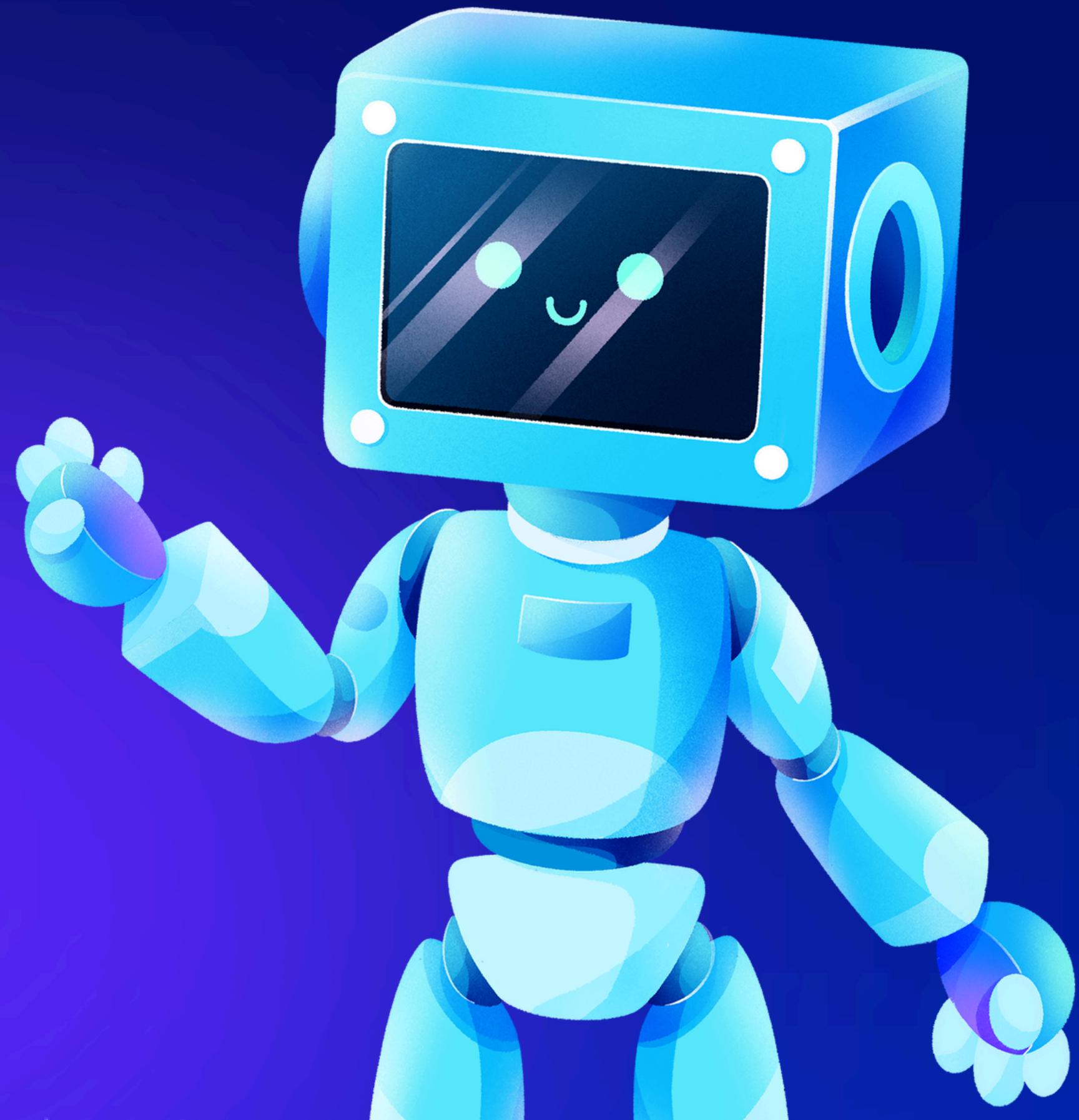


# REPASO DE PROGRAMACIÓN ORIENTADA A OBJETOS Y ASÍNCRONÍA EN JAVASCRIPT





# TABLA DE CONTENIDO.

- Repaso de POO en JavaScript 01
- Introducción a la Asincronía en JavaScript 02
- Ejercicio Integrado 03
- Comida 04



# REPASO DE POO EN JAVASCRIPT



## Clases y Objetos:

- Definición: Las clases son plantillas para crear objetos. Los objetos son instancias de clases que contienen propiedades y métodos.

# SISTEMA DE GESTIÓN DE BIBLIOTECA

```
class Libro {  
    constructor(titulo, autor, anio, disponible = true) {  
        this.titulo = titulo;  
        this.autor = autor;  
        this.anio = anio;  
        this.disponible = disponible;  
    }  
  
    prestar() {  
        if (this.disponible) {  
            this.disponible = false;  
            console.log(`El libro "${this.titulo}" ha sido prestado.`);  
        } else {  
            console.log(`El libro "${this.titulo}" no está disponible.`);  
        }  
    }  
  
    devolver() {  
        this.disponible = true;  
        console.log(`El libro "${this.titulo}" ha sido devuelto.`);  
    }  
  
    class Autor {  
        constructor(nombre, nacionalidad) {  
            this.nombre = nombre;  
            this.nacionalidad = nacionalidad;  
        }  
  
        publicar(libro) {  
            console.log(`El autor ${this.nombre} ha publicado el libro "${libro.titulo}" en el año ${libro.anio}.`);  
        }  
    }  
  
    const autor1 = new Autor('Gabriel García Márquez', 'Colombiano');  
    const libro1 = new Libro('Cien años de soledad', autor1, 1967);  
  
    autor1.publicar(libro1);  
    libro1.prestar(); // "El libro "Cien años de soledad" ha sido prestado."  
    libro1.devolver(); // "El libro "Cien años de soledad" ha sido devuelto."  
}
```

Imagina que estás construyendo un sistema de gestión de una biblioteca. Necesitas representar libros, autores y miembros de la biblioteca.



# HERENCIA:

Definición: Permite que una clase derive de otra, heredando sus propiedades y métodos.

Ejemplo: Sistema de Empleados  
Supongamos que tienes un sistema de recursos humanos que necesita manejar diferentes tipos de empleados, como empleados regulares y gerentes.



```
class Empleado {  
    constructor(nombre, salario) {  
        this.nombre = nombre;  
        this.salario = salario;  
    }  
  
    trabajar() {  
        console.log(`${this.nombre} está trabajando.`);  
    }  
  
    calcularSalarioAnual() {  
        return this.salario * 12;  
    }  
}  
  
class Gerente extends Empleado {  
    constructor(nombre, salario, departamento) {  
        super(nombre, salario);  
        this.departamento = departamento;  
    }  
  
    asignarTarea(tarea) {  
        console.log(`${this.nombre}, gerente del departamento  
de ${this.departamento}, ha asignado la tarea: ${tarea}.`);  
    }  
  
    calcularSalarioAnual() {  
        const bono = 10000;  
        return super.calcularSalarioAnual() + bono;  
    }  
}
```

```
const empleado1 = new Empleado('Ana Pérez', 3000);  
empleado1.trabajar();  
console.log(`Salario anual:  
${empleado1.calcularSalarioAnual()}`);  
  
const gerente1 = new Gerente('Carlos López', 5000,  
'Ventas');  
gerente1.trabajar();  
gerente1.asignarTarea('Incrementar las ventas en un  
10%');  
console.log(`Salario anual:  
${gerente1.calcularSalarioAnual()}`);
```



# MÉTODOS ESTÁTICOS:

Definición: Los métodos estáticos pertenecen a la clase y no a sus instancias.

Ejemplo: Conversor de Unidades

Vamos a crear una clase que actúe como un conversor de unidades de medida (por ejemplo, de kilómetros a millas y viceversa).



```
class ConversorUnidades {  
    static kilometrosAMillas(km) {  
        return km * 0.621371;  
    }  
  
    static millasAKilometros(millas) {  
        return millas / 0.621371;  
    }  
  
    static metrosAPies(metros) {  
        return metros * 3.28084;  
    }  
  
    static piesAMetros(pies) {  
        return pies / 3.28084;  
    }  
}  
  
const km = 5;  
const millas =  
    ConversorUnidades.kilometrosAMillas(km);  
    console.log(`\${km} kilómetros son \${millas.toFixed(2)}  
    millas.`);  
  
const pies = 10;  
const metros = ConversorUnidades.piesAMetros(pies);  
    console.log(`\${pies} pies son \${metros.toFixed(2)}  
    metros.`);
```

# INTRODUCCIÓN A LA ASÍNCRONÍA EN JAVASCRIPT

Concepto de Asincronía: JavaScript es un lenguaje de programación de un solo hilo, pero permite manejar operaciones asíncronas (como peticiones HTTP, temporizadores, etc.) mediante el uso de callbacks, promesas, y `async/await`.

# SETTIMEOUT Y SETINTERVAL

```
// Ejemplo con setTimeout
setTimeout(() => {
  console.log('Este mensaje aparece después de 2 segundos.');
}, 2000);
```

```
// Ejemplo con setInterval
let contador = 0;
const intervalo = setInterval(() => {
  contador += 1;
  console.log(`Contador: ${contador}`);
  if (contador === 5) {
    clearInterval(intervalo); // Detiene el intervalo
  }
}, 1000);
```

- **setTimeout:** Ejecuta una función después de un tiempo especificado.
- **setInterval:** Ejecuta una función repetidamente en intervalos de tiempo especificados.

# CALLBACKS

```
function obtenerDatos(callback) {  
    setTimeout(() => {  
        const datos = { nombre: 'Ana', edad: 28 };  
        callback(datos);  
    }, 2000);  
}
```

```
obtenerDatos((datos) => {  
    console.log(datos); // { nombre: 'Ana', edad: 28 }  
});
```

Definición: Una función que se pasa como argumento a otra función y se ejecuta después de que la operación asíncrona se complete.

# PROMESAS

```
function obtenerDatos() {  
    return new Promise((resolve, reject) => {  
        setTimeout(() => {  
            const datos = { nombre: 'Ana', edad: 28 };  
            resolve(datos);  
        }, 2000);  
    });  
}  
  
obtenerDatos()  
.then((datos) => {  
    console.log(datos); // { nombre: 'Ana', edad: 28 }  
})  
.catch((error) => {  
    console.error(error);  
});
```

Definición: Un objeto que representa la eventual finalización (o falla) de una operación asíncrona y su valor resultante.

# ASYNC/AWAIT

```
async function mostrarDatos() {  
    try {  
        const datos = await obtenerDatos();  
        console.log(datos); // { nombre: 'Ana', edad: 28 }  
    } catch (error) {  
        console.error(error);  
    }  
}  
  
mostrarDatos();
```

Definición: Una sintaxis más sencilla para trabajar con promesas, donde await se utiliza para esperar el resultado de una promesa dentro de una función async.

PROYECTO

: }





Para reforzar los conceptos de Programación Orientada a Objetos (POO) y Asincronismo en JavaScript, van a realizar proyecto sencillo que involucre la creación de una aplicación web de gestión de tareas (To-Do List). Este proyecto permitirá a los estudiantes aplicar los principios de POO, manejar asincronismo para simular la persistencia de datos, y utilizar HTML y CSS para crear una interfaz atractiva.

Proyecto: To-Do List con POO y Asincronismo  
Objetivos

1. Aplicar POO: Crear clases para representar tareas y la lista de tareas.
2. Manejar Asincronismo: Simular la persistencia de datos usando localStorage y fetch para obtener datos ficticios de un API.
3. Diseñar una Interfaz: Usar HTML y CSS para crear una interfaz de usuario intuitiva y atractiva.

# ESPECIFICACIONES DEL PROYECTO

## Clases en JavaScript

- Clase Tarea (Task)
  - Propiedades:
    - id: Identificador único de la tarea.
    - nombre: Nombre de la tarea.
    - completada: Estado de la tarea (true/false).
  - Métodos:
    - completarTarea(): Marca la tarea como completada.
    - editarTarea(nombre): Permite cambiar el nombre de la tarea.
- Clase Lista de Tareas (TaskList)
  - Propiedades:
    - tareas: Array que almacena las instancias de Task.
  - Métodos:
    - agregarTarea(tarea): Añade una nueva tarea a la lista.
    - eliminarTarea(id): Elimina una tarea por su ID.
    - obtenerTareas(): Devuelve la lista de tareas.
    - guardarEnLocalStorage(): Guarda la lista de tareas en localStorage.
    - cargarDeLocalStorage(): Carga las tareas desde localStorage.

# ESPECIFICACIONES DEL PROYECTO

## Asincronismo

- Usar la función fetch para simular la obtención de datos de una API externa (puedes usar jsonplaceholder para obtener una lista de tareas ficticias).
- Implementar una función async que simule el guardado de tareas en un servidor (usando setTimeout para emular el retraso de una solicitud HTTP).

## Interfaz de Usuario (HTML y CSS)

- Crear un formulario sencillo para agregar nuevas tareas.
- Crear una lista para mostrar las tareas, donde cada tarea tenga un botón para marcarla como completada y otro para eliminarla.
- Estilizar la aplicación con CSS para hacerla visualmente atractiva.

# COLORES

## Soft Neutrals

- Fondo: #F5F5F5
- Texto Principal: #333333
- Botones: #4A90E2
- Bordes y Líneas: #E0E0E0
- Resaltado: #50E3C2

## Earthy Tones

- Fondo: #FAF3E0
- Texto Principal: #4A4A4A
- Botones: #8C7B75
- Bordes y Líneas: #D6CFC7
- Resaltado: #FF7F50

Nota:

Fecha maxima de entrega es el 20 de agosto  
Maximo de puntos posibles: 7 puntos.

## Cool Blues

- Fondo: #EAF2F8
- Texto Principal: #34495E
- Botones: #5DADE2
- Bordes y Líneas: #AED6F1
- Resaltado: #2ECC71

## Warm Grays

- Fondo: #F0F0F0
- Texto Principal: #555555
- Botones: #A9A9A9
- Bordes y Líneas: #CCCCCC
- Resaltado: #FFA07A

THANK YOU!



# RESOURCE PAGE

