Memoria: Ejercicio 1 - Calculador de la letra del DNI

Descripción del problema:

El objetivo de este ejercicio es crear un calculador de la letra del DNI basado en la parte numérica que el usuario introduce. La letra del DNI se calcula usando un algoritmo que toma el número, realiza una operación módulo 23 y busca la letra correspondiente en una cadena predefinida.

Proceso de desarrollo:

1. Entrada: El usuario introduce el número del DNI en un campo de texto HTML.
2. Validación: Antes de proceder con el cálculo, validamos que la entrada tenga exactamente 8 dígitos y que sean numéricos.
3. Cálculo de la letra: Utilizamos la operación módulo 23 (numeroDNI % 23) para determinar la letra correspondiente de la cadena de letras.
4. Manejo de errores: Si la entrada no es válida (por ejemplo, si contiene letras o tiene más o menos de 8 dígitos), el programa lanza un error y se lo muestra al usuario.
5. Salida: La letra correspondiente se muestra en la página debajo del botón de cálculo.

Funcionamiento:

- HTML: Se presenta un campo de texto para introducir el número del DNI y un botón que activa la función de cálculo. También tenemos un párrafo (<p>) que muestra el resultado o un mensaje de error.
- JavaScript: La función calcularLetraDNI() se encarga de validar el DNI y calcular la letra correspondiente usando una cadena de caracteres predefinidos.
- Gestión de errores: Si el usuario introduce algo que no sea un número de 8 dígitos, se genera un mensaje de error que se muestra en pantalla.

Memoria: Ejercicio 2 - Presupuesto de un equipo informático

Descripción del problema:

El objetivo de este ejercicio es crear una página web que permita al usuario seleccionar componentes de un equipo informático mediante botones de opción (radio buttons). Dependiendo de las selecciones del usuario, se muestra un presupuesto total actualizado dinámicamente.

Proceso de desarrollo:

Selección de componentes: Creamos varias categorías de componentes (Procesador, Caja, Refrigeración) con opciones de selección.

Cálculo del presupuesto: Cuando el usuario selecciona un componente en cada categoría, se calcula el precio y se actualiza el presupuesto total.

Actualización de la tabla: La información seleccionada por el usuario (como el procesador y su precio) se muestra en una tabla HTML.

Funcionamiento:

- HTML: Definimos las opciones de componentes usando radio buttons y una tabla que mostrará el resumen del presupuesto.
- JavaScript: La función calcularPresupuesto() toma las selecciones del usuario, valida que haya opciones seleccionadas y calcula el total del presupuesto.
- Interactividad: Cada vez que se hace clic en el botón "Calcular Presupuesto", el total se actualiza dinámicamente y se muestra en la tabla.

Memoria: Ejercicio 3 - Gestión de libros (Promesas)

Descripción del problema:

En este ejercicio se debe gestionar un listado de libros usando promesas para simular una operación asíncrona. El objetivo es mostrar un listado de libros con un retraso de 2 segundos.

Proceso de desarrollo:

1. Array de libros: Definimos un array de objetos que representan libros, cada uno con propiedades como title, year, isbn y author.
2. Promesa: Usamos una promesa para simular el retraso en la carga de datos (2 segundos).
3. Manejo de la promesa: Mostramos el listado de libros una vez que la promesa se resuelve exitosamente.

Funcionamiento:

- Promesas: La función mostrarLibrosConRetraso() devuelve una promesa que se resuelve tras 2 segundos. Esto simula una operación de carga de datos asíncrona.
- Manejo de la promesa: Usamos el método .then() para manejar la resolución de la promesa y mostrar el listado de libros en la consola.

Memoria: Ejercicio 4 - Gestión de libros (Async/Await)

Descripción del problema:

Este ejercicio es similar al anterior, pero ahora implementamos la funcionalidad usando async/await, que es una forma más moderna y sencilla de manejar operaciones asíncronas en JavaScript.

Proceso de desarrollo:

1. Array de libros: Como en el ejercicio anterior, usamos un array de objetos que representan libros.
2. Retraso simulado: Usamos una función que devuelve una promesa para simular el retraso de 2 segundos en la carga de los libros.
3. Async/Await: Usamos async/await para manejar la ejecución de la función asíncrona de forma más clara y lineal.

Funcionamiento:

Async/Await: La función mostrarLibrosAsync() utiliza await para esperar la resolución de la promesa que simula la carga de los libros. Una vez resuelta, el código continúa y muestra los libros.

Mejor legibilidad: A diferencia de las promesas con .then(), async/await permite escribir el código de manera más secuencial, facilitando la comprensión

Memoria: Ejercicio 5 - Círculo animado con promesa

Descripción del problema:

El objetivo de este ejercicio es crear una animación que, al hacer clic en un botón, genere un círculo en la pantalla, que aumente progresivamente su tamaño hasta alcanzar una dimensión específica, y finalmente muestre un mensaje dentro del círculo.

La animación del círculo debe estar controlada mediante una promesa para garantizar que todo ocurra de manera asíncrona y en el momento adecuado.

Proceso de desarrollo:

1. Generar el círculo: Al hacer clic en un botón, se crea un elemento div que representa el círculo y se posiciona en el lugar deseado de la página.
2. Animar el círculo: El círculo comienza con un tamaño de 0px y, mediante CSS (propiedad transition), su tamaño aumenta durante un período de tiempo definido.
3. Mostrar el mensaje: Cuando la animación termina (al finalizar la transición del tamaño), se utiliza una promesa para insertar un mensaje dentro del círculo ("Hola, mundo").
4. Manejo de la promesa: La promesa se resuelve una vez que la animación ha terminado, permitiendo que el código continúe y muestre el mensaje dentro del círculo.

Funcionamiento del código:

HTML:

- Un botón que, al hacer clic, ejecuta la función go(), la cual inicia la animación del círculo.

CSS:

- Estilos para el círculo (.circle), que se inicia con un tamaño de 0px y, mediante transition, se anima hasta alcanzar el tamaño final deseado.
- Estilo para el texto dentro del círculo (.message-ball), que lo centra en el círculo.

JavaScript:

- La función showCircle(cx, cy, radius) crea un círculo en la página, y mediante la promesa, gestiona la animación de su tamaño.
- El círculo empieza con un tamaño de 0px, y luego se define su tamaño final en base al radio (radius).
- Usamos el evento transitionend para detectar cuándo ha finalizado la animación. Cuando termina, la promesa se resuelve y se agrega el mensaje dentro del círculo.
- La función go() espera a que la promesa se resuelva (es decir, a que termine la animación), y luego añade el mensaje dentro del círculo con div.append("Hola, mundo!").

Cómo llegamos hasta el código:

1- **Definición del problema:** Se requiere un círculo que crezca con una animación y que, al finalizar dicha animación, muestre un mensaje.

2- Planificación:

- Decidimos utilizar promesas para controlar de manera asíncrona el momento en que termina la animación del círculo.
- La animación del círculo se maneja a través de CSS usando la propiedad transition para el cambio de tamaño.

Desarrollo:

- Creamos un botón en HTML para disparar la acción.
- Definimos estilos en CSS para el círculo y el mensaje.
- Implementamos la lógica de generación y animación del círculo en JavaScript, utilizando una promesa para controlar el flujo y asegurarnos de que el mensaje solo se muestre cuando la animación termine.

Interacción:

Al hacer clic en el botón, el círculo se genera en la posición definida (150, 150), empieza a crecer hasta alcanzar el tamaño deseado (radio = 100px) y finalmente muestra el texto "Hola, mundo!" en su interior.

Todo el proceso está controlado por una promesa que se resuelve cuando la animación del círculo ha terminado, garantizando que el mensaje no aparezca antes de tiempo.