Paso 1: Preparar el Proyecto

1. **Crear la Estructura del Proyecto:**
    - Usa la misma estructura básica de archivos del proyecto anterior:
        - index.html: Página principal.
        - contacto.html: Página para implementar eventos.
        - style.css: Archivo para los estilos.
        - script.js: Archivo para la lógica en JavaScript.
        - README.md: Documento explicativo del proyecto.
2. **Configurar el Entorno:**
    - Usa **VS Code** como editor.
    - Instala **el servidor visto en clases** para probar las páginas localmente.

Paso 2: Programación Funcional

1. **Función con Currying para el Costo Total:**
    - Implementa una función que calcule el costo total:

Javascript

```
const calcularCosto = precioPorConsulta => numeroDeConsultas =>
precioPorConsulta * numeroDeConsultas;
const costoPaciente = calcularCosto(50000);
console.log(costoPaciente(3)); // Salida: 150000
```

2. **Explicación:**
    - La función calcularCosto retorna otra función que espera el número de consultas. Esto permite calcular el costo total con una sintaxis más limpia.

3. **Función Flecha para Tiempo Promedio de Espera:**

javascript

```
const calcularTiempoPromedio = (tiempos) => tiempos.reduce((a, b) => a + b, 0) / tiempos.length;

console.log(calcularTiempoPromedio([15, 20, 30])); // Salida: 21.67
```

Explicación:

- Se usa reduce para sumar los tiempos y calcular el promedio.

4. **Recursión para Total de Horas de Consulta:**

Javascript

```
const calcularHorasTotales = (horas, index = 0) => {
  if (index === horas.length) return 0;
  return horas[index] + calcularHorasTotales(horas, index + 1);
};
console.log(calcularHorasTotales([3, 4, 5, 2])); // Salida: 14
```

Explicación:

- La función se llama a sí misma para recorrer el arreglo y sumar las horas.

5. **Composición de Funciones para Descuentos:**

Javascript

```
const aplicarDescuento = costo => descuento => costo - (costo * descuento);
const calcularCostoConDescuento = (costo) => aplicarDescuento(costo)(0.1);
console.log(calcularCostoConDescuento(150000)); // Salida: 135000
```

Paso 3: Programación Orientada a Eventos y Asincronía

1. **Capturar Eventos del Usuario:**
    - En contacto.html, crea un formulario:

Html

```
<form id="form-contacto">
  <input type="text" placeholder="Nombre" required>
  <button type="submit">Enviar</button>
</form>
<div id="mensaje-confirmacion"></div>
```

    - En script.js, agrega un listener:

Javascript

```
const form = document.getElementById("form-contacto");
const mensaje = document.getElementById("mensaje-confirmacion");

form.addEventListener("submit", (e) => {
  e.preventDefault();
  mensaje.textContent = "Formulario enviado con éxito.";
});
Evento Personalizado:
javascript
const eventoPaciente = new Event("nuevoPaciente");

document.addEventListener("nuevoPaciente", () => {
  console.log("Un nuevo paciente ha llegado.");
});

document.dispatchEvent(eventoPaciente);
```

2. **Asincronía con Async/Await:**

Javascript

```
async function obtenerDoctores() {
  try {
    const response = await fetch("https://api.ejemplo.com/doctores");
    if (!response.ok) throw new Error("Error al obtener datos");
    const datos = await response.json();
    console.log(datos);
  } catch (error) {
    console.error("Error:", error.message);
  }
}
obtenerDoctores();
```

Paso 4: Programación Orientada a Objetos

1. **Implementar Clase Doctor:**

Javascript

```
class Doctor {
  constructor(nombre, especialidad, experiencia) {
    this.nombre = nombre;
    this.especialidad = especialidad;
    this._experiencia = experiencia;
  }

  get experiencia() {
    return this._experiencia;
  }

  set experiencia(valor) {
    if (valor < 0) throw new Error("La experiencia no puede ser negativa.");
    this._experiencia = valor;
  }

  mostrarInformacion() {
    return `Doctor ${this.nombre}, Especialidad: ${this.especialidad}, Experiencia:
${this.experiencia} años.`;
  }
}
```

2. **Crear Subclase Cirujano:**

```
javascript
class Cirujano extends Doctor {
  constructor(nombre, especialidad, experiencia, operaciones) {
    super(nombre, especialidad, experiencia);
    this.operaciones = operaciones;
  }

  mostrarInformacion() {
    return `Cirujano ${this.nombre}, Especialidad: ${this.especialidad}, Operaciones:
${this.operaciones}.`;
  }
}

const cirujano = new Cirujano("Carlos", "Cardiología", 10, 50);
console.log(cirujano.mostrarInformacion());
```

Paso 5: Documentar el Proyecto

1. **Actualiza el README.md:**
    - Explica:
        - Cómo se implementaron las funciones funcionales.
        - Uso de eventos y asincronía.
        - Detalles de las clases, herencia, encapsulación y polimorfismo.
2. **Incluye Capturas de Pantalla:**
    - Añade imágenes de la ejecución en consola y del sitio web.

Paso 6: Entrega

1. **Verifica el Proyecto:**
    - Prueba todas las funcionalidades en el navegador.
2. **Entrega en el Formato Solicitado:**
    - Sube el proyecto a GitHub o comprímelo en un archivo ZIP.