion sumar */
resultado = sumar(18,12);
```

¿Cómo ejecutar una función?

Para ejecutar una función, simplemente escribimos su nombre seguido de paréntesis. Si la función acepta parámetros, los pasamos dentro de los paréntesis.

Llamamos a la función 'saludar()' pasando el argumento "Juan". La función imprimirá el mensaje "Hola Juan" en la consola.

```
saludar("Juan");
// Output: ¡Hola, Juan!
```

Ejemplos de uso común de funciones:

- **Realizar cálculos:** se pueden crear funciones para realizar operaciones matemáticas, como sumar, restar, multiplicar o dividir números.

```
function sumar(a, b) {
  return a + b;
}
let resultado = sumar(3, 5);
console.log(resultado);
// Output: 8
```

- **Realizar tareas repetitivas:** si necesitas realizar una tarea repetitiva en tu programa, puedes encapsularla en una función y llamarla cada vez que la necesites.

```
function imprimirMensaje() {
  console.log("Este es un mensaje.")
}
imprimirMensaje();
// output: "Este es un mensaje"
imprimirMensaje();
// output: "Este es un mensaje"
```

- **Manipular y transformar datos:** las funciones también son útiles para manipular y transformar datos. Puedes crear funciones que convierten una cadena de texto a mayúsculas, calculen la longitud de una cadena, o realicen cualquier otro tipo de manipulación.

```
function convertirAMayusculas(texto) {
  return texto.toUpperCase();
}

let result = convertirAMayusculas("hola");
console.log(resultado); // Output: HOLA
```

- **Interactuar con el usuario:** se pueden utilizar funciones para solicitar datos al usuario y realizar acciones basadas en sus respuestas.

  En este caso, la función 'saludarUsuario()' recibe el nombre del usuario como argumento y lo utiliza para imprimir un saludo personalizado.

```
function saludarUsuario(nombre) {
  console.log("¡Hola, " + nombre + "!");
}

let nombre = prompt("Ingresa tu nombre:");
saludarUsuario(nombre);
```

Las funciones en JavaScript son una herramienta fundamental para estructurar y organizar el código, reutilizar tareas y manipular datos. Las funciones se crean según la necesidad del programa.

¿Qué es una variable?

Una variable es un espacio de almacenamiento utilizado para guardar cualquier tipo de dato como una cadena de caracteres, un valor numérico o estructuras de datos un poco más específicas. Al crear el código del programa, se deben declarar las variables que utiliza el programa y así reservar el espacio en memoria.

Crear una variable en JavaScript se llama "declarar" una variable.

¿Cómo se declara una variable?

JavaScript tiene tres tipos de declaraciones de variables.

var : declara una variable, opcionalmente la inicia a un valor.

let : declara una variable local con ámbito de bloque, opcionalmente la inicia a un valor.

const : declara un nombre de constante de solo lectura y ámbito de bloque.

Por ejemplo:

```
let residencia;
var apellido;
```

Después de la declaración, la variable no tiene valor (técnicamente es undefined).

Para asignar un valor a la variable, utilice el signo igual:

```
residencia = "Rosario";
```

```
apellido = "Perez";
```

También puede asignar un valor a la variable cuando la declara:

```
let nombre = "Guillermina";
```

Los puntos y comas separan las declaraciones de JavaScript.

Agregue un punto y coma al final de cada instrucción ejecutable:

```
let a, b, c;  // Declare 3 variables llamadas a, b, c
a = 5;        // Asigne el valor 5 a la variable: a
b = 6;        // Asigne el valor 6 a la variable: b
c = a + b;    // Asigne el valor correspondiente a la suma de: a + b a la variable: c
```

JavaScript ignora varios espacios. Puede agregar espacios en blanco a su secuencia de comandos para que sea más legible. Por lo que las siguientes líneas son equivalentes:

```
let persona = "Diego";
```

```
let persona="Diego";
```

Una buena práctica es poner espacios alrededor de los operadores (= + - * /):

```
let numero1 = 10; //Buena práctica dejar el espacio
let numero2 = 20;
let suma = numero1 + numero2;
```

JavaScript usa las palabras claves var, let y const para declarar variables .
Se utiliza un signo igual para asignar valores a las variables.
En JavaScript, **el signo igual (=) es un operador de "asignación"**, no un operador "igual a".

En este ejemplo, x se define como una variable. Entonces, a x se le asigna el valor 20

```
let x;
x = 20;  //Buena práctica dejar el espacio
```

¿Qué valores puede contener una variable?

Las variables de JavaScript pueden contener números como el número 100 y valores de texto como "John Cena".
En programación, los valores de texto se denominan cadenas de texto (también llamadas string).
JavaScript puede manejar muchos tipos de datos, pero por ahora, solo vamos a centrarnos en números y cadenas.
Las cadenas se escriben entre comillas simples o dobles. Los números se escriben sin comillas.
Si se pone un número entre comillas, se tratará como una cadena de texto.
Ejemplo:

```
const numero_pi = 3.14; // esto es un número
let nombrePersona = "John Cena"; // esto es una cadena de texto
let respuesta = "Si, soy yo!"; // esto es una cadena de texto
const edad = "21"; //esto es una cadena de texto
```

Nombres de variables:

Todas las variables de JavaScript deben identificarse con nombres únicos .

Estos nombres únicos se denominan identificadores .

Los identificadores pueden ser nombres cortos (como x e y) o nombres más descriptivos (edad, suma, volumen_total).

Las reglas generales para construir nombres para variables (identificadores únicos) son:

- Los nombres pueden contener letras, dígitos, guiones bajos y signos de dólar.
- Los nombres deben comenzar con una letra.
- Los nombres también pueden comenzar con $ y _ (pero no lo usaremos en este curso, usar el guión bajo no es muy común en JavaScript, pero una convención entre los programadores es usarlo para variables "privadas/ocultas").
- Los nombres distinguen entre mayúsculas y minúsculas (y e Y son variables diferentes).
- Las palabras reservadas (como las palabras clave de JavaScript) no se pueden usar como nombres.

Los identificadores de JavaScript distinguen entre mayúsculas y minúsculas.

Algunos ejemplos de nombres de variables son:

```
let persona_nombre;
let contador21;
let $edad;
let _password_api;
```

Palabras reservadas:

Las palabras reservadas en JavaScript son palabras especiales que **tienen un significado y una funcionalidad específica en el lenguaje.** Estas palabras están reservadas y no pueden ser utilizadas como nombres de variables, funciones o cualquier otro identificador en nuestro código.

Las palabras reservadas son fundamentales en JavaScript porque ayudan a definir la estructura y la lógica del código. Cada palabra reservada tiene una función específica y se utiliza para realizar tareas particulares.Aquí tienes algunos ejemplos de palabras reservadas en JavaScript:

- ‘var’, ‘let’, ‘const’: estas palabras reservadas se utilizan para declarar variables en JavaScript. ‘var’ se usaba anteriormente, pero ahora se recomienda utilizar 'let' o 'const' para declarar variables.
- ‘if’, ‘else if’, ‘else’: estas palabras reservadas se utilizan para realizar evaluaciones condicionales en el código. Permiten ejecutar diferentes bloques de código dependiendo de si se cumple una condición o no.
- ‘for’, ‘while’, ‘do while’: estas palabras reservadas se utilizan para crear bucles en JavaScript. Los bucles permiten repetir un bloque de código varias veces, facilitando la automatización de tareas.
- ‘function’: esta palabra reservada se utiliza para declarar una función en JavaScript. Las funciones son bloques de código que se pueden reutilizar y ejecutar cuando se necesiten.
- ‘return’: esta palabra reservada se utiliza dentro de una función para devolver un valor específico. Cuando se alcanza la palabra 'return', la función finaliza y devuelve el valor especificado.

Estos son solo algunos ejemplos de palabras reservadas en JavaScript. Existen muchas otras palabras reservadas que tienen diferentes propósitos y funcionalidades dentro del lenguaje.

Es esencial recordar que las palabras reservadas están reservadas únicamente en el contexto de JavaScript. Esto significa que no puedes utilizar estas palabras como nombres de variables, funciones o identificadores en tu código, ya que JavaScript las reconoce como palabras especiales y les da un significado específico.

Tipos de datos:

En JavaScript, los tipos de datos definen qué tipo de información puede ser almacenada y manipulada en una variable. Los tipos de datos determinan cómo se representa y se opera con los valores en un programa. Cada valor tiene un tipo de datos asociado.

Los tipos de datos primitivos en JavaScript son:

- Números (tipo de dato: **Number**): representa valores numéricos, como 5, 3.14, -10, etc.
- Cadena de texto (tipo de dato: **String**): representa una secuencia de caracteres, como "Hola", "JavaScript", etc.
- Booleano (tipo de dato: **Boolean**): representa un valor lógico verdadero o falso, es decir, 'true' o 'false'.
- Nulo (tipo de dato: **Null**): representa la ausencia de valor. Es un tipo de dato especial que solo tiene un valor posible: 'null'.
- Indefinido (tipo de dato: **Undefined**): representa una variable que ha sido declarada pero no ha sido asignada con ningún valor. Es el valor inicial de una variable.

En JavaScript, también existen tipos de datos no primitivos, que se conocen como objetos. Algunos ejemplos de estos tipos de datos son:

- Objeto (tipo de dato: **Object**): representa una colección de propiedades y valores asociados. Los objetos se utilizan para representar entidades complejas y pueden tener múltiples propiedades y métodos.
- Arreglo (tipo de dato: **Array**): representa una colección ordenada de valores. Los arreglos se utilizan para almacenar múltiples valores en una sola variable.
- Función (tipo de dato: **Function**): representa un bloque de código reutilizable que se puede ejecutar cuando se llama. Las funciones se utilizan para agrupar un conjunto de instrucciones y realizar una tarea específica.

En JavaScript, es posible convertir un tipo de datos en otro. Por ejemplo, se puede convertir un número en una cadena de texto o viceversa. Esto se puede lograr utilizando funciones y métodos específicos para cada tipo de dato.

El operador typeof:

El operador 'typeof' en JavaScript se utiliza para averiguar el tipo de datos de una variable. Nos permite saber si una variable es un número, una cadena de texto, un objeto, una función u otro tipo de dato. Con 'typeof', podemos realizar acciones diferentes dependiendo del tipo de datos con el que estamos trabajando. Es útil para verificar y gestionar correctamente los diferentes tipos de información en nuestro programa.

Podemos utilizar el operador 'typeof' seguido de una variable para obtener el tipo de datos de esa variable. Por ejemplo:

```
let nombre = "Juan";
console.log(typeof nombre);
```

En este caso, el operador 'typeof' nos devuelve '"string"', que indica que la variable 'nombre' es una cadena de texto.

El operador 'typeof' devuelve una cadena de texto que representa el tipo de datos de la variable. Algunos de los valores de retorno más comunes son:

- "number": para variables numéricas.
- "string": para variables de texto.
- "boolean": para variables booleanas (verdadero o falso).
- "undefined": para variables sin valor asignado.
- "object": para variables que son objetos o null.
- "function": para variables que son funciones.

Ejemplos de cómo usar el operador 'typeof':

```
let edadPersona = 25;
console.log(typeof edadPersona); // Output: "number"

let esActivo = true;
console.log(typeof esActivo); // Output: "boolean"

let persona1 = { nombre: "Juan", edad: 30 };
console.log(typeof persona1); // Output: "object"

let saludar = function() {
  console.log("Hola");
};
console.log(typeof saludar); // Output: "function"
```

En cada caso, el operador 'typeof' nos devuelve el tipo de datos correspondiente para cada variable.

El operador 'typeof' nos permite realizar acciones específicas según el tipo de datos de una variable. Por ejemplo, podemos validar si una variable es un número, ejecutar una función si es un objeto o realizar operaciones diferentes dependiendo del tipo de datos.

Conversión de tipos de datos:

A continuación, veremos algunos ejemplos de cómo se realiza la conversión de tipos de datos:

Conversión de número a cadena de texto:

Para convertir un número a una cadena de texto, podemos utilizar la función 'toString()'. Esta función convierte el número en una representación en forma de cadena de texto.

En este ejemplo, el número '42' se convierte en la cadena de texto '"42"' mediante la función 'toString()'.

```
let numero = 42;
let cadena = numero.toString();
console.log(typeof cadena); // Output: "string"
```

Conversión de cadena de texto a número:

Para convertir una cadena de texto a un número, podemos utilizar las funciones 'parseInt()' y 'parseFloat()'. Estas funciones analizan la cadena y devuelven un número.

En este caso, la cadena de texto '"3.14"' se convierte en el número 3.14 utilizando la función 'parseFloat()'.

```
let cadena = "3.14";
let numero = parseFloat(cadena);
console.log(typeof numero); // Output: "number"
```

Es importante tener en cuenta que al realizar conversiones de tipos de datos, es posible que se produzcan resultados inesperados si no se comprende completamente cómo se comporta JavaScript en estos casos. Por eso, es recomendable tener precaución y realizar conversiones explícitas cuando sea necesario.

Tipo de dato OBJECT:

Usar un objeto en JavaScript es útil cuando necesitas agrupar información relacionada y acciones asociadas en un solo lugar. Imagina que quieres representar a una persona en tu programa. En lugar de tener variables separadas para el nombre, la edad y la profesión de la persona, puedes crear un objeto que contenga estas propiedades. Esto hace que el código sea más organizado y fácil de entender.

En JavaScript, los objetos son como "contenedores" que pueden guardar información y realizar acciones. Puedes pensar en ellos como objetos reales, como una persona, un auto o una casa. Los objetos en JavaScript nos permiten agrupar características y comportamientos relacionados en un solo lugar.

Para crear un objeto en JavaScript, podemos utilizar las llaves '{ }'. Dentro de esas llaves, podemos agregar propiedades y sus respectivos valores. Las propiedades son características del objeto, como el nombre, el color, la edad, etc.

Ejemplo de creación de un objeto:

```
let persona = {
  nombre: "Juan",
  edad: 30,
};
```

```
let coche = {
  marca: "Toyota",
  modelo: "Corolla",
  año: 2021,
};
```

Un objeto está compuesto por propiedades y valores. Las propiedades son como etiquetas que describen una característica del objeto, y los valores son los datos asociados a esas propiedades. Por ejemplo, si estamos creando un objeto para representar a una persona, podríamos tener propiedades como "nombre" y "edad" con sus respectivos valores.

Podemos acceder a las propiedades de un objeto utilizando la notación de punto ('objeto.propiedad') o la notación de corchetes ('objeto["propiedad"]'). Si queremos obtener el valor de una propiedad, podemos utilizar estas notaciones.

Siguiendo el ejemplo anterior del objeto persona:

```
let persona = {
  nombre: "Juan",
  edad: 30,
};
console.log(persona.nombre); // Output: "Juan"
/* estamos accediendo a la propiedad 'nombre' del objeto persona). */
```

Un método en un objeto es una función que puede realizar acciones o cálculos relacionados con el objeto. Es como una habilidad que tiene el objeto para hacer algo. Por ejemplo, un objeto "persona" puede tener un método llamado "saludar" que imprime un mensaje de saludo en la consola.

Ejemplo de un método de un objeto:

```
let perro = {
  ladrar: function () {
    console.log("Guau, guau");
  },
};

perro.ladrar(); // output: "Guau, guau"
/*
  accedemos al método ladrar del objeto perro. Al hacer esto se ejecuta el codigo
  del método ladrar que es un console.log que muestra el texto "Guau, guau".
*/
```

Podemos cambiar los valores de las propiedades de un objeto después de haberlo creado. Si queremos actualizar la información de un objeto, simplemente asignamos un nuevo valor a la propiedad correspondiente.

```
let persona = {
  nombre: "Juan",
  edad: 30
};

persona.edad = 31;
console.log(persona.edad); // Output: 31

/* Modificamos el valor de la propiedad edad
   del objeto persona asignándole el número 31,
   por lo que ahora dicha propiedad valdrá 31
   y no 30 como cuando se declaró el objeto. */
```

Además de crear objetos utilizando la notación de llaves, en JavaScript también podemos utilizar funciones constructoras para crear objetos. Una función constructora es como una plantilla que nos permite crear múltiples objetos con las mismas características y métodos predefinidos.

```
function Persona(nombre, edad) {
  this.nombre = nombre;
  this.edad = edad;
}

let persona1 = new Persona("Juan", 30);
let persona2 = new Persona("María", 28);

/* Creamos una función constructora llamada
   Persona, la cual recibe las dos propiedades
   del objeto que se desea crear. Por lo que
   luego para crear un objeto Persona, solo se
   deberá llamar a la función constructora
   pasandole entre ( ) las dos propiedades del
   objeto que en este caso son nombre y edad
*/
```

Tipo de dato ARRAY:

Un array en JavaScript es una estructura de datos que nos permite almacenar y organizar múltiples valores en una sola variable. Podemos pensar en él como una lista que contiene varios elementos. Los arrays son útiles cuando necesitamos trabajar con conjuntos de datos relacionados o cuando queremos almacenar múltiples valores bajo un mismo nombre.

Para crear un array en JavaScript, utilizamos corchetes '[]' y separamos los elementos con comas. Cada elemento puede ser cualquier valor, como números, texto u otros objetos.

En este ejemplo, creamos un array llamado ‘numeros’ que contiene una lista de números:

```
let numeros = [1, 2, 3, 4, 5];
```

Aquí creamos un array llamado ‘frutas’ que contiene diferentes nombres de frutas:

```
let frutas = ["manzana", "banana", "naranja"];
```

Cada elemento en un array tiene una posición o índice que comienza en 0. Podemos acceder a los elementos de un array utilizando su índice. Por ejemplo, para acceder al primer elemento de un array, utilizamos 'array[0]', para el segundo elemento usamos 'array[1]', y así sucesivamente.

En este ejemplo, accedemos al segundo elemento del array 'numeros' y lo mostramos en la consola:

```
let numeros = [1, 2, 3, 4, 5];
console.log(numeros[1]); // Output: 2
```

En JavaScript, los índices de los arrays comienzan en 0, por lo que 'numeros[1]' nos da el segundo elemento del array.

Podemos modificar los elementos de un array asignando nuevos valores a través de su índice. Simplemente seleccionamos el elemento deseado y le asignamos un nuevo valor.

Aquí actualizamos el valor del tercer elemento del array 'frutas':

```
let frutas = ["manzana", "banana", "naranja"];
frutas[2] = "pera";
console.log(frutas); // Output: ["manzana", "banana", "pera"]
```

Hemos cambiado el valor del tercer elemento de "naranja" a "pera".

En un array, podemos agregar nuevos elementos al final utilizando el método 'push()'. También podemos eliminar elementos utilizando los métodos 'pop()' para eliminar el último elemento, 'shift()' para eliminar el primer elemento y 'splice()' para eliminar elementos en una posición específica.

En este ejemplo, añadimos un nuevo elemento al final del array 'frutas' utilizando el método 'push()':

```
let frutas = ["manzana", "banana", "naranja"];
frutas.push("uva");
console.log(frutas); // Output: ["manzana", "banana", "naranja", "uva"]
```

Hemos agregado la fruta "uva" al final del array.

También podemos eliminar el último elemento de un array utilizando el método 'pop()'. Por ejemplo:

```
let frutas = ["manzana", "banana", "naranja"];
frutas.pop();
console.log(frutas); // Output: ["manzana", "banana"]
```

Hemos eliminado el último elemento "naranja" del array.

Podemos obtener la longitud de un array, es decir, la cantidad de elementos que contiene, utilizando la propiedad 'length'. Esto nos permite saber cuántos elementos hay en el array y usarlo en nuestras operaciones.

Acá mostramos cómo obtener la longitud de un array utilizando la propiedad 'length':

```
let frutas = ["manzana", "banana", "naranja"];
console.log(frutas.length); // Output: 3
```

El array 'frutas' tiene una longitud de 3, lo que significa que contiene tres elementos.

Podemos recorrer todos los elementos de un array utilizando bucles, como el bucle 'for'. Esto nos permite realizar operaciones con cada elemento del array de manera eficiente.

```
let numeros = [1, 2, 3, 4, 5];
for (let i = 0; i < numeros.length; i++) {
  console.log(numeros[i]);
}
```

En este ejemplo, utilizamos un bucle 'for' para iterar sobre todos los elementos del array 'numeros' y mostrarlos en la consola. El bucle 'for' recorre cada elemento del array 'numeros' utilizando el índice 'i'. Se imprime cada elemento en la consola.

```
let frutas = ["manzana", "banana", "naranja"];
frutas.forEach(function (fruta) {
  console.log(fruta);
});
```

En este ejemplo utilizamos el método 'forEach()' para iterar sobre cada elemento del array 'frutas'. El método 'forEach()' recibe una función como argumento, la cual se ejecuta para cada elemento del array. En este ejemplo, simplemente imprimimos cada fruta en la consola.

```
let numeros = [1, 2, 3, 4, 5];
let suma = 0;
for (let i = 0; i < numeros.length; i++) {
  suma += numeros[i];
}
console.log(suma); // Output: 15
```

En este caso, utilizamos un bucle 'for' para iterar sobre los elementos del array 'numeros' y sumarlos en la variable 'suma'. Al final, imprimimos el resultado de la suma en la consola.