Grado en Ingeniería Informática del Software

Software y estándares para la Web

P7. ECMASCRIPT PURO O “VANILLA”

Escuela de Ingeniería Informática
Universidad de Oviedo

Contenido

Objetivos

En esta práctica se va a realizar:

- Manejo del paradigma de orientación a objetos utilizando clases en el estándar ECMAScript.
- El manejo de colecciones de objetos, excepciones y objetos predefinidos dentro del estándar ECMAScript.
- La realización de pruebas unitarias y pruebas de integración con una clase del estándar ECMAScript
- La generación de código HTML desde el documento de script y su posterior inserción en el documento HTML original.
- La validación del código HTML estático y el código HTML generado

IMPORTANTE: Recuerda las pautas de trabajo establecidas en la primera sesión de prácticas (P0. Pautas de trabajo): valida todos los documentos HTML, valida todas las hojas de estilo CSS, comprueba la adaptabilidad y la accesibilidad con las herramientas proporcionadas.

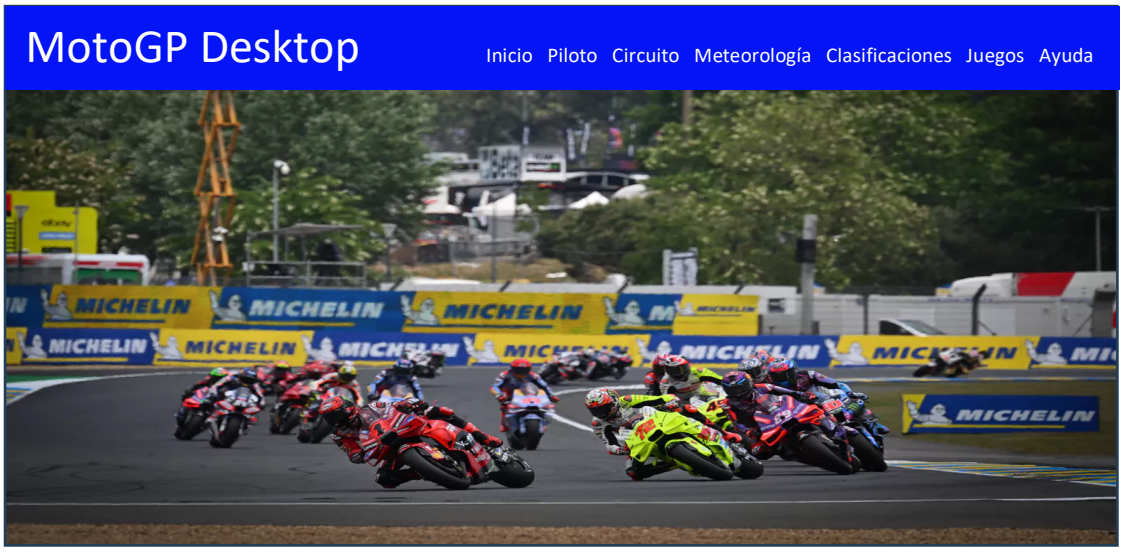
IMPORTANTE: Las palabras en letra naranja negrita son nombres de variables o métodos que se deben respetar a lo largo del desarrollo de las tareas para que se pueda seguir el guion paso a paso sin problemas.

(*) ACLARACIÓN: Los ejemplos utilizados en los ejercicios de ECMAScript contienen código ECMAScript incrustado dentro de los archivos HTML. Esto se realiza con fines didácticos para facilitar la explicación en un único archivo. En los documentos HTML no se debe incrustar código ECMAScript, salvo: referencias a archivos ECMAScript, creación de instancias de las clases e invocación de métodos.

> Todos los ejemplos disponibles se pueden consultar en:
> https://web.dptoinformatica.uniovi.es/cueva/JavaScript/index.html

Temática del proyecto: MotoGP Desktop

Se va a crear MotoGP Desktop para alojar el proyecto de la asignatura. El proyecto es evolutivo y será creado, completado y modificado en las diferentes prácticas de la asignatura.



Ejercicio 1: Implementación de un cronómetro

En este ejercicio se creará una clase Cronómetro que permita establecer el tiempo transcurrido en minutos, segundos y décimas de segundo.

Este ejercicio se considera una prueba unitaria de funcionamiento del cronómetro, que posteriormente se utilizará en otros ejercicios. Por lo tanto, el ejercicio se centra en la funcionalidad del cronómetro y no en la presentación de este.

La funcionalidad del cronómetro a implementar será similar a la que se puede observar en el siguiente enlace:

https://web.dptoinformatica.uniovi.es/cueva/JavaScript/94-EC6-Ejemplo-clase-Cronometro.html**(*)**

NOTA: Se debe utilizar por defecto el objeto predefinido Temporal de ECMAScript para el manejo del tiempo relacionado con el cronómetro. Solamente en caso de que el objeto Temporal no esté disponible se podrá utilizar el objeto predefinido Date de ECMAScript.

Tarea 1. Creación del documento JavaScript

Dentro de la carpeta js del directorio del proyecto MotoGP-Desktop crea un nuevo documento llamado cronometro.js

Tarea 2. Creación de la clase Cronometro

Dentro del documento creado en la tarea anterior crea la clase

El constructor debe tener una instrucción que inicialice el atributo **tiempo** de la clase Cronometro al valor cero.

Tarea 3. Inicialización del cronometro

Añade un método **arrancar** a la clase Cronometro.

Añade a este método la creación del atributo **inicio** de la clase Cronometro. Este atributo marca el momento temporal en el que se inicia el cronómetro.

Guía para resolver la tarea 3

NOTA: El objeto predefinido Temporal se ha añadido al estándar ECMAScript en el año 2025 y no está disponible en todos los navegadores.

El atributo **inicio** tendrá por valor uno de los siguientes:

- Si el objeto Temporal está disponible, **inicio** será una instancia de alguno de los objetos disponibles en **Temporal**
- Si el objeto Temporal no está disponible, **inicio** será un objeto de tipo **Date** cuyo valor sea el momento actual.

Utiliza excepciones (bloque try-catch) para controlar la disponibilidad del objeto Temporal y pasar al uso del objeto predefinido Date en caso de que Temporal no esté disponible en el navegador que se esté usando.

Consulta la información relativa al objeto predefinido **Temporal** en el siguiente enlace:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Temporal

Consulta la información relativa al objeto predefinido **Date** en el siguiente enlace:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date

Para comprender como se pueden manejar las excepciones en ECMAScript puedes consultar el siguiente enlace:

https://web.dptoinformatica.uniovi.es/cueva/JavaScript/66-Try-Catch.html(*)

Tarea 4. Actualización de la información del cronómetro

Añade el método **actualizar** a la clase Cronometro.

Este método será el encargado de actualizar la información del cronómetro en intervalos de una décima de segundo.

Guía para resolver la tarea 4

El método **actualizar** debe obtener el momento temporal actual y calcular el tiempo que ha transcurrido desde el tiempo recogido en el atributo **inicio** de la clase. El resultado de ese cálculo se debe almacenar en el atributo **tiempo** de la clase.

Añade al método **arrancar** de la clase Cronometro una llamada al método **actualizar**, de tal manera que esta se ejecute cada décima de segundo. Asigna esta llamada al atributo **corriendo** de la clase.

Consulta el siguiente enlace para conocer la forma de realizar llamadas que se ejecuten de forma automática conforme a un intervalo de tiempo dado:

https://developer.mozilla.org/en-US/docs/Web/API/Window/setInterval

La función **bind** de JavaScript tiene como principal cometido es asegurar que el valor de **this** se mantenga constante, incluso cuando la función es llamada en un contexto diferente, como en el caso de utilizar un intervalo. Consulta los siguientes enlaces para conocer la forma en la que se debe utilizar **bind** en este tipo de llamadas:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/bind

https://web.dptoinformatica.uniovi.es/cueva/JavaScript/94-EC6-Ejemplo-clase-Cronometro.html**(*)**

Tarea 5. Métodos de utilidad de la clase Cronometro

Añade a la clase Cronometro un método llamado **mostrar** que muestre, en un párrafo y en formato mm:ss.s, la cantidad de tiempo que ha contado el cronómetro en un momento dado en minutos, segundos y décimas de segundo.

Añade a la clase Cronometro un método llamado **parar** que detenga el cronómetro.

Añade a la clase Cronometro un método llamado **reiniciar** que ponga el cronómetro a cero.

Guía para resolver la tarea 5

El valor del atributo **tiempo** de la clase Cronometro es una cantidad temporal en milisegundos.

Debes convertir el valor almacenado en el atributo **tiempo** a una cadena de texto que refleje los minutos (con 2 dígitos), los segundos (con 2 dígitos) y las décimas de segundo (con 1 dígito).

Realiza los cálculos matemáticos correspondientes para descomponer la variable **tiempo** (milisegundos) en minutos, segundos y milisegundos. Convierte el resultado de esos cálculos a un número entero con el método **parseInt()**.

Consulta la información de la clase **String** y sus métodos de utilidad para conocer cómo representar un número añadiendo ceros por delante:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String

Crea una cadena de texto con el formato solicitado en el enunciado de la tarea que represente los minutos, segundos y décimas de segundo obtenidos del cálculo anterior. Utiliza los métodos del objeto **document** para encontrar *el primer párrafo existente dentro del elemento main del documento* e introduce la cadena creada como valor textual de ese párrafo.

El método **parar** debe limpiar el intervalo que está almacenado en el atributo **corriendo** de la clase Cronometro llamando al método **clearInterval**.

El método **reiniciar** debe:

- Limpiar el intervalo que está almacenado en el atributo **corriendo** de la clase Cronometro llamando a **clearInterval**
- Poner el atributo **tiempo** al valor 0
- Invocar al método **mostrar**.

Tarea 6. Creación del documento cronometro.html

Crea el documento cronometro.html en el directorio raíz del proyecto MotoGP-Desktop. Debe tener la estructura general de todos los documentos que componen el proyecto MotoGP-Desktop (header con h1 y nav, migas de navegación), y enlazar las hojas de estilo estilo.css y layout.css

Tarea 7. Interfaz para la clase Cronometro

Crea un interfaz sencillo para realizar las pruebas de funcionamiento de la clase Cronometro. No es necesario añadir nada nuevo a nivel de CSS al proyecto MotoGP-Desktop.

Modifica el documento cronometro.html para enlazar el documento cronometro.js y crea un objeto de la clase Cronometro.

Guía para resolver la tarea 7

Utiliza el elemento <script> para importar el documento cronometro.js como un script externo en la cabecera del documento cronometro.html. Consulta la información sobre este elemento en los siguientes enlaces:

https://html.spec.whatwg.org/multipage/scripting.html#the-script-element

https://developer.mozilla.org/en-US/docs/Web/HTML/Reference/Elements/script

Añade un elemento main al cuerpo del documento cronometro.html e incluye en este elemento los siguientes componentes:

- Un párrafo para mostrar el tiempo que marca el cronómetro, debe contener inicialmente el texto '00:00.0'
- Tres botones: uno para arrancar el cronómetro, otro para pararlo y un tercero para reiniciarlo.

Consulta los enlaces incluidos a continuación con respecto al elemento button de HTML para la representación de los botones. Utiliza la representación por defecto de los botones, no añadas estilos.

https://html.spec.whatwg.org/multipage/form-elements.html#the-button-element

https://web.dptoinformatica.uniovi.es/cueva/JavaScript/13Botones.html(*)

https://web.dptoinformatica.uniovi.es/cueva/JavaScript/114-evento-botones.html(*)

Haz que los botones invoquen a las siguientes funciones del reloj cuando reciban pulsaciones del usuario, utilizando para ello el atributo **onclick**:

- El botón "Arrancar" debe llamar al método **arrancar** de la clase Cronometro
- El botón "Parar" debe llamar al método **parar** de la clase Cronometro
- El botón "Reiniciar" debe llamar al método **reiniciar** de la clase Cronometro

Tarea 8. Actualización del listado de juegos en el documento juegos.html

Actualiza la lista no ordenada de enlaces existente en el documento juegos.html y añade, en segundo lugar, un enlace al documento cronometro.html con el texto "Cronómetro".

Resultado del ejercicio 1

Una vez completado el ejercicio deberían haberse añadido los siguientes documentos al proyecto MotoGP-Desktop:

- Documento cronometro.js en el directorio js del proyecto
- Documento cronometro.html en el directorio raíz del proyecto

Además, se debe haber modificado el contenido del documento juegos.html para incluir para incluir el enlace de acceso al juego del cronometro.

Recuerda introducir la información sobre el documento cronometro.html en el documento de ayuda de la web.

Ejercicio 2: Juego de memoria con cartas – Métodos de utilidad

En este ejercicio se van a crear una serie de métodos de utilidad que se utilizarán durante la utilización del juego de cartas para realizar algunas acciones comunes.

Tarea 1. Crea los atributos de la clase Memoria

Añade a la clase Memoria los siguientes atributos, dentro del constructor:

- **tablero_bloqueado**, que indica si el tablero se encuentra bloqueado a la interacción del usuario. **Valor de inicialización: true.**
- **primera_carta**, que indica cual es la primera carta a la que se ha dado la vuelta en esta interacción. **Valor de inicialización: null.**
- **segunda_carta**, que indica cual es la segunda carta a la que se ha dado la vuelta en esta interacción. **Valor de inicialización: null.**

Guía para resolver la tarea 1

Observa en el siguiente ejemplo como se crean los atributos en el constructor al definir una clase:

https://web.dptoinformatica.uniovi.es/cueva/JavaScript/68-ES6-Ejemplo-uso-de-clases.html(*)

Tarea 2. Barajando las cartas del juego

Añade a la clase Memoria un método llamado **barajarCartas**.

Este método debe obtener el conjunto de cartas del juego de memoria existente en el documento memoria.html y barajarlas, de tal forma que el orden de las cartas en el juego sea siempre aleatorio.

Guía para resolver la tarea 2

Se debe utilizar el objeto predefinido **document** para obtener, en forma de colección de objetos, el conjunto de cartas del juego. Recuerda que en el contenedor donde están las cartas hay otro elemento al que no se debe modificar su posición y asegúrate de que los números aleatorios generados están dentro de las dimensiones de la colección de objetos. Recuerda que las colecciones de objetos empiezan en el índice cero.

Se debe utilizar el objeto predefinido **Math** para obtener números aleatorios que permitan modificar el orden de las cartas en el juego.

Consulta la información sobre el objeto predefinido **document** y a los métodos **querySelector** y **appendChild** en los siguientes enlaces:

https://developer.mozilla.org/es/docs/Web/API/Document

https://developer.mozilla.org/es/docs/Web/API/Document/querySelector

https://developer.mozilla.org/es/docs/Web/API/Node/appendChild

https://developer.mozilla.org/es/docs/Web/API/Document_Object_Model/Selection_and_traversal_on_the_DOM_tree

Consulta la información relacionada con la propiedad **children** para conocer cómo se accede a los elementos que están dentro de otro elemento:

https://developer.mozilla.org/en-US/docs/Web/API/Element/children

Consulta la información relacionada con el objeto predefinido **Math** y sus métodos en el siguiente enlace:

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Math

Tarea 3. Reiniciando el tablero

Añade a la clase Memoria un método llamado **reiniciarAtributos**.

Este método es el encargado de devolver, al valor de inicialización, los atributos creados para la clase Memoria en la tarea 1 de este ejercicio.

Tarea 4. Deshabilitando las cartas que han formado una pareja

Añade a la clase Memoria un método llamado **deshabilitarCartas**.

Este método es el encargado de deshabilitar las interacciones sobre las cartas de memoria que ya han sido emparejadas.

Guía para resolver la tarea 4

Utiliza el estado '**revelada**' para marcar las cartas de memoria que ya forman parte de una pareja, de tal forma que no vuelvan a ponerse boca abajo en ningún momento. Una vez realizado el paso anterior, invoca al método **reiniciarAtributos**.

Modifica la hoja de estilos **cards.css** para hacer que las tarjetas con estado '**revelada**' tengan el mismo estilo que las tarjetas con estado '**volteada**'

Tarea 5. Comprobando el final del juego

Añade a la clase Memoria un método llamado **comprobarJuego**.

Este método debe comprobar si quedan cartas por emparejar en el juego o si ya se han descubierto todas las parejas.

Guía para resolver la tarea 5

Utiliza los métodos del objeto predefinido **document** para acceder a las cartas del juego y saber si todas tienen el estado '**revelada**'. En caso afirmativo, el juego habrá acabado.

Añade una llamada a este método dentro del método **deshabilitarCartas**, antes del reinicio del tablero.

Consulta la información sobre el objeto predefinido **document** y el método **querySelectorAll** en los siguientes enlaces:

https://developer.mozilla.org/es/docs/Web/API/Document

https://developer.mozilla.org/en-US/docs/Web/API/Document/querySelectorAll

Resultado del ejercicio 2

Una vez completado el ejercicio deberían haberse realizado cambios únicamente sobre el fichero memoria.js

Ejercicio 3: Juego de memoria con cartas – Lógica del juego

En este ejercicio se completará la lógica del juego de memoria con cartas, de tal manera que se pueda completar una partida al mismo.

Tarea 1. Creación del método que pone las cartas bocabajo

Añade a la clase Memoria el método **cubrirCartas**.

Este método pone bocabajo las dos últimas cartas descubiertas por el usuario cuando no son iguales.

Guía para resolver la tarea 1

Lo primero que debe hacer el método es bloquear el tablero del juego, para evitar pulsaciones en el mismo mientras se ponen las cartas bocabajo, utilizando el atributo **tablero_bloqueado**.

Una vez bloqueado el tablero se deben hacer dos acciones en el orden especificado a continuación:

1. Quitar el atributo **data-estado** de las tarjetas que estaban volteadas
2. Invocar al método **reiniciarAtributos**

Consulta los atributos de la clase Memoria creados en la tarea 1 del ejercicio anterior para conocer cuáles debes usar a fin de identificar las tarjetas que han sido giradas en las dos últimas pulsaciones del usuario.

De cara a que el usuario pueda comprobar visualmente que las cartas no eran iguales, las dos acciones anteriores deben ejecutarse con un retardo; por ejemplo, 1,5 segundos.

Consulta los siguientes ejemplos en relación con el método **setTimeout** de ECMAScript para conocer cómo se pueden ejecutar acciones con un retardo temporal:

https://developer.mozilla.org/es/docs/Web/API/Window/setTimeout

https://web.dptoinformatica.uniovi.es/cueva/JavaScript/58Reloj.html**(*)**

Ajusta el retardo temporal de la ejecución para que el usuario pueda ver con claridad que las cartas volteadas no son iguales antes de volver a ponerlas bocabajo.

Tarea 2. Comprobando la pareja de cartas volteada

Añade a la clase Memoria un método llamado **comprobarPareja**.

Este método debe comprobar si las cartas que se encuentran en el estado '**volteada**' son iguales o no. Si las tarjetas son iguales, estas se deshabilitarán; si las tarjetas son diferentes, estas se pondrán de nuevo bocabajo.

Guía para resolver la tarea 2

Utiliza los atributos **primera_carta** y **segunda_carta** de la clase Memoria para realizar la comprobación de la pareja. Para comprobar una pareja se debe acceder a la imagen almacenada en la tarjeta y realizar la comprobación sobre uno de los atributos de esta.

Consulta la información relacionada con la propiedad **children** para conocer cómo se accede a los elementos que están dentro de otro:

https://developer.mozilla.org/en-US/docs/Web/API/Element/children

Consulta la información relacionada con el método **getAttribute** para conocer cómo se puede acceder al valor de un atributo:

https://developer.mozilla.org/en-US/docs/Web/API/Element/getAttribute

Utiliza un operador ternario de ECMAScript para, a partir del valor de la comprobación anterior, decidir si se invoca al método que deshabilita las cartas (**deshabilitarCartas**) o al método que las pone bocabajo de nuevo (**cubrirCartas**).

Consulta el siguiente enlace para conocer cómo es el operador ternario en ECMAScript y descubrir cómo se utiliza:

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators/Conditional_operator

Tarea 3. Añadiendo la lógica principal del juego al método voltearCarta

Modifica el método **voltearCarta** creado en la sesión anterior para incluir en él la lógica principal del juego de memoria con cartas.

Guía para resolver la tarea 3

Cuando se pulsa una carta en el juego se deben realizar varias comprobaciones: que la carta pulsada no está deshabilitada, que la carta pulsada no está volteada y que el tablero no está bloqueado.

Si ninguna de las tres comprobaciones anteriores obtuvo un resultado verdadero, se puede entonces voltear la carta que se ha pulsado a través del atributo personalizado **data-estado**.

Una vez volteada la carta, se deben tener en cuenta los siguientes casos:

a) En el caso de que esta fuera la primera carta que se voltea en esta jugada se almacena como tal en la variable **primera_carta** y se retorna la ejecución del método.

b) En el caso de que esta fuera la segunda carta que se voltea en esta jugada, se almacena como tal en la variable **segunda_carta** y se invoca al método **comprobarPareja**.

Recuerda que en la hoja de estilos **cards.css** debe existir una regla para las tarjetas que estén en el estado '**revelada**'. A nivel de estilos, el estado '**revelada**' y el estado '**volteada**' son iguales.

Tarea 4. Inicializando el juego de memoria con cartas

Modifica el constructor de la clase Memoria para inicializar de forma correcta el juego de memoria con cartas.

Guía para resolver la tarea 4

La inicialización del juego de memoria con cartas se compone de los siguientes pasos: inicialización de los atributos, barajado de las cartas y desbloqueo del tablero

Comprueba en base a las indicaciones de la tarea 1 del ejercicio anterior que la inicialización de los atributos es correcta.

Tarea 5. Comprobación del funcionamiento del juego

Accede al documento memoria.html y comprueba que el juego de memoria con cartas funciona correctamente: cuando se destapa una pareja las cartas se quedan bocarriba el resto del juego y si se destapan dos cartas que son diferentes estas se vuelen a tapar automáticamente, transcurrido el tiempo indicado en el retardo.

Resultado del ejercicio 3

Una vez completado el ejercicio deberían haberse realizado cambios únicamente sobre el fichero memoria.js

Ejercicio 4: Juego de memoria con cartas – Integración de la clase Cronómetro

En este ejercicio se añadirá al juego de memoria con cartas un cronómetro que permita conocer el tiemplo empleado por el usuario para completar el juego.

Se debe utilizar la clase Cronometro creada en el Ejercicio 1 de este guion para realizar una prueba de integración de dicha clase en el código del juego de memoria con cartas.

Tarea 1. Instanciando la clase Cronometro en el juego de memoria

Modifica el documento memoria.html para enlazar el documento cronometro.js

Crea un objeto de la clase Cronometro en el constructor de la clase Memoria, debajo de la última instrucción que tenga el constructor.

Guía para resolver la tarea 1

Utiliza el elemento <script> para importar el documento cronometro.js como un script externo en la cabecera del documento memoria.html. Consulta la información sobre este elemento en los siguientes enlaces:

https://html.spec.whatwg.org/multipage/scripting.html#the-script-element

https://developer.mozilla.org/en-US/docs/Web/HTML/Reference/Elements/script

Observa en el siguiente ejemplo cómo se crea la instancia de una clase:

https://web.dptoinformatica.uniovi.es/cueva/JavaScript/68-ES6-Ejemplo-uso-de-clases.html(*)

Tarea 2. Actualización del interfaz del juego de memoria con cartas

Añade un párrafo con el texto ‘00:00.0’ debajo del encabezado de orden 2 que da título al juego de memoria con cartas.

Guía para resolver la tarea 2

Modifica la hoja de estilos **layout-cards.css** para que este párrafo ocupe todas las columnas del grid y su texto salga centrado, igual que el encabezado de orden 2.

Recuerda modificar el método **barajarCartas** para tomar en consideración que el párrafo donde saldrá el cronómetro tampoco se debe barajar.

Tarea 3. Arrancando el cronómetro al iniciar el juego

Modifica el constructor de la clase Memoria para que se invoque al método **arrancar** de la clase Cronometro como última instrucción.

Tarea 4. Parando el cronómetro al finalizar el juego de memoria

Modifica el método **comprobarJuego** de la clase Memoria para que se invoque al método **parar** de la clase Cronometro cuando todas las parejas de cartas hayan sido descubiertas.

Tarea 5. Validación del código estático y dinámico del documento memoria.html

Se debe validar el contenido del documento memoria.html

Guía para resolver la tarea 5

Valida el **código estático** del documento memoria.html usando la opción "Validate by File Upload" del validador de HTML del W3C.

Valida el **código dinámico** del documento memoria.html usando la opción "Direct Input" del validador de HTML del W3C. Obtén el código dinámico utilizando las herramientas de desarrollador del navegador.

Resultado del ejercicio 4

Una vez completado el ejercicio se debe haber modificado el contenido del documento memoria.html para incluir la referencia a la clase Cronometro y el contenido del fichero memoria.js para incluir las llamadas a la clase Cronometro.

Recuerda

Se requiere el uso correcto de los elementos HTML5 establecidos en los ejercicios y se valorará positivamente el uso correcto y adecuado al contexto y funcionalidad, de elementos adicionales HTML5.

Se requiere el uso correcto de las propiedades de los módulos CSS establecidos en los ejercicios y se valorará positivamente el uso correcto y adecuado al contexto, de propiedades y módulos adicionales CSS.

Solamente se permite el paradigma de orientación de objetos y todo debe estar organizado con clases y objetos. Todo el código de las clases ECMAScript creadas para la solución de los ejercicios planteados debe estar recogido en documentos JavaScript separados, nunca incrustados en los documentos HTML. Además, no se permite el uso de ningún tipo de bibliotecas externas y debe usarse ECMAScript puro o "vanilla", salvo en aquellos ejercicios cuyo enunciado indique que se use una biblioteca externa (por ejemplo, jQuery).

Las siguientes condiciones son de obligado cumplimiento en todo el proyecto:

- **TODOS** los documentos HTML que componen el proyecto deben ser HTML5 válidos y sin advertencias utilizando el validador de lenguajes de marcado del W3C.
    - Recuerda que el contenido de los documentos HTML puede cambiar después de la ejecución del código ECMAScript. En ese caso, se debe comprobar la validez del código HTML en todos los diferentes estados por los que pase el documento.
- Se deben utilizar las etiquetas semánticas de HTML5 (section, article, etc.) y no está permitido el uso de bloques anónimos (**div**). En aquellos ejercicios en los que se permita el uso de bloques anónimos (**div**) estará indicado en el enunciado del ejercicio; **fuera de esos ejercicios, el uso de bloques anónimos no está permitido.**
- Se deben utilizar selectores de CSS específicos, el uso de selectores id y class no está permitido. **En aquellos ejercicios en los que se permita el uso de los selectores class o id estará indicado expresamente en el enunciado del ejercicio.**
- **TODAS** las reglas de todas las hojas de estilo deben estar precedidas por un comentario donde se indique la especificidad del (o los) selectores de la regla.
- **TODAS** las hojas de estilo que se utilizan en el sitio web deben ser validas utilizando el validador CSS del W3C
- **TODAS** las hojas de estilo deben tener 0 advertencias.
    - Recuerda seleccionar en "Más opciones" el informe de "Todas las advertencias" dentro de las opciones del validador CSS del W3C.
    - Excepcionalmente se permite las advertencias referidas a la verificación de los colores (color y background-color). **OBLIGATORIAMENTE** se debe indicar mediante un comentario en la regla de la hoja de estilo afectada la herencia de colores garantizando que la advertencia ha sido comprobada, verificada y garantizando que no provoca efectos laterales no deseados.
    - Excepcionalmente se permiten las advertencias referidas a la redefinición de propiedades derivadas del uso de @media-queries. **OBLIGATORIAMENTE** se debe indicar mediante un comentario en las reglas de la hoja de estilo afectada que propiedades se están redefiniendo.

- Se debe garantizar la adaptabilidad y realizar su verificación para todos los documentos que componen el proyecto.
    - Se deben utilizar medidas relativas en las hojas de estilo.
- Se debe garantizar la accesibilidad del proyecto mediante los test de las herramientas de accesibilidad para el nivel AAA de las WCAG 2.0 con 0 errores de modo automático en todos los documentos que lo componen.

El no cumplimiento de las características anteriores derivará en la invalidación del proyecto.