Unidad de Trabajo 6: Práctica tutorizada sobre Manejo del DOM mediante JavaScript

Aplicación To-Do List

Introducción

Mediante esta práctica vas a construir una aplicación web que permite al usuario gestionar las tareas en un día determinado. Esta aplicación permite agregar nuevas tareas, marcarlas como realizadas y finalmente ordenarlas en función de si están completas o no. A la hora de programar te servirá no sólo para afianzar el manejo del DOM mediante la selección y creación de nodos sino también en los eventos así como en el manejo del tiempo mediante JavaScript. Es no sólo un proyecto para afianzar y mejorar tus conocimientos y habilidades sino también una aplicación interesante para, con ciertas mejoras, incluir en tu portafolio web. Esta práctica está basada en el recurso dado por freeCodeCamp.org en el que se proponen 7 proyectos para practicar HTML, CSS y JavaScript. Te invito a que visites el enlace a los mismos y realices otros de los proyectos ahí presentados.

Parte 1: Configurando el archivo index.html

En primer lugar, vamos a ver los elementos que forman parte del archivo HTML. En este archivo, tienes como elementos más destacables en el encabezado un enlace a la fuente Lato que se va a emplear en el proyecto y, en el caso de la etiqueta script, puedes ver que se ha añadido el atributo **defer**. Este atributo, indica al navegador que el script, aún siendo descargado y analizado junto con el código HTML y CSS, no se va a ejecutar hasta que el árbol del DOM ha sido construido, permitiendo así evitar errores en la selección de elementos del árbol de nodos. Cabe destacar que, este atributo sólo es válido cuando el script está en un archivo externo, no siendo útil en aquellos casos en los que quieras incluir JavaScript en el propio HTML.

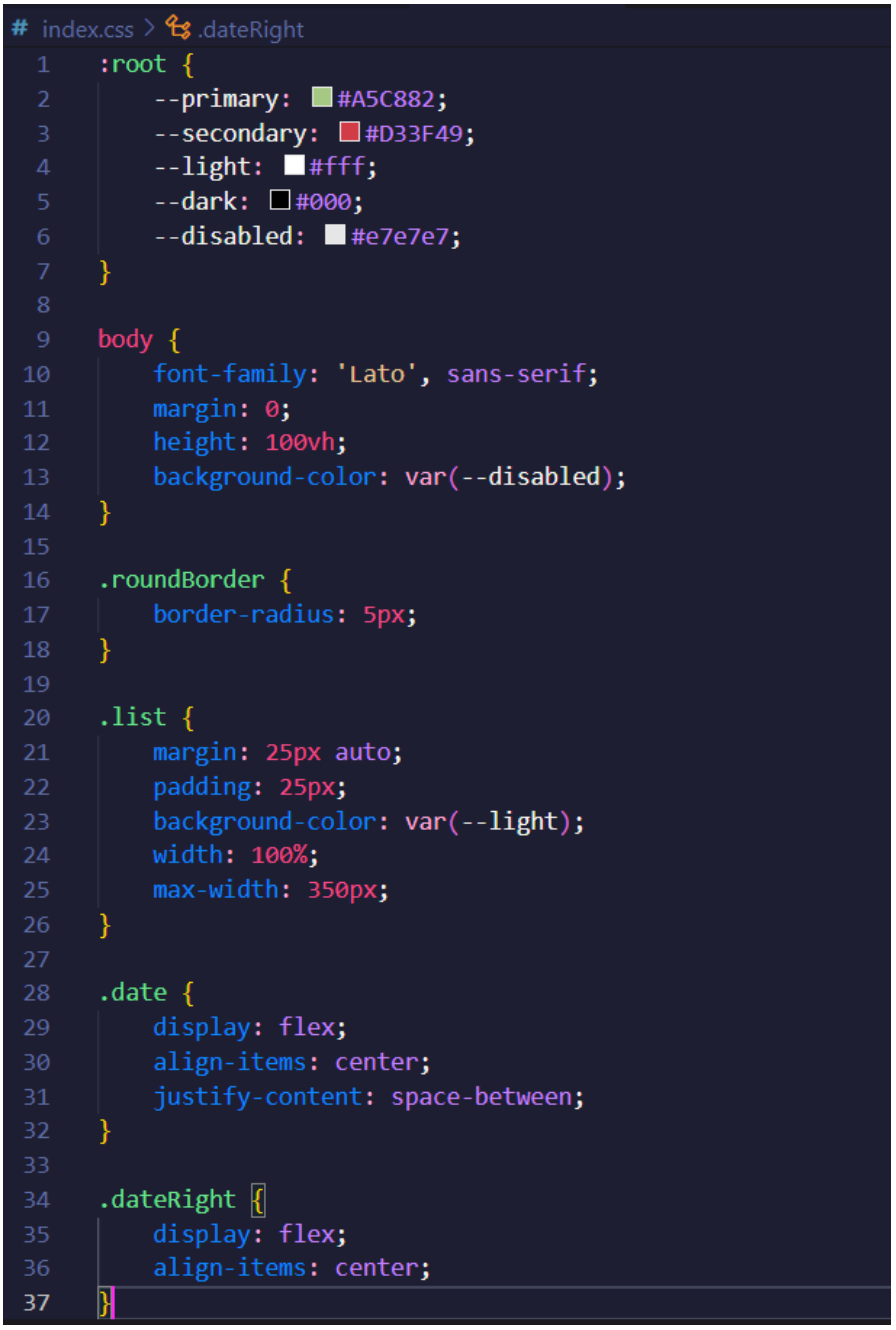
En cuanto al body, además de los elementos div para mostrar la fecha actual, tendrás un elemento **form** para los campos de input así como los botones. Fíjate que, uno de los botones se ha definido como **type = submit** mientras que el otro se define como **type = button**. Esto se ha hecho así para diferenciar a continuación los eventos que asociaremos a esos elementos, de modo que el navegador pueda distinguir fácilmente entre el evento submit del evento *click*, asociándolo convenientemente a cada botón. Por último, nota que se ha usado un modelo de gestión de eventos en línea que, tal y como sabes, puede no ser lo más conveniente actualmente.

index.html > ...

```html
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Lista de Tareas</title>
    <link href="https://fonts.googleapis.com/css2?family=Lato:wght@100;400;900&display=swap" rel="stylesheet">
    <link rel="stylesheet" href="styles.css">
    <script src="script.js" defer></script>
</head>
<body>
    <div class="list roundBorder">
        <div class="date">
            <div class="dateRight">
                <div id="dateNumber"></div>
                <div>
                    <div id="dateMonth"></div>
                    <div id="dateYear"></div>
                </div>
            </div>
            <div id="dateText"></div>
        </div>
        <form onsubmit="addNewTask(event)">
            <input type="text" name="taskText" autocomplete="off" placeholder="Nueva tarea" class="roundBorder">
            <button type="submit" class="addTaskButton">+</button>
            <button type="button" class="orderButton roundBorder" onclick="renderOrderedTasks()">Ordenar</button>
        </form>
        <div id="tasksContainer"></div>
    </div>
</body>
</html>
```

Parte 2: Configurando el archivo styles.css

A continuación se muestra la configuración del archivo de estilo CSS. Una parte a destacar del mismo es la clase **done** que permite modificar el aspecto de las tareas según estén marcadas como hechas o no. Igualmente, la pertenencia de un elemento a esa clase permite ordenar las tareas mediante JavaScript para distinguir entre aquellas que están realizadas de las que no.



index.css > #dateNumber

```css
#dateNumber {
    font-size: 50px;
}

#dateText {
    letter-spacing: 3px;
    text-transform: uppercase;
}

#dateMonth, #dateYear {
    margin-left: 7px;
    font-size: 20px;
}

form {
    margin: 20px 0;
    display: grid;
    grid-auto-flow: column;
    grid-template-columns: auto 50px auto;
    grid-column-gap: 15px;
}

input {
    border: none;
    padding-left: 10px;
    background-color: var(--disabled);
}

input:focus {
    outline: none;
}

.addTaskButton, .orderButton {
    border: none;
    font-weight: bold;
    cursor: pointer;
}
```

```css
# index.css > #dateNumber
75
76  .addTaskButton {
77      height: 35px;
78      border-radius: 50%;
79      font-size: 18px;
80      background-color: var(--disabled);
81  }
82
83  .orderButton {
84      background-color: var(--secondary);
85      color: var(--light);
86  }
87
88  .task {
89      background-color: var(--primary);
90      padding: 15px;
91      margin-top: 15px;
92      color: var(--light);
93      cursor: pointer;
94  }
95
96  .done {
97      background-color: var(--disabled);
98      text-decoration: line-through;
99      color: var(--dark);
100 }
101
```

Parte 3: Configurando el archivo script.js

A continuación vamos a ver detenidamente la configuración del archivo script explicando su funcionalidad.

Paso 1: Seleccionar los elementos del DOM

En este caso, se selecciona cada uno de los **div** para mostrar los diferentes campos de la fecha así como el empleado para mostrar las tareas agendadas por el usuario.

```javascript
JS index.js > ...
1   // Selección de elementos del DOM
2   const dateNumber = document.getElementById('dateNumber');
3   const dateText = document.getElementById('dateText');
4   const dateMonth = document.getElementById('dateMonth');
5   const dateYear = document.getElementById('dateYear');
6   const tasksContainer = document.getElementById('tasksContainer');
7
```

Paso 2: Creación de la función que genera la fecha actual y la muestra en pantalla

En este caso, las funciones se han incorporado como **arrow functions**.

¿Qué es una arrow function?

Una *arrow function* o función de flecha es una sintaxis más corta para declarar funciones en JavaScript. Este tipo de función usa la sintaxis **() => {}**, eliminando la necesidad de la palabra clave function. Además, las funciones flecha tratan el contexto *this* de manera diferente, lo que es útil en algunas situaciones.

Sintaxis básica

```
const nombreFuncion = (parametro1, parametro2) => {
  // cuerpo de la función
  return parametro1 + parametro2;
};
```

Ejemplo simplificado

Si la función solo tiene una línea y devuelve un valor, se puede omitir las llaves {} y la palabra clave return:

```
const sumar = (a, b) => a + b;
```

Para funciones con un solo parámetro, también se pueden omitir los paréntesis:

```
const cuadrado = x => x * x;
```

¡Importante!

1. **No tienen su propio this**: El this en una *arrow function* hace referencia al contexto en el que la función fue definida, no al contexto desde donde se invocó. Esto las hace útiles en métodos dentro de clases o en funciones setTimeout, map, filter, etc., ya que evita problemas con el *this*.
2. **No tienen arguments**: Las *arrow functions* no tienen acceso a arguments, que es un objeto que contiene todos los argumentos pasados a la función. Para funciones con argumentos variables, es mejor usar una función tradicional.

Ejemplo práctico

```
function Persona() {
  this.edad = 0;

  setInterval(() => {
    this.edad++; // `this` hace referencia a la instancia de Persona
    console.log(this.edad);
  }, 1000);
```

```
}

const persona = new Persona();
```

En este caso, la función flecha en **setInterval** mantiene el contexto *this* de la instancia Persona, lo cual sería diferente si usáramos una función tradicional.

Volviendo a nuestro código:

```
// Función para mostrar la fecha actual
const setDate = () => {
    const date = new Date();
    dateNumber.textContent = date.toLocaleString('es', { day: 'numeric' });
    dateText.textContent = date.toLocaleString('es', { weekday: 'long' });
    dateMonth.textContent = date.toLocaleString('es', { month: 'short' });
    dateYear.textContent = date.toLocaleString('es', { year: 'numeric' });
};

```

Tal y como puedes ver, una vez generada la fecha actual del objeto **Date()**, se asigna cada uno de los elementos al **div** correspondiente. Un método útil aplicado en este caso es **toLocaleString()**. Este método nos devuelve el objeto fecha directamente como un string y, además, permite seleccionar la configuración local, definiendo el idioma de la fecha (español en nuestro caso), así como las propiedades de salida. En cada caso, puedes ver como se selecciona el día del mes en formato numérico (comenzando en 1); el día de la semana en formato largo/completo (para que no abrevie el día); el mes abreviado y, por último, el año en formato numérico.

Paso 3: Creación de la función para agregar una nueva tarea

```
// Función para agregar una nueva tarea
const addNewTask = event => {
    event.preventDefault();
    const { value } = event.target.taskText;
    if (!value) return;

    const task = document.createElement('div');
    task.classList.add('task', 'roundBorder');
    task.addEventListener('click', changeTaskState);
    task.textContent = value;
    tasksContainer.prepend(task);
    event.target.reset();
};

```

En primer lugar, puesto que esta función se va a asociar al botón tipo submit del formulario se debe evitar su acción por defecto. A continuación, es de destacar la instancia **if(!value) return**. Esta línea de código, se encarga de verificar si el usuario ha introducido un valor en el input de la tarea. En caso de no ser así, el return detiene la ejecución del resto de líneas de la función, evitando la creación de variables vacías que ocupan espacio en la memoria. A

continuación, es de destacar la asignación de un listener a cada div de tarea para asignar la función que cambia el estado de la tarea de realizado a no realizado. Por último, el método **prepend()** indica al navegador que la tarea creada se añada antes del primer nodo hijo del div, asegurando que las nuevas tareas se muestren en primer lugar.

Paso 4: Creación de la función para marcar tareas como hechas, ordenarlas y mostrarlas en pantalla.

La primera función se encarga de cambiar el estado de las tareas de no realizadas a realizadas. Para ello, se toma el div sobre el que ocurre el evento y, mediante el método **toggle()**, se le asigna la clase done si no pertenece a dicha clase. En caso contrario, si el div en cuestión ya tiene la clase done se le elimina dicha clase volviendo a la clase task.

Por otro lado, la función **order()** selecciona todas las tareas agregadas por el usuario (childNodes) e itera sobre las mismas determinando si están marcadas como realizadas o no mediante un operador ternario. Por último, el return de esta función emplea el operador **spread**, el cual asegura que la salida de dicha función en forma de array sea correcta.

Finalmente, la función **renderOrderedTasks()** se encarga de volver a representar las tareas ordenadas en función de si están realizadas o no puesto que, al aplicarla al método order() toma como variable el return de dicha función.

```
31  // Función para cambiar el estado de la tarea a "hecho" o "pendiente"
32  const changeTaskState = event => {
33      event.target.classList.toggle('done');
34  };
35
36  // Función para ordenar las tareas (hechas primero)
37  const order = () => {
38      const done = [];
39      const toDo = [];
40      tasksContainer.childNodes.forEach(el => {
41          el.classList.contains('done') ? done.push(el) : toDo.push(el);
42      });
43      return [...toDo, ...done];
44  };
45
46  // Función para mostrar las tareas ordenadas
47  const renderOrderedTasks = () => {
48      order().forEach(el => tasksContainer.appendChild(el));
49  };
50
51  // Ejecuta la función de fecha al cargar la página
52  setDate();
```

Parte 4: Añadiendo extras a nuestro código

1. En la primera parte del proyecto, comentamos que se ha usado un modelo de gestión de eventos en línea que, tal y como sabes, puede no ser lo más conveniente actualmente. **Modifica el código (HTML y JS) y emplea una gestión de eventos basada en listeners tal y como vimos en el tema pasado.**
2. Añade un botón adicional para eliminar todas las tareas. **Añadiremos un botón en el HTML que elimine todas las tareas al hacer clic en él.**
3. Añade un botón adicional para editar las tareas. Permitiremos editar cada tarea individual. Esto implica que, **al hacer doble clic en una tarea, el texto se pueda modificar.**

Entrega:

La entrega será un .zip con el código y los extras añadidos a este. Se ha de indicar mediante un comentario las partes del código que han sido modificadas y cuáles corresponden con las funcionalidades extras.