Grado en Ingeniería Informática del Software

# Software y estándares para la Web

## P6. ECMASCRIPT PURO O “VANILLA”

Escuela de Ingeniería Informática
Universidad de Oviedo

# Contenido

# Objetivos

En esta práctica se va a realizar:

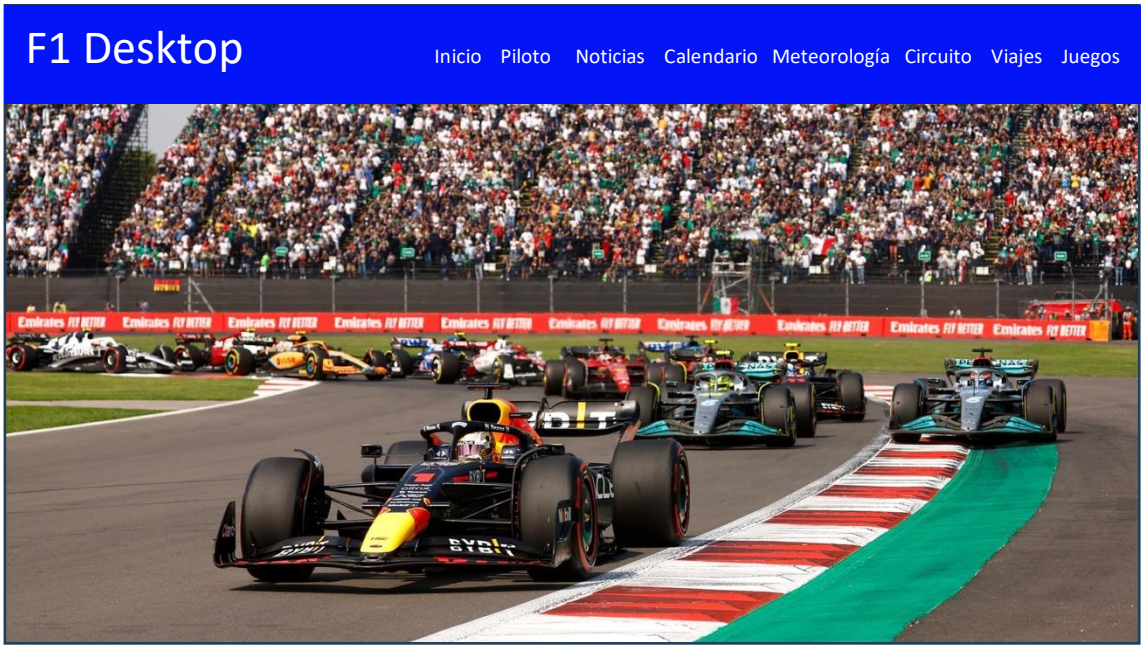
- La creación de un documento de script utilizando el estándar ECMAScript.
- La creación de objetos para la realización de diferentes tareas en combinación con HTML.
- La utilización de objetos predefinidos dentro del estándar ECMAScript.
- El manejo de eventos de ratón dentro del estándar ECMAScript.
- 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.**

# Temática del proyecto: F1 Desktop

El proyecto F1 Desktop es evolutivo y será creado, completado y modificado en las diferentes prácticas de la asignatura.



# Ejercicio 1: Información de un país con HTML y JS

En este ejercicio se va a utilizar un documento JS para componer un fichero HTML5 que contenga información sobre un país del mundo.

El país sobre el que se debe trabajar viene determinado por el circuito de Fórmula 1 con el que se ha trabajado en las sesiones de prácticas 4 y 5 (prácticas de XML). En el anexo I de este documento se encuentra la tabla de circuitos del mundial de F1, el país en el que se encuentran y las ciudades más próximas.

## Tarea 1. Creación del documento JavaScript

Dentro de la carpeta js del directorio del proyecto Escritorio Virtual crea un nuevo fichero llamado pais.js

## Tarea 2. Creación de la clase País

Dentro del fichero creado en la tarea anterior crea la clase País y añade a dicha clase los atributos necesarios para representar la siguiente información: nombre del país, nombre de la capital, nombre del circuito de F1, cantidad de población, tipo de forma de gobierno, coordenadas de la línea de meta del circuito y religión mayoritaria.

### Guía para resolver la tarea 2

Para ver como se crea una clase en ECMAScript puede consultarse el ejercicio:

http://di002.edv.uniovi.es/~cueva/JavaScript/68-ES6-Ejemplo-uso-de-clases.html

## Tarea 3. Creación de métodos en la clase País

Añade métodos a la clase País que permitan: construir un objeto de dicha clase, inicializar los valores de los atributos, acceder a la información de algunos atributos en forma de texto y escribir información en el documento html.

### Guía para resolver la tarea 3

Se deben añadir, al menos, los siguientes métodos:

- Un método constructor que reciba como parámetros los datos de algunos de los atributos creados en el apartado anterior (por ejemplo: nombre del país, nombre de la capital y cantidad de población).
- Un método que rellene el valor del resto de atributos existentes.
- Varios métodos que devuelvan la información principal del país (nombre y capital) en forma de cadena de texto, uno por cada atributo.
- Un método que devuelva la información secundaria del país (nombre del circuito, población, forma de gobierno y religión mayoritaria) con la estructura de una lista de HTML5 dentro de una cadena.
- Un método que escriba en el documento la información de coordenadas de la línea de meta del circuito.

## Tarea 4. Rellenado inicial del documento meteorologia.html

Modifica el documento meteorologia.html para incluir en él una referencia al fichero pais.js y crea un objeto de la clase País con la información del país resultante de aplicar la fórmula explicada al principio del ejercicio.

### Guía para resolver la tarea 4

Utiliza los métodos creados en la tarea anterior para escribir en el documento meteorologia.html toda la información del objeto de la clase País que has creado.

Observa en el siguiente ejemplo cómo se crea la instancia de la clase y como se llama al método correspondiente para escribir la información en el documento html utilizando el método **document.write()**.

http://di002.edv.uniovi.es/~cueva/JavaScript/68-ES6-Ejemplo-uso-de-clases.html

Se aconseja consultar la especificación del método **document.write()** en:

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

Recuerda que algunos métodos devuelven información y otros escriben directamente la información sobre el documento.

Observa el siguiente ejemplo para ver las diferentes formas en las que se puede invocar al método **write** del objeto **document** para añadir información al documento html.

http://di002.edv.uniovi.es/~cueva/JavaScript/74-InfoNavegador.html

Se aconseja consultar la especificación del método **document.querySelector()** en:

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

### Resultado del ejercicio 1

Una vez completado el ejercicio deberían haberse añadido los siguientes ficheros al proyecto del Escritorio Virtual:

- Fichero pais.js en el directorio js del proyecto

Además, se debe haber modificado el contenido de los ficheros meteorologia.html, para incluir la información de la clase Pais, y estilo.css para modificar la presentación de los elementos del fichero anterior (en caso necesario).

# Ejercicio 2: Juego de memoria

En este ejercicio se utilizará JavaScript en combinación con HTML y CSS para programar un juego de memoria.

En este juego habrá un grupo de 12 cartas boca abajo y el usuario deberá ir pulsando en las diferentes cartas para voltearlas y hacer parejas. Para conseguir esta funcionalidad, las cartas pasarán por tres estados diferentes: el estado inicial (boca abajo), el estado "flip" (boca arriba) y el estado "revealed" (boca arriba y forma parte de una pareja ya descubierta).

**NOTA: Para la representación de los estados en este juego se utilizarán los atributos data de HTML5. Este tipo de atributos solo se pueden utilizar en los ejercicios que lo indiquen explícitamente y el incumplimiento de esta restricción invalidará el proyecto.**

## Tarea 1. Creación del menú en juegos.html

Edita el documento juegos.html y añade una sección al inicio del documento que represente el menú de acceso a los diferentes juegos que se van a crear.

Todos los ficheros html en los que se enlace un juego deberán tener esta misma sección de menú en la parte superior, de tal manera que desde un juego se pueda acceder siempre a los demás.

## Tarea 2. Creación del documento memoria.html

Crea el fichero memoria.html en el directorio raíz del proyecto de Escritorio Virtual.

Añade al menú creado en la tarea anterior un enlace que permita el acceso al fichero memoria.html

## Tarea 3. Creación de un nuevo documento JavaScript

Dentro de la carpeta js del directorio del proyecto Escritorio Virtual crea un nuevo fichero llamado memoria.js.

Añade una referencia a este nuevo fichero en el documento memoria.html

## Tarea 4. Creación de la clase Memoria

Dentro del fichero creado en la tarea anterior crea la clase Memoria.

Dentro de dicha clase crea un objeto JSON denominado **elements** que contenga la declaración de los elementos que luego representarán las tarjetas del juego de memoria. Cada tarjeta tendrá un atributo **element** con el nombre del equipo y un valor **source** con la dirección url de una imagen con el logo del equipo.

Para hacer el juego temático con respecto a los contenidos del proyecto Escritorio F1 (F1 Desktop), utilizaremos los siguientes valores para los nombres de los elementos y las imágenes para las tarjetas del juego:

| ELEMENT | SOURCE |
|---|---|
| RedBull | https://upload.wikimedia.org/wikipedia/de/c/c4/Red_Bull_Racing_logo.svg |

| | |
|---|---|
| McLaren | https://upload.wikimedia.org/wikipedia/en/6/66/McLaren_Racing_logo.svg |
| Alpine | https://upload.wikimedia.org/wikipedia/fr/b/b7/Alpine_F1_Team_2021_Logo.svg |
| AstonMartin | https://upload.wikimedia.org/wikipedia/fr/7/72/Aston_Martin_Aramco_Cognizant_F1.svg |
| Ferrari | https://upload.wikimedia.org/wikipedia/de/c/c0/Scuderia_Ferrari_Logo.svg |
| Mercedes | https://upload.wikimedia.org/wikipedia/commons/f/fb/Mercedes_AMG_Petronas_F1_Logo.svg |

## Guía para resolver la tarea 4

Consultar el siguiente enlace para conocer la estructura interior de un objeto JSON así como para comprender como recorrerlo y obtener la información que almacena:

https://developer.mozilla.org/es/docs/Learn/JavaScript/Objects/JSON

Debido a que hay 6 elementos diferentes y el juego constará de 12 cartas, se deben crear dos entradas de cada elemento en el objeto JSON.

## Tarea 5. Creación del constructor de la clase Memoria

Crea un constructor para la clase Memoria que cree los siguientes atributos:

- **hasFlippedCard**, el atributo que indica si ya hay una carta dada la vuelta; **valor de inicialización: false.**
- **lockBoard**, el atributo que indica si el tablero se encuentra bloqueado a la interacción del usuario; **valor de inicialización: false.**
- **firstCard**, el atributo que indica cuál es la primera carta a la que se ha dado la vuelta en esta interacción; **valor de inicialización: null.**
- **secondCard**, el atributo que indica cuál es la segunda carta a la que se ha dado la vuelta en esta interacción; **valor de inicialización: null.**

## Tarea 6. Creación de métodos de utilidad en la clase Memoria

Para que el juego funcione correctamente es necesario crear una serie de métodos de utilidad que realicen ciertas actividades: barajar los elementos del objeto JSON, voltear las tarjetas cuando termina la interacción del usuario, resetear el tablero, comprobar si se ha descubierto una pareja y deshabilitar la interacción sobre las cartas que forman parte de una pareja ya descubierta.

## Guía para resolver la tarea 6

Se deben crear los siguientes métodos:

- Método **shuffleElements**: este método coge el objeto de JSON y baraja los elementos, para que las tarjetas estén en un orden diferente en cada partida del juego. Se puede utilizar cualquier método de ordenación para recorrer y barajar los elementos, como por ejemplo el algoritmo Durstenfeld.

- Método **unflipCards**: este método bloquea el tablero en primer lugar y luego voltea las cartas que estén bocarriba y resetea el tablero.
    - **NOTA**: para que la ejecución del volteo de las tarjetas y el reseteo del tablero se realice con cierto margen temporal después del volteo de la segunda tarjeta se puede meter dentro de un delay utilizando el método setTimeout de ECMAScript.
- Método **resetBoard**: pone a null las variables ***firstCard*** y **secondCard** y pone a false las variables **hasFlippedCard** y **lockBoard**.
- Método **checkForMatch**: comprueba si las cartas volteadas son iguales. Si lo son, llama al método **disableCards** y si no lo son llama al método **unflipCards**.
    - **NOTA**: se puede usar un operador ternario de ECMAScript para simplificar la programación de este método
- Método **disableCards**: este método deshabilita las interacciones sobre las tarjetas de memoria que ya han sido emparejadas. Para ello modifica el valor del atributo **data-state** a **revealed** y después invoca al método **resetBoard**.

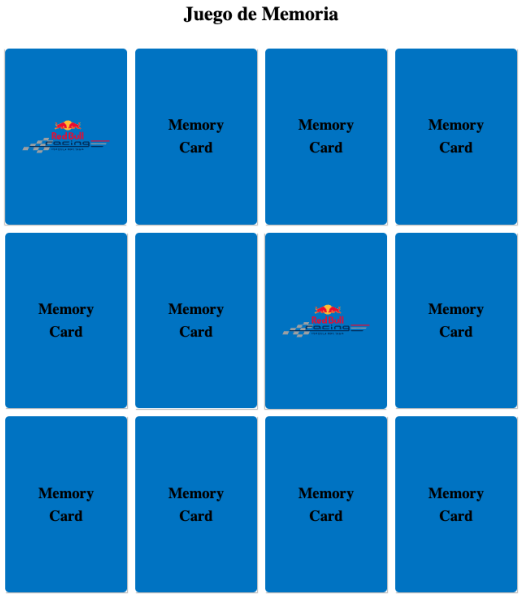
Para consultar las particularidades de cada una de las características de ECMAScript utilizadas en los métodos de utilidad de la clase Memoria se recomienda utilizar los siguientes enlaces:

- Método setTimeout
- Operador ternario

## Tarea 7. Creación de los elementos del juego

Crea, dentro del body del documento memoria.html, una sección que contenga un encabezado con el texto “Juego de Memoria”. En esta sección se mostrarán las tarjetas del juego.

El tablero del juego se debe ver como una sucesión de tarjetas dispuestas de tal manera que haya 4 elementos por fila y existan 3 filas de elementos, tal y como se observa en la siguiente imagen.



**NOTA:** De momento, las tarjetas de memoria del juego no se verán correctamente ya que el encabezado estará tapado por la imagen. Esto es debido a que será necesario añadirles transformaciones y transiciones a nivel de CSS para que la imagen esté oculta inicialmente y la tarjeta "se dé la vuelta" cuando el usuario la seleccione.

## Guía para resolver la tarea 7

Cada tarjeta del juego estará representada por un nodo article dentro de esta sección.

Se debe crear un método llamado **createElements** que recorra el objeto JSON creado en la tarea 3 del ejercicio y cree, por cada elemento existente en el JSON, un nodo article en el documento html para representar cada tarjeta del juego de memoria.

El contenido de cada nodo article será:

- Un atributo "**data-element**" cuyo valor sea igual al valor de la variable **element** extraída del JSON.
- Un encabezado de orden 3 con el texto "Tarjeta de memoria" que se visualizará cuando la tarjeta esté bocabajo (situación inicial).
- Una imagen cuyo atributo **src** apunte a la imagen correspondiente al logo del elemento representado en la tarjeta y cuyo atributo **alt** sea igual al valor de la variable **element** extraída del JSON. Esta imagen solamente se visualizará cuando la tarjeta sea clicada por el usuario y se dé la vuelta.

Para crear los elementos necesarios para el desarrollo del juego así como la sección que los contiene y añadirlos al documento, utiliza los métodos createElement y appendChild del estándar ECMAScript. Puedes consultar la información relativa a estos métodos en los siguientes enlaces:

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

Aplica FLEXBOX a la sección del documento memoria.html que contiene los elementos del juego. Recuerda que la disposición de los elementos debe ser en filas de 4 elementos. Debes hacer la adaptación correspondiente para que el primer elemento de dicha sección (el encabezado con el título del juego) ocupe toda la primera fila por sí mismo.

Referencia: https://www.w3.org/Style/CSS/current-work

Más información:

- https://css-tricks.com/snippets/css/a-guide-to-flexbox/
- https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Flexbox

## Tarea 8. Añadiendo las pulsaciones al juego

Para que las tarjetas de memoria respondan a la interacción del usuario es necesario asociarles el evento clic a través de JavaScript.

Crea un método denominado **addEventListeners** que recorra todas las tarjetas creadas en la tarea anterior y que provoque una llamada al método **flipCard** de la clase Memoria cuando se lance dicho evento.

- **NOTA:** El método **flipCard** se debe invocar utilizando la característica bind de JavaScript de la siguiente manera: **this.flipCard.bind(card, this)**

## Guía para resolver la tarea 8

Consultar el evento global **onclick**:

https://developer.mozilla.org/es/docs/Web/API/GlobalEventHandlers/onclick

Consultar el evento **keydown**, que se produce cuando se pulsa una tecla

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

Para comprender el manejo del evento **onclick** puede consultarse el ejercicio:

http://di002.edv.uniovi.es/~cueva/JavaScript/13Botones.html

Consulta los siguientes enlaces sobre **bind** para comprender el uso del mismo:

http://di002.edv.uniovi.es/~cueva/JavaScript/94-EC6-Ejemplo-clase-Cronometro.html

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

## Tarea 9. Transformación y transición de los elementos del juego

Para que las tarjetas de memoria se vean correctamente en pantalla e interaccionen a las pulsaciones del usuario es necesario incluir fragmentos de código tanto a nivel de CSS como a nivel de JavaScript.

### *Modificaciones a nivel de CSS*

Para que el logo de cada tarjeta de memoria esté oculto y no se vea cuando la tarjeta esté boca abajo es necesario rotar la imagen en el eje Y 180 grados, utilizando el método **rotate** de CSS dentro de la propiedad **transform**. Esto se completará además con las propiedades **backface-visibility** al valor **hidden** tanto en la imagen como en el encabezado de orden 3 existentes dentro de cada tarjeta de memoria **transform-style** al valor **preserve-3d** para que el efecto funcione correctamente.

Cuando el usuario pulse sobre una tarjeta y esta tenga que ser volteada, se utilizará el método **rotate** de CSS para voltear la tarjeta completa 180 grados en el eje Y, dentro de la propiedad **transform**. De esta forma pasará a verse la imagen con el logo y se ocultará el encabezado, dando la sensación de que la tarjeta se da la vuelta.

Para completar todo lo anterior es necesario declarar la propiedad **transition** de CSS en cada artículo al valor "**transform .5s**". Esto hace que cada vez que se ejecute una transformación sobre uno de los artículos esta se ejecute en un espacio de tiempo de medio segundo.

### *Modificaciones a nivel de JavaScript*

Dentro de la clase memoria es necesario crear el método **flipCard** que se encargue de dar la vuelta a las tarjetas cuando estas sean pulsadas por el usuario. Este método recibe como parámetro una variable **game** que representa el juego.

En primer lugar se deben realizar tres comprobaciones:

1. Si la tarjeta pulsada por el usuario ya estaba revelada y formaba parte de una pareja ya descubierta (atributo **data-state** a **revealed**), el método retorna y no hace nada más.
2. Si la propiedad **lockBoard** del juego estaba al valor true, el método retorna y no hace nada más.
3. Si la tarjeta pulsada por el usuario coincide con la tarjeta pulsada anteriormente como primer elemento de la pareja actual (variable **firstCard** del juego), el método retorna y no hace nada más.

Si el método continúa su ejecución normal después de las tres comprobaciones anteriores porque ninguna de ellas se cumple, se deben realizar las siguientes acciones:

1. Modificar el atributo **data-state** de la tarjeta al valor **flip** para que la tarjeta se dé la vuelta y se vea la imagen.
2. Comprobar si el juego ya tenía una tarjeta volteada a través de la variable **flippedCard**
    a. En caso negativo, poner la variable **flippedCard** a verdadero y asignar a la variable **firstCard** el valor de la tarjeta actual (this).
    b. En caso afirmativo, asignar a la variable **secondCard** el valor de la tarjeta actual (this) e invocar al método **checkForMatch**.

## Guía para resolver la tarea 9

Para la aplicación de las transformaciones y transiciones de CSS que permiten animar el volteo de las tarjetas del juego de memoria y hacer que la tarjeta se vea de forma diferente en función de si está boca arriba o boca abajo, consulta la información relativa al módulo [TRANSFORMS] a través de los siguientes enlaces:

- Referencia: https://www.w3.org/Style/CSS/current-work
- Ejemplo de tarjeta con volteo: https://jsfiddle.net/solisjaime/72ujf08a/

## Tarea 10. Modificación del constructor de la clase Memoria

Una vez creados todos los componentes del juego, es necesario modificar el constructor de la clase Memoria para que el juego se inicialice cuando se construya la instancia de dicha clase y modificar el documento html para incluir la llamada que inicialice el juego.

## Guía para resolver la tarea 10

Modifica el constructor de la clase Memoria añadiendo llamadas a los métodos **shuffleElements**, **createElements** y **addEventListeners** (en ese orden).

Posteriormente, añade una etiqueta script antes del cierre de la etiqueta body en el documento memoria.html y crea una instancia de la clase Memoria para que el juego se vea en pantalla y pueda ser utilizado.

## Resultado del ejercicio 2

Una vez completado el ejercicio deberían haberse añadido los siguientes ficheros al proyecto del Escritorio Virtual:

- Fichero memoria.js en el directorio js del proyecto
- Fichero memoria.html en el directorio raíz del proyecto
- Fichero memoria.css en el directorio estilo del proyecto

Además, se debe haber modificado la sección de menú del fichero juegos.html para incluir el enlace de acceso al juego de memoria.

# Ejercicio 3: Juego de tiempo de reacción

En este ejercicio se utilizará JavaScript en combinación con HTML y CSS para realizar un juego que mida el tiempo de reacción del usuario.

El tiempo de reacción de un piloto de F1 es un factor determinante para poder realizar ciertas acciones de la carrera (por ejemplo, la salida) en condiciones óptimas y es uno de los elementos que más entrenan los pilotos.

Para crear el juego se representará un semáforo como el utilizado en las carreras de F1 donde se irán iluminando las luces en color rojo para después apagarse todas cuando el semáforo "dé la salida de la carrera", midiendo el tiempo que tarda el usuario en pulsar el ratón desde que las luces se apagan. Cuanto menor sea el tiempo transcurrido entre el apagado del semáforo y el clic realizado por el usuario, mejor será el tiempo de reacción.

Utilizando Grid Layout se creará una estructura en celdas que represente las luces del semáforo. En combinación con la estructura anterior se tratará el evento de clic para contabilizar el tiempo que pasa entre el apagado del semáforo y la reacción del usuario.

**IMPORTANTE: En este ejercicio se permite la utilización del bloque anónimo div únicamente para la representación de las luces del semáforo. Los estilos de CSS asociados a las luces del semáforo no pueden utilizar como selector el bloque div.**

**IMPORTANTE: En este ejercicio se permite el uso del atributo class de HTML para utilizarlo en la funcionalidad de apagar y encender las luces del semáforo.**

**IMPORTANTE: El uso del bloque anónimo div y el atributo class de HTML, así como el uso de selectores de CSS a partir de los atributos class sólo se permiten en aquellos ejercicios que lo indiquen explícitamente y el incumplimiento de esta restricción invalidará el proyecto.**

## Tarea 1. Creación del documento semaforo.html

Crea el fichero semaforo.html en el directorio raíz del proyecto de Escritorio F1.

Añade al menú creado en el ejercicio anterior dentro del fichero juegos.html un enlace que permita el acceso al fichero semaforo.html

## Tarea 2. Creación de la clase Semáforo

Dentro de la carpeta js del directorio del proyecto Escritorio Virtual crea un nuevo fichero llamado semaforo.js. Añade una referencia a este nuevo fichero en el documento semaforo.html.

Dentro del fichero semaforo.js crea una clase Semáforo que tenga los siguientes atributos:

- Un array de números llamado **levels** que represente la dificultad del juego con los valores 0.2, 0.5, 0.8
- Un entero llamado **lights** que represente el número de luces del semáforo; **valor de incialización 4**.
- Una fecha llamada **unload_moment** que almacenará el momento en el que se inicia la secuencia de apagado del semáforo; **valor de inicialización null**.

- Una fecha llamada **clic_moment** que almacenará el momento en el que el usuario pulsa el botón que determina su tiempo de reacción; **valor de inicialización null**.

Añade a la clase Semáforo un método constructor que inicialice la dificultad del juego de forma aleatoria.

## Guía para resolver la tarea 2

Crea un atributo de nombre **difficulty** para almacenar la dificultad de juego e inicialízalo a partir de un número aleatorio de ECMAScript que permita acceder a una de las variables de dificultad almacenadas en el array **levels**, utilizando el método random del objeto predefinido Math. Para ello, debes generar un número aleatorio comprendido entre 0 (incluido) y 3 (excluido) y usarlo para acceder a la posición correspondiente del array **levels**.

Más información: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/random

## Tarea 3. Inicializando el semáforo

Crea un método auxiliar de nombre **createStructure** dentro de la clase Semáforo.

Este método será el encargado de crear en el documento HTML las luces del semáforo y el resto de elementos que se utilizarán en el juego a través de ECMAScript.

## Guía para resolver la tarea 3

El método **createStructure** debe añadir dentro de la etiqueta main del documento los siguientes elementos: un encabezado para el título del juego, tantos bloques div para luces del semáforo como número de luces tenga la variable **lights** y dos botones, uno para arrancar el semáforo y otro que se pulse para obtener el tiempo de reacción.

El uso del div para la representación de las luces viene dado por la falta de una etiqueta adecuada para el caso, siguiendo la recomendación recogida en el estándar de HTML:

> *Authors are strongly encouraged to view the* div *element as an element of last resort, for when no other element is suitable. Use of more appropriate elements instead of the* div *element leads to* ***better accessibility*** *for readers and* ***easier maintainability*** *for authors.*

Se puede consultar la información sobre el bloque anónimo div en el siguiente enlace: https://html.spec.whatwg.org/multipage/grouping-content.html#the-div-element

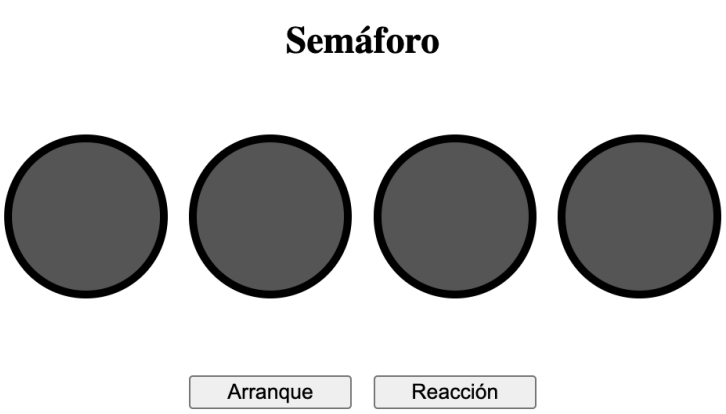
Invoca al método **createStructure** dentro del constructor de la clase Semáforo.

## Tarea 4. Pintando el juego del semáforo

Crea una estructura de cuadrícula que permita representar el semáforo en la pantalla. El semáforo va a tener 4 luces por lo que la cuadrícula deberá tener ese número de columnas. En cuanto a las filas: el encabezado irá encima de las luces del semáforo, los botones debajo del semáforo y la información del tiempo de reacción se situará debajo de los botones, por lo que la cuadrícula tendrá 4 filas.

Para simplificar el proceso de asignación del tamaño de las celdas se debe establecer un tamaño fijo para la sección, de tal forma que las celdas se repartan el tamaño de forma igualitaria. El tamaño de la sección ha de establecerse en medidas relativas, en concordancia con las normas de la asignatura con respecto al uso de medidas absolutas.

El resultado final debe ser similar al que se muestra en la siguiente imagen:




## Guía para resolver la tarea 4

Aplica GRID Layout para conseguir la disposición de los elementos del semáforo que se observa en la imagen de ejemplo. Consulta las características del módulo [CSS-GRIDLAYOUT] a través de los siguientes enlaces:

- Referencia: https://www.w3.org/Style/CSS/current-work
- Más información: https://css-tricks.com/snippets/css/complete-guide-grid/ y https://jsfiddle.net/solisjaime/ztos4689/

Utiliza el color de fondo y el borde del elemento para hacer que el borde del círculo sea más oscuro que la zona que se iluminará de rojo, para que dé la sensación de ser un semáforo real. Puedes utilizar el color negro para el borde (#000 en hexadecimal) y un color negro con una transparencia en el **canal alpha** para el centro de la luz (#0007 en hexadecimal). Consulta las características del **canal alpha** de los colores en el siguiente enlace: https://developer.mozilla.org/en-US/docs/Web/CSS/hex-color

Para conseguir que todas las luces del semáforo sean un círculo perfecto será necesario hacer que las celdas sean cuadradas. Para ello, utilizaremos la propiedad **aspect-ratio** de CSS, que permite establecer la relación de aspecto de la caja que genera agente de usuario para un elemento; el valor que es necesario utilizar para esta propiedad es 1. Consulta más sobre esta propiedad en el siguiente enlace: https://developer.mozilla.org/en-US/docs/Web/CSS/aspect-ratio

Para hacer que las luces del semáforo sean redondas, utiliza la propiedad **border-radius** de CSS que redondea las esquinas de las cajas. Para conseguir que el círculo sea perfecto, el **border-radius** aplicado debería ser la mitad del tamaño de la caja que el agente de usuario asigna a cada bloque div que representa a una luz. Consulta más sobre esta propiedad en el siguiente enlace: https://developer.mozilla.org/en-US/docs/Web/CSS/border-radius

## Tarea 5. Añadiendo el evento al botón de arranque del semáforo

El primer botón del juego, con el texto "Arranque", será el encargado de hacer que las luces del semáforo se vayan encendiendo una a una.

Crea el método **initSequence** dentro de la clase semáforo y configura el botón para que invoque a este método. Haz que este método añada un atributo al código HTML para desencadenar la animación de las luces del semáforo.

## Guía para resolver la tarea 5

El método **initSequence** añadirá la clase de CSS **load** a la etiqueta main para provocar que las luces del semáforo se vayan encendiendo. Esta clase será la encargada de animar el encendido del semáforo, haciendo que las luces se vayan encendiendo de una en una hasta estar todas rojas. Consulta las propiedades del objeto Element que representa los nodos del árbol DOM en ECMAScript para saber cómo añadir clases a un nodo:

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

Una vez que el botón de arranque sea pulsado se deberá deshabilitar, ya que una vez iniciada la secuencia de arranque no se podrá volver a arrancar hasta que haya terminado dicha secuencia. Consulta las propiedades de los botones en el siguiente enlace para conocer cuál de ellas permite deshabilitar su funcionamiento:

https://developer.mozilla.org/en-US/docs/Web/HTML/Element/button - attributes

Recuerda modificar el código del método **createStructure** para añadir al botón la invocación al método **initSequence** en el evento **onclick**, estableciendo el valor para el atributo correspondiente.

## Tarea 6. Encendiendo las luces del semáforo

Añade a la CSS el código necesario para que se ejecute una animación que vaya encendiendo las luces del semáforo una a una hasta que todas estén encendidas, utilizando la clase **load**.

## Guía para resolver la tarea 6

Una animación de CSS permite que un elemento altere una o varias de sus propiedades a lo largo de un espacio de tiempo concreto. Las animaciones pueden desencadenarse de forma automática o pueden verse desencadenadas por alguna acción del usuario bien sobre el elemento que se anima (por ejemplo, haciendo clic) o sobre otro elemento de la página.

En el caso del encendido del semáforo, la animación se desencadena a partir de una pulsación en el botón de arranque que añade una clase de CSS al elemento main del documento HTML y la propiedad de CSS que se verá afectada por esta animación es el color de fondo (**background-color)** de los bloques div que representan las luces del semáforo, que pasarán de negro (apagado) a rojo (encendido).

La duración de la animación de encendido de las luces del semáforo será de 0.5s, siendo necesario tener en cuenta que cada luz se encenderá de forma independiente y con posterioridad al encendido de la luz inmediatamente anterior; esto implica que la segunda y

sucesivas ejecuciones de la animación tendrán un retraso (**delay**) en su ejecución en proporción a su posición en el semáforo. Además, las luces deben permanecer en el estado final de la animación (encendidas) cuando esta termine, por lo que será necesario utilizar la propiedad **animation-fill-mode.**

Consulta la información relativa a las animaciones de CSS a través de **keyframes** en el siguiente enlace: https://developer.mozilla.org/en-US/docs/Web/CSS/@keyframes

Consulta el siguiente ejemplo de animación utilizando **keyframes**:
https://jsfiddle.net/solisjaime/s9om1uqb/

## Tarea 7. Apagando las luces del semáforo

Añade a la CSS el código necesario para que se ejecute una animación que apague todas las luces del semáforo a la vez y utiliza la variable **difficulty** para crear un timeout aleatorio de tiempo que desencadene dicha animación.

## Guía para resolver la tarea 7

Una animación de CSS permite que un elemento altere una o varias de sus propiedades a lo largo de un espacio de tiempo concreto. Las animaciones pueden desencadenarse de forma automática o pueden verse desencadenadas por alguna acción del usuario bien sobre el elemento que se anima o sobre otro elemento de la página.

En el caso del apagado del semáforo, la animación se desencadena de forma automática y la propiedad de CSS que se verá afectada por esta animación es el color de fondo (**background-color)** de los bloques div que representan las luces del semáforo, que pasarán de rojo (encendido) a negro (apagado). La duración de la animación de apagado del semáforo será de 0.1s y se aplicará a todas las luces a la vez.

La clase de css que se utilizará para el apagado del semáforo será **unload** y el nombre de la animación será **out**.

Consulta la información relativa a las animaciones de CSS a través de **keyframes** en el siguiente enlace: https://developer.mozilla.org/en-US/docs/Web/CSS/@keyframes

El método setTimeout de ECMAScript permite ejecutar un trozo de código una vez que ha expirado un temporizador que se le pasa como parámetro. Tomando en consideración que la secuencia completa de encendido del semáforo toma 1.5 segundos (1500 ms) y usando el valor de la variable **difficulty** multiplicado por 100, configura un timeout dentro del método **initSequence** de la clase Semáforo, debajo del resto del código de ese método; cuando se consuma ese timeout se deben realizar dos acciones:

- obtener la fecha y hora actual, almacenándola en el atributo **unload_moment**
- invocar al método **endSequence**.

El método **endSequence** es el encargado de habilitar el segundo botón del juego, para que el usuario pueda pulsarlo y registrar su tiempo de reacción. Dentro de este método también se debe añadir la clase **unload** a la etiqueta main del documento para que se ejecute el apagado de las luces del semáforo. Consulta las propiedades de los botones en el siguiente enlace para conocer cuál de ellas permite habilitar su funcionamiento:

https://developer.mozilla.org/en-US/docs/Web/HTML/Element/button#attributes

## Tarea 8. Mostrando el tiempo de reacción en pantalla

El segundo botón del juego, con el texto "Reacción", será el encargado de registrar el tiempo de reacción del usuario al apagado del semáforo.

Crea el método **stopReaction** dentro de la clase semáforo y configura el botón para que invoque a este método. Haz que este método muestre el tiempo de reacción del usuario en la pantalla y habilite de nuevo el botón de arranque para permitir reiniciar el juego.

## Guía para resolver la tarea 8

El método **stopReaction** debe realizar las siguientes acciones, por orden: en primer lugar, obtener la fecha actual y guardarla en la variable **clic_moment**; en segundo lugar, calcular la diferencia entre los valores de las variables **unload_moment** y **clic_moment** en milisegundos, siendo esta diferencia el tiempo de reacción del usuario; en tercer lugar, crear un párrafo donde informar el usuario de su tiempo de reacción y mostrarlo por pantalla; en cuarto lugar, quitar las clases **load** y **unload** de la etiqueta main; y en quinto y último lugar, deshabilitar el botón "Reacción" y habilitar el botón "Arranque".

Consulta el siguiente enlace relacionado con los métodos de la clase Date de ECMAScript para conocer que método debe invocarse a la hora de hacer el cálculo del tiempo transcurrido entre las fechas **unload_moment** y **clic_moment** en milisegundos: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date#instance_methods

Consulta el siguiente enlace relacionado con el objeto Number de ECMAScript para conocer qué método debe invocarse para redondear el tiempo de reacción del usuario a 3 decimales, que son el número de decimales que se usan para los tiempos en la Fórmula 1: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number

## Resultado del ejercicio 3

Una vez completado el ejercicio deberían haberse añadido los siguientes ficheros al proyecto del Escritorio Virtual:

- Fichero semaforo.js en el directorio js del proyecto
- Fichero semaforo.html en el directorio raíz del proyecto
- Fichero semaforo_grid.css en el directorio estilo del proyecto

Además, se debe haber modificado la sección de menú del fichero juegos.html para incluir el enlace de acceso al juego del semáforo.

# 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. 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.

# Anexo I. Circuitos del Mundial de Fórmula 1 2024

El listado contiene los 24 circuitos del mundial de F1.

| CODIGO | CIRCUITO | PAÍS | CIUDAD |
|---|---|---|---|
| 1 | BAHRÉIN | BAHRÉIN | SAHKIR |
| 2 | YEDDAH | ARABIA SAUDÍ | YEDDAH |
| 3 | ALBERT PARK | AUSTRALIA | MELBOURNE |
| 4 | SUZUKA | JAPÓN | SUZUKA |
| 5 | SHANGHAI | CHINA | SHANGHAI |
| 6 | MIAMI | ESTADOS UNIDOS | MIAMI |
| 7 | A. ENZO E DINO FERRARI | ITALIA | IMOLA |
| 8 | MÓNACO | MÓNACO | MONTECARLO |
| 9 | GILLES VILLENEUVE | CANADÁ | MONTREAL |
| 10 | BARCELONA-CATALUNYA | ESPAÑA | BARCELONA |
| 11 | RED BULL RING | AUSTRIA | KNITTELFELD |
| 12 | SILVERSTONE | GRAN BRETAÑA | SILVERSTONE |
| 13 | HUNGARORING | HUNGRÍA | MOGYORÓD |
| 14 | SPA-FRANCORCHAMPS | BÉLGICA | SPA-FRANCORCHAMPS |
| 15 | ZANDVOORT | PAÍSES BAJOS | ZANDVOORT |
| 16 | MONZA | ITALIA | MONZA |
| 17 | BAKÚ CITY | AZERBAIYÁN | BAKÚ |
| 18 | MARINA BAY | SINGAPUR | SINGAPUR |
| 19 | C. OF THE AMERICAS | ESTADOS UNIDOS | AUSTIN |
| 20 | A. HERMANOS RODRÍGUEZ | MÉXICO | MÉXICO D.F |
| 21 | JOSÉ CARLOS PACE (INTERLAGOS) | BRASIL | SAO PAULO |
| 22 | LAS VEGAS | ESTADOS UNIDOS | LAS VEGAS |
| 23 | LOSAIL | QATAR | LOSAIL |
| 24 | YAS MARINA | EMIRATOS ÁRABES | ABU DHABI |