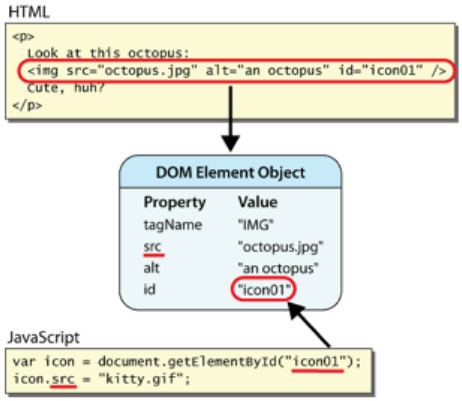
SERVICIO NACIONAL DE APRENDIZAJE

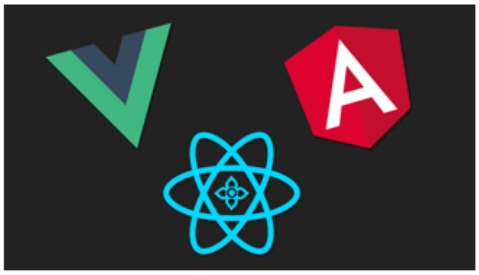
APRENDIZ:	Marco Antonio Mesa Cáceres		
No. DOCUMENTO:	74362247	**FICHA No.**	2670142
OBJETIVO:	**Investigar y desarrollar cuestionario JavaScript**		

3.2.1 Basados en los anteriores video tutoriales y su investigación propia responda a las siguientes preguntas, generando una definición sobre cada una de ellas con sus propias palabras, debe de realizar ejemplos de cada uno de los conceptos:

- ✓ ¿Con sus palabras realice una comparación entre las diferentes versiones de CSS, que ha evolucionado en cada versión?
- ✓ ¿Investigue que es la ECMAScript y Con sus palabras determine su relación con JavaScript, identifique los cambios que han ocurrido durante cada una de sus versiones?
- ✓ ¿Investigue y Compare los distintos Motores de JavaScript (V8, Chakra, SpiderMonkey, Etc)?
- ✓ Identifique y explique el ejemplo que se muestra en la imagen derecha.
- ✓ Realice Ejemplos del uso de los siguientes Temas del Lenguaje Javascript en una carpeta y súbala a un repositorio en Github:
  - Comentarios.
  - Declaraciones (var, let y const).
  - Tipos de Datos (Boolean, Null, Undefined, Number, String, Symbol y Object), Conversion de Datos.



- Literales (Array, Boolean, Integers, Flotantes, Objetos, RegExp, String)
- Sentencias Condicionales (if, else, switch)
- Sentencias de Captura de Errores (throw, try - catch).
- Ciclos e Iteraciones (for, while, do..while, labels para ciclos, break, continue, for..in, for..of)
- Funciones (Como expresiones, llamadas, alcance de variables, Recursividad, Alcance de Variables, Closures, Argumentos, Parametros, Funciones Flecha, Funciones Predefinidas).
- Operadores (Asignacion y Asignacion destructurada, Comparacion, Aritmeticos, Logicos, String, Ternario, Delete, typeof, void, in, instanceOf, presendencia de operadores, this, super, operador de propagacion).
- Colecciones (Array, Metodos Array, Matrices, Maps, Sets).



- Objetos (Declaracion, Propiedades, Funciones de Listado, Constructores, Create, Metodos, Herencia, getters y setters, Comparacion de Objetos).

✓ JavaScript es uno de los lenguajes más revolucionarios y que más crecimiento tiene es por esto que en los últimos años la cantidad de frameworks que se han creado son bastantes. Investigue sobre los frameworks (VueJS, Angular y React) que actualmente se están usando y realice una comparación entre estos.

DESARROLLO:

1. Comparación entre las diferentes versiones de CSS

CSS ha evolucionado significativamente desde su primera versión, introduciendo nuevas características y mejorando las existentes para ofrecer más control sobre el diseño y la presentación de las páginas web.

- **CSS1**: La primera versión de CSS, publicada en 1996, estableció las bases para el estilo de los documentos HTML, incluyendo características básicas como colores, fuentes y alineación de texto.
  - **Ejemplo CSS1**:

css

```
body {
    background-color: white;
    color: black;
    font-family: Arial, sans-serif;
}
```

- **CSS2**: Publicado en 1998, CSS2 introdujo posicionamiento absoluto y relativo, z-index, estilos para impresión, y soporte para medios específicos.
  - **Ejemplo CSS2**:

css

```
.container {
    position: relative;
}

.element {
    position: absolute;
    top: 50px;
    left: 100px;
}
```

```
@media print {
    body {
        font-size: 12pt;
    }
}
```

- **CSS3**: Dividido en módulos, CSS3 trajo muchas mejoras y nuevas características, como transiciones, transformaciones, animaciones, y más control sobre el diseño con flexbox y grid layout.
    - **Ejemplo CSS3**:

      css

```
.box {
    width: 100px;
    height: 100px;
    background-color: blue;
    transition: transform 0.5s;
}

.box:hover {
    transform: rotate(45deg);
}

.grid-container {
    display: grid;
    grid-template-columns: repeat(3, 1fr);
    gap: 10px;
}
```

2. ECMAScript y su relación con JavaScript

ECMAScript es una especificación de lenguajes de scripting mantenida por Ecma International. JavaScript es una implementación de esta especificación. Las versiones de ECMAScript se actualizan periódicamente para introducir nuevas funcionalidades y mejoras al lenguaje.

- **ES3 (1999)**: Añadió expresiones regulares, manejo de excepciones y mejor manejo de cadenas.
- **ES5 (2009)**: Introdujo características como `strict mode`, métodos de array (`forEach`, `map`, `filter`), y mejoras en la manipulación de objetos.
- **ES6 (2015)**: También conocido como ECMAScript 2015, trajo grandes cambios como `let` y `const`, clases, módulos, funciones flecha y promesas.
- **Ejemplo ES6**:

  javascript

```
// Declaraciones con let y const
let variableLet = "Soy un let";
const variableConst = "Soy un const";
```

```
// Función flecha
const suma = (a, b) => a + b;

// Promesa
const promesa = new Promise((resolve, reject) => {
    if (true) {
        resolve("Promesa resuelta");
    } else {
        reject("Promesa rechazada");
    }
});
```

3. Comparación de los distintos motores de JavaScript

Motores de JavaScript son implementaciones que interpretan y ejecutan el código JavaScript. Los principales motores son:

- **V8**: Desarrollado por Google, utilizado en Google Chrome y Node.js. Es conocido por su alta performance y el uso de la compilación Just-In-Time (JIT).
- **Chakra**: Desarrollado por Microsoft para su navegador Edge. También usa JIT y se destaca por su integración con Windows.
- **SpiderMonkey**: El motor de JavaScript de Mozilla, utilizado en Firefox. Es uno de los motores más antiguos y ha sido actualizado para soportar las últimas versiones de ECMAScript.
- **JavaScriptCore**: También conocido como Nitro, es el motor de JavaScript utilizado en Safari de Apple.

4. Explicación del ejemplo en la imagen derecha

La imagen proporciona un ejemplo de manipulación del DOM (Document Object Model) utilizando HTML y JavaScript.

1. **HTML:**
    - En el HTML, se muestra un párrafo `<p>` que contiene un texto y una etiqueta `<img>` que muestra una imagen de un pulpo.
    - La etiqueta `<img>` tiene varios atributos:
        - `src`: especifica la ruta de la imagen "octopus.jpg".
        - `alt`: proporciona un texto alternativo para la imagen, en este caso, "un pulpo".
        - `id`: identifica la imagen con el valor "icon01".
2. **JavaScript:**
    - En el JavaScript, se selecciona la imagen utilizando su ID "icon01" con `document.getElementById("icon01")` y se asigna a la variable `warIcon`.
    - Luego, se cambia la fuente (`src`) de la imagen utilizando `icon.src = "kitten.gif"`, lo que significa que la imagen del pulpo se reemplaza por una imagen de un gatito con la ruta "kitten.gif".

En resumen, este ejemplo ilustra cómo se puede acceder y manipular elementos del DOM utilizando JavaScript, en este caso, cambiando la imagen mostrada en la página web.

5. Ejemplos de uso de temas de JavaScript

Repositorio en GitHub

Se puede crear una carpeta con los ejemplos anteriores y subirla a un repositorio en GitHub siguiendo estos pasos:

1. **Crea una carpeta en tu máquina local**.
2. **Añade los archivos de ejemplos** (`comentarios.js`, `declaraciones.js`, etc.).
3. **Inicializa un repositorio Git**:

   bash

   ```
   git init
   ```

4. **Añade los archivos al repositorio**:

   bash

   ```
   git add .
   ```

5. **Realiza un commit con un mensaje descriptivo**:

   bash

   ```
   git commit -m "Añadir ejemplos de comentarios y declaraciones en JavaScript"
   ```

6. **Crea un nuevo repositorio en GitHub** desde tu cuenta.
7. **Conecta tu repositorio local al repositorio de GitHub**:

   bash

   ```
   git remote add origin https://github.com/tu_usuario/tu_repositorio.git
   ```

8. **Sube los archivos al repositorio de GitHub**:

   bash

   ```
   git push -u origin master
   ```

Comentarios

- **Ejemplo**:

```javascript
// Esto es un comentario de una línea
console.log("Hola"); // Comentario al final de una línea

/*
  Esto es un comentario
  de múltiples líneas
*/
```

Declaraciones (var, let y const)

- **Ejemplo**:

```javascript
var variableVar = "Soy var"; // Declaración con var
let variableLet = "Soy let"; // Declaración con let
const variableConst = "Soy const"; // Declaración con const

// Ejemplo práctico
if (true) {
    var dentroVar = "Disponible fuera del bloque";
    let dentroLet = "Solo disponible dentro del bloque";
    const dentroConst = "Solo disponible dentro del bloque";
}

console.log(dentroVar); // Funciona
// console.log(dentroLet); // Error
// console.log(dentroConst); // Error
```

Tipos de Datos (Boolean, Null, Undefined, Number, String, Symbol y Object), Conversion de Datos.

1. **Boolean**: Representa un valor verdadero o falso.
    - **Ejemplo**:

      ```javascript
      let isJavaScriptFun = true;
      console.log(isJavaScriptFun); // true
      ```

2. **Null**: Representa la ausencia intencional de cualquier valor.
    - **Ejemplo**:

      ```javascript
      let emptyValue = null;
      ```

```
console.log(emptyValue); // null
```

3. **Undefined**: Indica que una variable no ha sido asignada a un valor.
   - **Ejemplo**:

     javascript

     ```
     let notAssigned;
     console.log(notAssigned); // undefined
     ```

4. **Number**: Representa tanto números enteros como de punto flotante.
   - **Ejemplo**:

     javascript

     ```
     let integer = 42;
     let float = 3.14;
     console.log(integer); // 42
     console.log(float); // 3.14
     ```

5. **String**: Representa una cadena de caracteres.
   - **Ejemplo**:

     javascript

     ```
     let greeting = "Hello, World!";
     console.log(greeting); // "Hello, World!"
     ```

6. **Symbol**: Un valor único y no modificable.
   - **Ejemplo**:

     javascript

     ```
     let symbol1 = Symbol("description");
     let symbol2 = Symbol("description");
     console.log(symbol1 === symbol2); // false
     ```

7. **Object**: Colección de propiedades, donde cada propiedad es una asociación entre una clave (string o Symbol) y un valor.
   - **Ejemplo**:

     javascript

     ```
     let person = {
         name: "Alice",
         age: 25
     };
     console.log(person); // { name: 'Alice', age: 25 }
     ```

Conversión de Datos:

- **Ejemplo**:

  javascript

```
let num = "123";
let convertedNumber = Number(num); // Convertir string a número
console.log(convertedNumber); // 123

let booleanValue = Boolean(num); // Convertir string a booleano
console.log(booleanValue); // true

let str = String(456); // Convertir número a string
console.log(str); // "456"
```

Literales

1. **Array**:
   - **Ejemplo**:

     javascript

```
let fruits = ["apple", "banana", "cherry"];
console.log(fruits); // ["apple", "banana", "cherry"]
```

2. **Boolean**:
   - **Ejemplo**:

     javascript

```
let isStudent = false;
console.log(isStudent); // false
```

3. **Integers y Flotantes**:
   - **Ejemplo**:

     javascript

```
let integer = 42;
let float = 3.14159;
console.log(integer); // 42
console.log(float); // 3.14159
```

4. **Objetos**:
   - **Ejemplo**:

     javascript

```
let car = {
    make: "Toyota",
```

```
    model: "Corolla",
    year: 2020
};
console.log(car); // { make: 'Toyota', model: 'Corolla',
year: 2020 }
```

5. **RegExp**:
   - **Ejemplo**:

     javascript

     ```
     let regex = /ab+c/;
     console.log(regex); // /ab+c/
     ```

6. **String**:
   - **Ejemplo**:

     javascript

     ```
     let message = "Hello, World!";
     console.log(message); // "Hello, World!"
     ```

Sentencias Condicionales

7. **if, else**:
   - **Ejemplo**:

     javascript

     ```
     let age = 18;
     if (age >= 18) {
         console.log("You are an adult.");
     } else {
         console.log("You are a minor.");
     }
     ```

8. **switch**:
   - **Ejemplo**:

     javascript

     ```
     let fruit = "apple";
     switch (fruit) {
         case "banana":
             console.log("Banana is yellow.");
             break;
         case "apple":
             console.log("Apple is red.");
             break;
         default:
             console.log("Unknown fruit.");
     }
     ```

Sentencias de Captura de Errores

1. **throw, try - catch**:
    - **Ejemplo**:

      javascript

      ```
      try {
          let result = riskyOperation();
      } catch (error) {
          console.log("An error occurred: " + error.message);
      }

      function riskyOperation() {
          throw new Error("Something went wrong!");
      }
      ```

Ciclos e Iteraciones

1. **for**:
    - **Ejemplo**:

      javascript

      ```
      for (let i = 0; i < 5; i++) {
          console.log(i);
      }
      ```

2. **while**:
    - **Ejemplo**:

      javascript

      ```
      let count = 0;
      while (count < 5) {
          console.log(count);
          count++;
      }
      ```

3. **do..while**:
    - **Ejemplo**:

      ```
      javascript
      Copy code
      let i = 0;
      do {
          console.log(i);
          i++;
      } while (i < 5);
      ```

4. **labels para ciclos, break, continue**:

```
javascript
Copy code
outerLoop: for (let i = 0; i < 3; i++) {
    for (let j = 0; j < 3; j++) {
        if (i === 1 && j === 1) {
            break outerLoop;
        }
        console.log(i, j);
    }
}
```

5. **for..in**:

```
javascript
Copy code
let obj = { a: 1, b: 2, c: 3 };
for (let key in obj) {
    console.log(key, obj[key]);
}
```

6. **for..of**:

```
javascript
Copy code
let iterable = [10, 20, 30];
for (let value of iterable) {
    console.log(value);
}
```

Funciones

1. **Como expresiones**:

```
javascript
Copy code
let add = function(a, b) {
    return a + b;
};
```

2. **Llamadas**:

```
javascript
Copy code
function greet(name) {
    console.log("Hello, " + name);
}
greet("Alice");
```

3. **Alcance de variables**:

```
javascript
Copy code
let globalVar = "global";

function scopeTest() {
    let localVar = "local";
    console.log(globalVar); // global
    console.log(localVar); // local
}

scopeTest();
// console.log(localVar); // Error
```

4. **Recursividad**:

```
javascript
Copy code
function factorial(n) {
    if (n === 0) {
        return 1;
    }
    return n * factorial(n - 1);
}
console.log(factorial(5)); // 120
```

5. **Closures**:

```
javascript
Copy code
function makeCounter() {
    let count = 0;
    return function() {
        count++;
        return count;
    };
}

let counter = makeCounter();
console.log(counter()); // 1
console.log(counter()); // 2
```

6. **Argumentos y Parámetros**:

```
javascript
Copy code
function sum(a, b) {
    return a + b;
}
console.log(sum(2, 3)); // 5
```

7. **Funciones Flecha**:

```
javascript
Copy code
let add = (a, b) => a + b;
console.log(add(5, 3)); // 8
```

8. **Funciones Predefinidas**:

```
javascript
Copy code
let str = "Hello, world!";
console.log(str.toUpperCase()); // "HELLO, WORLD!"
```

Operadores

1. **Asignación y Asignación destructurada**:

```
javascript
Copy code
let x = 10;
let [a, b] = [1, 2];
```

2. **Comparación**:

```
javascript
Copy code
console.log(5 == "5"); // true
console.log(5 === "5"); // false
```

3. **Aritméticos**:

```
javascript
Copy code
let sum = 10 + 5;
```

4. **Lógicos**:

```
javascript
Copy code
let result = true && false;
```

5. **String**:

```
javascript
Copy code
let greeting = "Hello, " + "world!";
```

6. **Ternario**:

```
javascript
Copy code
let age = 18;
```

```
let canVote = (age >= 18) ? "Yes" : "No";
```

7. **Delete**:

```
javascript
Copy code
let obj = { a: 1, b: 2 };
delete obj.a;
```

8. **typeof**:

```
javascript
Copy code
console.log(typeof 123); // "number"
```

9. **void**:

```
javascript
Copy code
void function() {
    console.log("This will run without returning a value");
}();
```

10. **in**:

```
javascript
Copy code
let obj = { a: 1, b: 2 };
console.log("a" in obj); // true
```

11. **instanceof**:

```
javascript
Copy code
console.log([] instanceof Array); // true
```

12. **Precedencia de operadores**:

```
javascript
Copy code
let result = 2 + 3 * 4; // 14
```

13. **this**:

```
javascript
Copy code
let person = {
    name: "Alice",
    greet: function() {
        console.log(this.name);
    }
};
```

```
person.greet(); // "Alice"
```

14. **super**:

```
javascript
Copy code
class Parent {
    constructor() {
        this.name = "Parent";
    }
    greet() {
        console.log("Hello from " + this.name);
    }
}

class Child extends Parent {
    constructor() {
        super();
        this.name = "Child";
    }
    greet() {
        super.greet();
        console.log("Hello from " + this.name);
    }
}

let child = new Child();
child.greet();
```

15. Operador de Propagación

- **Operador de Propagación** (. . .): Permite expandir elementos de un iterable.

```
javascript
Copy code
let array = [1, 2, 3];
let newArray = [...array, 4, 5];
console.log(newArray); // [1, 2, 3, 4, 5]

let obj1 = { a: 1, b: 2 };
let obj2 = { ...obj1, c: 3 };
console.log(obj2); // { a: 1, b: 2, c: 3 }
```

Colecciones en JavaScript

1. Array

Un **Array** es una colección ordenada de elementos. Los arrays pueden almacenar elementos de cualquier tipo de dato y se acceden por su índice.

- **Crear un array**:

```
javascript
Copy code
let array = [1, 2, 3, 4, 5];
console.log(array); // [1, 2, 3, 4, 5]
```

- **Acceder a un elemento del array**:

```
javascript
Copy code
console.log(array[0]); // 1
```

- **Modificar un elemento del array**:

```
javascript
Copy code
array[1] = 10;
console.log(array); // [1, 10, 3, 4, 5]
```

2. Métodos Array

Los **métodos de array** son funciones que permiten manipular y trabajar con arrays de manera eficiente.

- **push()**: Añade uno o más elementos al final de un array.

```
javascript
Copy code
array.push(6);
console.log(array); // [1, 10, 3, 4, 5, 6]
```

- **pop()**: Elimina el último elemento de un array.

```
javascript
Copy code
array.pop();
console.log(array); // [1, 10, 3, 4, 5]
```

- **shift()**: Elimina el primer elemento de un array.

```
javascript
Copy code
array.shift();
console.log(array); // [10, 3, 4, 5]
```

- **unshift()**: Añade uno o más elementos al inicio de un array.

```
javascript
Copy code
array.unshift(0);
```

```
console.log(array); // [0, 10, 3, 4, 5]
```

- **forEach()**: Ejecuta una función por cada elemento del array.

```
javascript
Copy code
array.forEach(element => {
  console.log(element); // Imprime cada elemento del array
});
```

- **map()**: Crea un nuevo array con los resultados de la función aplicada a cada elemento.

```
javascript
Copy code
let doubled = array.map(element => element * 2);
console.log(doubled); // [0, 20, 6, 8, 10]
```

- **filter()**: Crea un nuevo array con los elementos que pasan una condición.

```
javascript
Copy code
let even = array.filter(element => element % 2 === 0);
console.log(even); // [0, 10, 4]
```

- **reduce()**: Aplica una función a un acumulador y a cada elemento del array (de izquierda a derecha) para reducirlo a un único valor.

```
javascript
Copy code
let sum = array.reduce((accumulator, currentValue) => accumulator +
currentValue, 0);
console.log(sum); // 22
```

3. Matrices

Una **matriz** es un array de arrays. Se utiliza para representar datos en múltiples dimensiones.

- **Crear una matriz**:

```
javascript
Copy code
let matrix = [
  [1, 2, 3],
  [4, 5, 6],
  [7, 8, 9]
];
console.log(matrix);
```

- **Acceder a un elemento de la matriz**:

```
javascript
Copy code
console.log(matrix[1][2]); // 6 (segunda fila, tercer columna)
```

4. Maps

Un **Map** es una colección de pares clave-valor donde las claves pueden ser de cualquier tipo.

- **Crear un Map**:

```
javascript
Copy code
let map = new Map();
```

- **Añadir elementos a un Map**:

```
javascript
Copy code
map.set('name', 'Alice');
map.set('age', 25);
console.log(map);
```

- **Acceder a un valor por su clave**:

```
javascript
Copy code
console.log(map.get('name')); // 'Alice'
```

- **Verificar si una clave existe en el Map**:

```
javascript
Copy code
console.log(map.has('age')); // true
```

- **Eliminar un elemento por su clave**:

```
javascript
Copy code
map.delete('age');
console.log(map.has('age')); // false
```

- **Iterar sobre un Map**:

```
javascript
Copy code
map.forEach((value, key) => {
  console.log(`${key}: ${value}`);
});
```

5. Sets

Un **Set** es una colección de valores únicos.

- **Crear un Set**:

```
javascript
Copy code
let set = new Set();
```

- **Añadir elementos a un Set**:

```
javascript
Copy code
set.add(1);
set.add(2);
set.add(2); // No se añade, ya que el valor 2 ya está en el Set
console.log(set); // Set { 1, 2 }
```

- **Verificar si un valor existe en el Set**:

```
javascript
Copy code
console.log(set.has(1)); // true
console.log(set.has(3)); // false
```

- **Eliminar un elemento de un Set**:

```
javascript
Copy code
set.delete(2);
console.log(set); // Set { 1 }
```

- **Iterar sobre un Set**:

```
javascript
Copy code
set.add(3);
set.add(4);
set.forEach(value => {
  console.log(value);
});
```

Objetos en JavaScript

1. Declaración

Un **objeto** en JavaScript es una colección de propiedades, donde cada propiedad es una asociación de una clave (o nombre) con un valor.

- **Declarar un objeto**:

```
javascript
Copy code
let person = {
  name: "Alice",
  age: 25,
  city: "New York"
};
console.log(person);
```

2. Propiedades

Las **propiedades** de un objeto son pares clave-valor.

- **Acceder a propiedades**:

```
javascript
Copy code
console.log(person.name); // "Alice"
console.log(person['age']); // 25
```

- **Modificar propiedades**:

```
javascript
Copy code
person.age = 26;
console.log(person.age); // 26
```

- **Añadir nuevas propiedades**:

```
javascript
Copy code
person.country = "USA";
console.log(person.country); // "USA"
```

3. Funciones de Listado

- **Listar todas las propiedades de un objeto**:

```
javascript
Copy code
console.log(Object.keys(person)); // ["name", "age", "city",
"country"]
console.log(Object.values(person)); // ["Alice", 26, "New York",
"USA"]
console.log(Object.entries(person)); // [["name", "Alice"], ["age",
26], ["city", "New York"], ["country", "USA"]]
```

4. Constructores

Un **constructor** es una función utilizada para crear y inicializar objetos.

- **Definir un constructor**:

```
javascript
Copy code
function Person(name, age, city) {
  this.name = name;
  this.age = age;
  this.city = city;
}

let person1 = new Person("Bob", 30, "Los Angeles");
console.log(person1);
```

5. `Object.create`

El método `Object.create` crea un nuevo objeto utilizando un objeto existente como prototipo del nuevo objeto.

- **Utilizar `Object.create`**:

```
javascript
Copy code
let personPrototype = {
  greet: function() {
    console.log(`Hello, my name is ${this.name}`);
  }
};

let person2 = Object.create(personPrototype);
person2.name = "Charlie";
person2.greet(); // "Hello, my name is Charlie"
```

6. Métodos

Los **métodos** son funciones asociadas a un objeto.

- **Definir métodos**:

```
javascript
Copy code
let person3 = {
  name: "Dave",
  age: 40,
  greet: function() {
    console.log(`Hello, my name is ${this.name}`);
  }
};

person3.greet(); // "Hello, my name is Dave"
```

7. Herencia

La **herencia** permite que un objeto derive propiedades y métodos de otro objeto.

- **Herencia con prototipos**:

```
javascript
Copy code
function Employee(name, age, position) {
  Person.call(this, name, age); // Llamar al constructor de la
clase padre
  this.position = position;
}

Employee.prototype = Object.create(Person.prototype);
Employee.prototype.constructor = Employee;

let employee1 = new Employee("Eve", 35, "Manager");
console.log(employee1);
```

8. Getters y Setters

Los **getters** y **setters** permiten definir métodos que se ejecutan al obtener o establecer una propiedad.

- **Definir getters y setters**:

```
javascript
Copy code
let person4 = {
  firstName: "Frank",
  lastName: "Smith",
  get fullName() {
    return `${this.firstName} ${this.lastName}`;
  },
  set fullName(name) {
    [this.firstName, this.lastName] = name.split(" ");
  }
};

console.log(person4.fullName); // "Frank Smith"
person4.fullName = "George Brown";
console.log(person4.fullName); // "George Brown"
```

9. Comparación de Objetos

La comparación de objetos en JavaScript verifica la referencia y no los valores de las propiedades.

- **Comparar objetos**:

```
javascript
Copy code
let obj1 = { a: 1 };
```

```
let obj2 = { a: 1 };
let obj3 = obj1;

console.log(obj1 == obj2); // false (diferentes referencias)
console.log(obj1 === obj2); // false (diferentes referencias)
console.log(obj1 === obj3); // true (misma referencia)
```

Ejemplo Completo

Aquí tienes un ejemplo completo que muestra la mayoría de estos conceptos juntos:

```
javascript
Copy code
// Constructor
function Person(name, age) {
  this.name = name;
  this.age = age;
}

Person.prototype.greet = function() {
  console.log(`Hello, my name is ${this.name}`);
};

// Crear objeto usando constructor
let person1 = new Person("Alice", 25);
person1.greet(); // "Hello, my name is Alice"

// Object.create
let personPrototype = {
  greet: function() {
    console.log(`Hello, my name is ${this.name}`);
  }
};

let person2 = Object.create(personPrototype);
person2.name = "Bob";
person2.greet(); // "Hello, my name is Bob"

// Getters y Setters
let person3 = {
  firstName: "Charlie",
  lastName: "Brown",
  get fullName() {
    return `${this.firstName} ${this.lastName}`;
  },
  set fullName(name) {
    [this.firstName, this.lastName] = name.split(" ");
  }
};

console.log(person3.fullName); // "Charlie Brown"
person3.fullName = "David Smith";
console.log(person3.fullName); // "David Smith"

// Herencia
```

```
function Employee(name, age, position) {
  Person.call(this, name, age); // Llamar al constructor de la clase
padre
  this.position = position;
}

Employee.prototype = Object.create(Person.prototype);
Employee.prototype.constructor = Employee;

let employee1 = new Employee("Eve", 35, "Manager");
employee1.greet(); // "Hello, my name is Eve"
console.log(employee1.position); // "Manager"

// Comparación de objetos
let obj1 = { a: 1 };
let obj2 = { a: 1 };
let obj3 = obj1;

console.log(obj1 == obj2); // false
console.log(obj1 === obj2); // false
console.log(obj1 === obj3); // true
```

Vue.js

Descripción

- **Vue.js** es un framework progresivo de JavaScript utilizado para construir interfaces de usuario. Se puede integrar fácilmente con otros proyectos y bibliotecas existentes. Fue creado por Evan You y lanzado por primera vez en 2014.

Características

- **Simplicidad y Flexibilidad**: Fácil de aprender e integrar con proyectos existentes.
- **Componentes Reutilizables**: Permite crear componentes reutilizables y mantenibles.
- **Reactivo**: Utiliza un sistema de data binding reactivo.
- **Documentación**: Excelente documentación, lo que facilita su aprendizaje.

Ventajas

- **Curva de Aprendizaje Suave**: Ideal para principiantes.
- **Ligero y Rápido**: Buena performance y peso ligero.
- **Flexibilidad**: Puede ser utilizado como una biblioteca para una parte del proyecto o como un framework completo.

Desventajas

- **Comunidad Menor**: Menor comunidad comparado con React y Angular.
- **Soporte de Empresas**: Menor adopción por grandes empresas en comparación con React y Angular.

Angular

Descripción

- **Angular** es un framework de JavaScript desarrollado por Google, lanzado originalmente en 2010 como AngularJS y reescrito en 2016 como Angular 2+. Es un framework robusto para construir aplicaciones web de gran escala.

Características

- **TypeScript**: Basado en TypeScript, que añade características de tipado estático y otros beneficios de ES6.
- **Arquitectura Completa**: Proporciona una solución completa con inyección de dependencias, enrutamiento, formularios, validación, etc.
- **Componentes Reutilizables**: Promueve el uso de componentes reutilizables y modulares.
- **MVVM**: Sigue el patrón Model-View-ViewModel.

Ventajas

- **Soporte Corporativo**: Amplio soporte de Google y adopción por grandes empresas.
- **Escalabilidad**: Ideal para aplicaciones grandes y complejas.
- **Herramientas y Librerías**: Amplia gama de herramientas, bibliotecas y CLI para facilitar el desarrollo.

Desventajas

- **Curva de Aprendizaje**: Curva de aprendizaje más empinada debido a su complejidad y uso de TypeScript.
- **Peso**: Aplicaciones más pesadas comparadas con Vue y React.

React

Descripción

- **React** es una biblioteca de JavaScript para construir interfaces de usuario, desarrollada por Facebook y lanzada en 2013. Aunque es técnicamente una biblioteca, a menudo se utiliza como un framework debido a su ecosistema robusto.

Características

- **Componentes Basados en Estado**: Usa componentes basados en estado y props para crear interfaces dinámicas.
- **Virtual DOM**: Implementa un Virtual DOM para mejorar el rendimiento.

- **JSX**: Usa JSX, una extensión de JavaScript que permite escribir código similar a HTML dentro de JavaScript.
- **Unidirectional Data Flow**: Flujo de datos unidireccional, facilitando el seguimiento de cambios de estado.

Ventajas

- **Amplia Adopción**: Gran adopción y soporte comunitario.
- **Ecosistema Rico**: Gran cantidad de bibliotecas y herramientas disponibles.
- **Performance**: Alto rendimiento gracias al Virtual DOM.

Desventajas

- **Configuración Inicial**: Puede requerir configuración adicional y dependencia en herramientas externas.
- **JSX**: JSX puede ser extraño y difícil de adoptar para algunos desarrolladores nuevos.

Comparación

Característica	Vue.js	Angular	React
Desarrollador	Evan You	Google	Facebook
Lanzamiento	2014	2010 (AngularJS), 2016 (Angular 2+)	2013
Arquitectura	MVVM	MVC/MVVM	V en MVC
Lenguaje Base	JavaScript/TypeScript opcional	TypeScript	JavaScript/JSX
Curva de Aprendizaje	Suave	Empinada	Moderada
Tamaño del Framework	Ligero	Pesado	Moderado
Flexibilidad	Alta	Media	Alta
Ecosistema	Moderado	Completo	Robusto
Documentación	Excelente	Buena	Buena
Soporte Corporativo	Menor	Alto	Alto
Popularidad	Alta	Alta	Muy Alta

Conclusión

- **Vue.js** es ideal para desarrolladores que buscan simplicidad y flexibilidad, con una curva de aprendizaje suave y excelente documentación. Es una gran opción para proyectos pequeños a medianos.

- **Angular** es adecuado para proyectos grandes y complejos que requieren una solución robusta y completa. Es respaldado por Google y tiene una curva de aprendizaje más empinada debido a su complejidad y uso de TypeScript.
- **React** es una biblioteca muy popular respaldada por Facebook, ideal para construir interfaces de usuario dinámicas y eficientes. Tiene un ecosistema robusto y es flexible, aunque puede requerir configuraciones adicionales.

3.2.2 Ver los video tutoriales del curso ("**Introducción a JavaScript y jQuery**") dados por el instructor y realizar los ejemplos y ejercicios realizados en los videos.

Introducción a JavaScript

Ejemplo 1: Hola Mundo

```
html

<!DOCTYPE html>
<html>
<head>
    <title>Ejemplo de Hola Mundo en JavaScript</title>
</head>
<body>

<script>
    // Mostrar un mensaje "Hola Mundo" en la consola del navegador
    console.log("Hola Mundo");
</script>

</body>
</html>
```

Ejemplo 2: Variables y Operaciones

```
html

<!DOCTYPE html>
<html>
<head>
    <title>Ejemplo de Variables y Operaciones en JavaScript</title>
</head>
```

```
<body>

<script>
    // Declarar variables
    let x = 5;
    let y = 3;

    // Realizar operaciones
    let suma = x + y;
    let resta = x - y;
    let multiplicacion = x * y;
    let division = x / y;

    // Mostrar resultados en la consola del navegador
    console.log("Suma:", suma);
    console.log("Resta:", resta);
    console.log("Multiplicación:", multiplicacion);
    console.log("División:", division);
</script>

</body>
</html>
```

Ejemplo 3: Manipulación del DOM

```
html

<!DOCTYPE html>
<html>
<head>
    <title>Ejemplo de Manipulación del DOM en JavaScript</title>
</head>
<body>

<button id="btn">Haz clic</button>

<script>
    // Obtener el botón
    let boton = document.getElementById("btn");

    // Agregar un evento de clic al botón
    boton.addEventListener("click", function() {
        // Cambiar el texto del botón cuando se hace clic
        boton.textContent = "¡Haz clic de nuevo!";
    });
</script>

</body>
</html>
```

Introducción a jQuery

Ejemplo 1: Uso básico de jQuery

```
html

<!DOCTYPE html>
<html>
<head>
    <title>Ejemplo de Uso básico de jQuery</title>
    <!-- Incluir jQuery desde un CDN -->
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></
script>
</head>
<body>

<button id="btn">Haz clic</button>

<script>
    // Esperar a que el documento esté listo
    $(document).ready(function() {
        // Agregar un evento de clic al botón usando jQuery
        $("#btn").click(function() {
            // Cambiar el texto del botón cuando se hace clic
            $(this).text("¡Haz clic de nuevo!");
        });
    });
</script>

</body>
</html>
```

Ejemplo 2: Animación con jQuery

```
html

<!DOCTYPE html>
<html>
<head>
    <title>Ejemplo de Animación con jQuery</title>
    <style>
        #caja {
            width: 100px;
            height: 100px;
            background-color: red;
        }
    </style>
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></
script>
</head>
<body>

<div id="caja"></div>

<script>
    // Esperar a que el documento esté listo
    $(document).ready(function() {
        // Animar la caja cuando se hace clic en ella
```

```
        $("#caja").click(function() {
            $(this).animate({width: "200px", height: "200px"}, 1000);
        });
    });
</script>

</body>
</html>
```

Ejercicio Práctico: Calculadora Simple con JavaScript

html

```
<!DOCTYPE html>
<html>
<head>
    <title>Calculadora Simple</title>
</head>
<body>

<h2>Calculadora Simple</h2>

<input type="number" id="num1">
<select id="operacion">
    <option value="+">+</option>
    <option value="-">-</option>
    <option value="*">*</option>
    <option value="/">/</option>
</select>
<input type="number" id="num2">
<button onclick="calcular()">Calcular</button>

<p id="resultado"></p>

<script>
    function calcular() {
        let num1 = parseFloat(document.getElementById("num1").value);
        let num2 = parseFloat(document.getElementById("num2").value);
        let operacion = document.getElementById("operacion").value;
        let resultado;

        switch (operacion) {
            case "+":
                resultado = num1 + num2;
                break;
            case "-":
                resultado = num1 - num2;
                break;
            case "*":
                resultado = num1 * num2;
                break;
            case "/":
                resultado = num1 / num2;
                break;
            default:
                resultado = "Operación no válida";
```

```
        }

        document.getElementById("resultado").textContent = "Resultado: "
+ resultado;
    }
</script>

</body>
</html>
```