

ACTIVIDAD 4

CONVERTIDOR HEXADECIMAL

Vamos a implementar una aplicación Web que al cargarse tiene el siguiente aspecto. El archivo **colores.html** contiene el siguiente formulario:



El formulario permite especificar un color en modelo hexadecimal y convertirlo a RGB o viceversa, mostrando en el panel derecho una muestra de color.

Nótese que en la parte inferior hay dos capas con id **errores** y **listado** respectivamente que inicialmente están ocultas.

Todos los eventos se definirán e implementarán en el archivo **convertidor.js**.

Para almacenar los colores y realizar las conversiones se proporciona el archivo **color.js** una clase llamada **Color** que nos permite gestionar un color tanto en formato RGB numérico como en Hexadecimal

| Clase Color | | | |
|--|-----------------------|-------------|--|
| Propiedades | | | |
| Nombre | Tipo | Visibilidad | Descripción |
| #valorRGB | entero[3] | privada | Array de 3 elementos que contiene la representación en RGB del color. El primer elemento del array es el componente rojo, el segundo es el verde y el tercero es el azul. |
| #valorHex | cadena | Privada | Cadena de caracteres que contiene la representación en hexadecimal del color. |
| Métodos | | | |
| Nombre | Parámetros | Devuelve | Descripción |
| Color() Color(hex) Color([r,g,b]) | hex:cadena [r,g,b] | - | Constructor: <ul style="list-style-type: none"> En caso de no recibir ningún parámetro, el color se inicializará a blanco (#ffffff). En caso de recibir un único parámetro de tipo cadena, este representa una cadena hexadecimal. En caso de recibir un array, estos representan los colores rgb (rojo, verde y azul). Independientemente del número de parámetros recibido, las propiedades valorRGB y valorHex tendrán que inicializarse correctamente |

| | | | |
|--------------------------|---------------|-----------|---|
| set ValorRGB(rgb) | rgb:entero[3] | - | Establece el valor de la propiedad valorRGB Nótese que el valor de la propiedad valorHex debería actualizarse. |
| set ValorHex(hex) | hex:cadena | - | Establece el valor de la propiedad valorHex Nótese que el valor de la propiedad valorRGB debería actualizarse. |
| get Rojo() | - | entero | Devuelve el valor correspondiente al rojo |
| get Verde() | - | entero | Devuelve el valor correspondiente al verde |
| get Azul() | - | entero | Devuelve el valor correspondiente al azul |
| get ValorHex() | - | cadena | Devuelve el valor hexadecimal |
| #RGB2Hex(rgb) | rgb:entero[3] | cadena | Método privado. Recibe 1 array con los 3 valores en RGB y calcula su equivalente en hexadecimal, devolviéndolo en forma de cadena. |
| #hex2RGB(cadena) | hex:cadena | entero[3] | Método privado. Recibe 1 cadena que representa un color en formato hexadecimal y calcula su equivalente en RGB, devolviéndolo en forma de array de 3 elementos. |

NOTA: Para una explicación más detallada sobre los modos de color y cómo convertir entre ellos, consultar el Anexo I: Colores Web.

Es necesario controlar que en los campos de texto se introduzcan números (entre 0 y 255) en el caso de los campos R, G, B y hexadecimal de 6 cifras en el campo # (ojo, no vale cualquier combinación de 6 caracteres). Queda a elección del alumno cómo implementar esta funcionalidad. Alguna opción puede ser:

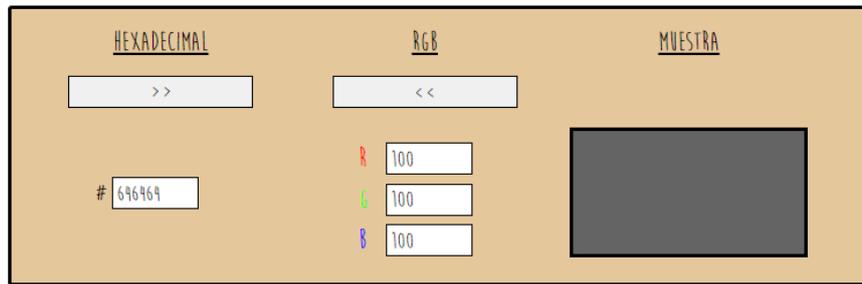
- Usar propiedades de los campos input de html
- Usar expresiones regulares
- Usar funciones de tratamiento de cadenas
- Etc.

Los valores iniciales de los campos deberían ser los correspondientes al color blanco. Para ello, en el evento de carga de página tendremos que instanciar un nuevo objeto de tipo **Color** (usando el constructor sin parámetros) y rellenar los campos de texto de la página a partir de los getters del objeto. Además, el foco debería estar situado en la caja de texto correspondiente al valor hexadecimal.

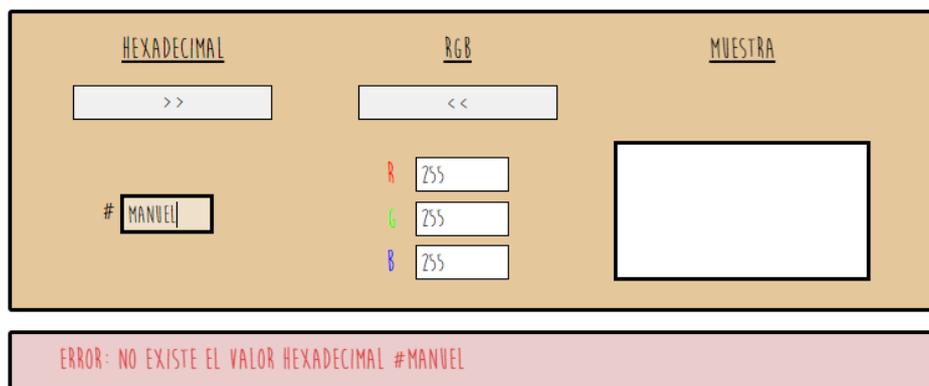
La funcionalidad de los botones >> y << debería ser la siguiente:

- Al pulsar sobre el botón >> el valor hexadecimal se convertirá a RGB y se muestra el color de fondo sobre la capa de muestra. Para realizar las conversiones se usará una nueva instancia de **Color** que crearemos en el evento de pulsación sobre el botón.

- Al pulsar sobre el botón << los valores R, G y B se convierten a hexadecimal y se muestra el color de fondo de la muestra. Para realizar las conversiones se usará la instancia de **Color**.



En caso de alguno de los campos no tenga un valor válido (habrá que realizar la validación previamente), aparecerá un mensaje en el campo con id **errores** y el **foco pasará al campo incorrecto**:



Por otra parte, se quiere recordar el último valor correcto que hemos introducido antes de cerrar la página. Para ello, vamos a **guardar el valor hexadecimal** una vez que se cierra la página con **LocalStorage**. Se usará el evento **onunload** de **window** para detectar el cierre de página.

Al **cargar la página** de nuevo, obtendremos dicho valor y actualizaremos de manera oportuna los campos de texto y el panel. En caso de que no exista ningún valor almacenado en local al cargar la página, se cargará el color por defecto de la página (blanco) como se explica en el primer punto.

RETO [OPCIONAL]

Versiona el ejercicio anterior para utilizar el “nuevo” atributo de html4 `<input type="color">`.

