Esercitazioni su Manipolazione HTML

1. Aggiungere un nuovo elemento alla lista

Specifiche funzionali:

- Viene fornita una lista `<ul>` con un id specifico.
- La funzione deve aggiungere un nuovo elemento `<li>` con del testo personalizzato.

Analisi Top-Down:

1. Identifica il contenitore (`<ul>`) tramite il suo id.
2. Crea un nuovo elemento `<li>`.
3. Aggiungi il testo all'interno dell'elemento `<li>`.
4. Collega l'elemento `<li>` alla lista.

Analisi Bottom-Up:

1. Comincia creando un nuovo elemento.
2. Aggiungi del testo al nuovo elemento.
3. Collega l'elemento alla lista.

Codice:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Aggiungi Elemento</title>
</head>
<body>
    <ul id="myList">
        <li>Elemento 1</li>
        <li>Elemento 2</li>
    </ul>
    <button onclick="addListItem('Nuovo Elemento')">Aggiungi</button>

    <script>
        function addListItem(text) {
            // Trova la lista tramite il suo id
            const list = document.getElementById('myList');

            // Crea un nuovo elemento <li>
            const newItem = document.createElement('li');

            // Imposta il testo del nuovo elemento
            newItem.textContent = text;

            // Aggiungi il nuovo elemento alla lista
            list.appendChild(newItem);
        }
    </script>
</body>
</html>
```

2. Cambiare il colore di un elemento

Specifiche funzionali:

- L'utente clicca su un pulsante per cambiare il colore di un elemento.

Analisi Top-Down:

1. Identifica l'elemento da modificare tramite il suo id.
2. Cambia il colore di sfondo con un nuovo valore.

Analisi Bottom-Up:

1. Usa `getElementById` per selezionare un elemento.
2. Modifica lo stile tramite la proprietà `style.backgroundColor`.

Codice:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Cambia Colore</title>
</head>
<body>
    <div id="colorBox" style="width: 100px; height: 100px; background-color:
lightblue;"></div>
    <button onclick="changeColor('yellow')">Cambia Colore</button>

    <script>
        function changeColor(color) {
            // Seleziona l'elemento
            const box = document.getElementById('colorBox');

            // Cambia il colore di sfondo
            box.style.backgroundColor = color;
        }
    </script>
</body>
</html>
```

3. Rimuovere un elemento dalla pagina

Specifiche funzionali:

- L'utente clicca un pulsante per rimuovere un elemento.

Analisi Top-Down:

1. Identifica l'elemento tramite il suo id.
2. Rimuovi l'elemento usando il metodo `remove`.

Analisi Bottom-Up:

1. Trova l'elemento da eliminare.
2. Usa il metodo `remove`.

Codice:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Rimuovi Elemento</title>
</head>
<body>
    <div id="removeMe">Questo elemento sarà rimosso.</div>
    <button onclick="removeElement('removeMe')">Rimuovi</button>

    <script>
        function removeElement(elementId) {
            // Trova l'elemento da rimuovere
            const element = document.getElementById(elementId);

            // Rimuovi l'elemento dalla pagina
            if (element) {
                element.remove();
            }
        }
    </script>
</body>
</html>
```

4. Nascondere e mostrare un elemento

Specifiche funzionali:

- Alterna la visibilità di un elemento tramite un pulsante.

Analisi Top-Down:

1. Seleziona l'elemento.
2. Controlla la proprietà `style.display`.
3. Modifica la proprietà per nascondere o mostrare l'elemento.

Codice:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Nascondi/Mostra</title>
</head>
<body>
    <div id="toggleDiv">Questo è un testo visibile.</div>
    <button onclick="toggleVisibility('toggleDiv')">Nascondi/Mostra</button>

    <script>
        function toggleVisibility(elementId) {
```

```
            // Trova l'elemento
            const element = document.getElementById(elementId);

            // Alterna la visibilità
            if (element.style.display === 'none') {
                element.style.display = 'block';
            } else {
                element.style.display = 'none';
            }
        }
    </script>
</body>
</html>
```

5. Creare un timer che aggiorna un elemento

Specifiche funzionali:

- Aggiorna il contenuto di un elemento ogni secondo.

Analisi Top-Down:

1. Usa `setInterval` per aggiornare regolarmente il contenuto.
2. Modifica il testo di un elemento.

Codice:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Timer</title>
</head>
<body>
    <div id="timer">Timer: 0</div>

    <script>
        let count = 0;
        setInterval(() => {
            // Aggiorna il contenuto del timer
            document.getElementById('timer').textContent = `Timer: ${++count}`;
        }, 1000);
    </script>
</body>
</html>
```

6. Creare una tabella dinamica

Specifiche funzionali:

- Creare una tabella con un numero specificato di righe e colonne.
- Aggiungere contenuti predefiniti nelle celle.

Analisi Top-Down:

1. Usa `createElement` per costruire gli elementi della tabella (`<table>`, `<tr>`, `<td>`).
2. Itera su righe e colonne per creare celle.
3. Aggiungi la tabella al contenitore.

Analisi Bottom-Up:

1. Crea un elemento `<table>`.
2. Genera righe (`<tr>`) e celle (`<td>`) in un ciclo.
3. Inserisci la tabella nel DOM.

Codice:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Tabella Dinamica</title>
</head>
<body>
    <div id="tableContainer"></div>
    <button onclick="createTable(3, 4)">Crea Tabella (3x4)</button>

    <script>
        function createTable(rows, cols) {
            // Crea la tabella
            const table = document.createElement('table');
            table.style.border = '1px solid black';
            table.style.borderCollapse = 'collapse';

            // Crea le righe e le colonne
            for (let i = 0; i < rows; i++) {
                const row = document.createElement('tr');
                for (let j = 0; j < cols; j++) {
                    const cell = document.createElement('td');
                    cell.textContent = `R${i + 1}C${j + 1}`;
                    cell.style.border = '1px solid black';
                    cell.style.padding = '5px';
                    row.appendChild(cell);
                }
                table.appendChild(row);
            }

            // Inserisci la tabella nel contenitore
            const container = document.getElementById('tableContainer');
            container.innerHTML = ''; // Pulisci il contenitore
            container.appendChild(table);
        }
    </script>
</body>
</html>
```

7. Aggiungere un’immagine dinamicamente

Specifiche funzionali:

- L'utente può aggiungere un'immagine alla pagina specificando un URL.

Analisi Top-Down:

1. L'utente inserisce un URL tramite input.
2. Crea un elemento `<img>` con l'attributo `src`.
3. Inserisci l'immagine nel DOM.

Codice:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Aggiungi Immagine</title>
</head>
<body>
    <input type="text" id="imageUrl" placeholder="Inserisci URL immagine" />
    <button onclick="addImage()">Aggiungi Immagine</button>
    <div id="imageContainer"></div>

    <script>
        function addImage() {
            // Ottieni l'URL dall'input
            const url = document.getElementById('imageUrl').value;

            // Crea un elemento immagine
            const img = document.createElement('img');
            img.src = url;
            img.alt = 'Immagine dinamica';
            img.style.maxWidth = '300px';
            img.style.display = 'block';
            img.style.marginTop = '10px';

            // Aggiungi l'immagine al contenitore
            const container = document.getElementById('imageContainer');
            container.appendChild(img);
        }
    </script>
</body>
</html>
```

8. Validare un form HTML

Specifiche funzionali:

- L'utente inserisce dati in un form.
- La funzione controlla se tutti i campi obbligatori sono compilati.
- Mostra un messaggio di errore o conferma.

Analisi Top-Down:

1. Ottieni i valori dei campi del form.
2. Controlla se i campi obbligatori sono vuoti.
3. Mostra un messaggio appropriato.

Codice:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Validazione Form</title>
</head>
<body>
    <form id="myForm">
        <input type="text" id="name" placeholder="Nome" required /><br />
        <input type="email" id="email" placeholder="Email" required /><br />
        <button type="button" onclick="validateForm()">Invia</button>
    </form>
    <p id="message"></p>

    <script>
        function validateForm() {
            // Ottieni i valori dei campi
            const name = document.getElementById('name').value.trim();
            const email = document.getElementById('email').value.trim();

            // Verifica se i campi sono vuoti
            if (!name || !email) {
                document.getElementById('message').textContent = 'Tutti i campi
sono obbligatori!';
                document.getElementById('message').style.color = 'red';
            } else {
                document.getElementById('message').textContent = 'Form inviato
con successo!';
                document.getElementById('message').style.color = 'green';
            }
        }
    </script>
</body>
</html>
```

9. Aggiungere una lista dinamica da un array

Specifiche funzionali:

- Creare una lista `<ul>` utilizzando gli elementi di un array.

Analisi Top-Down:

1. Crea un array di elementi.
2. Itera sull'array e crea `<li>` per ogni elemento.
3. Aggiungi gli elementi alla lista `<ul>`.

Codice:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Lista Dinamica</title>
</head>
<body>
```

```
    <div id="listContainer"></div>
    <button onclick="createList()">Crea Lista</button>

    <script>
        function createList() {
            // Array di elementi
            const items = ['Item 1', 'Item 2', 'Item 3', 'Item 4'];

            // Crea una lista
            const ul = document.createElement('ul');

            // Aggiungi gli elementi alla lista
            items.forEach(item => {
                const li = document.createElement('li');
                li.textContent = item;
                ul.appendChild(li);
            });

            // Aggiungi la lista al contenitore
            const container = document.getElementById('listContainer');
            container.innerHTML = ''; // Pulisci il contenitore
            container.appendChild(ul);
        }
    </script>
</body>
</html>
```

10. Spostare elementi tra due liste

Specifiche funzionali:

- L'utente sposta elementi tra due liste tramite pulsanti.

Analisi Top-Down:

1. Crea due liste (`<ul>`).
2. Permetti di spostare elementi da una lista all'altra.

Codice:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Sposta Elementi</title>
</head>
<body>
    <div>
        <ul id="listA">
            <li onclick="moveTo('listB', this)">Elemento 1</li>
            <li onclick="moveTo('listB', this)">Elemento 2</li>
        </ul>
        <ul id="listB">
            <li onclick="moveTo('listA', this)">Elemento 3</li>
            <li onclick="moveTo('listA', this)">Elemento 4</li>
        </ul>
    </div>
```

```
    <script>
        function moveTo(targetListId, item) {
            // Trova la lista di destinazione
            const targetList = document.getElementById(targetListId);

            // Rimuovi l'elemento dalla lista attuale e aggiungilo alla nuova
lista
            targetList.appendChild(item);
        }
    </script>
</body>
</html>
```

Lavorare con array e oggetti in JavaScript

1. Calcolare la somma di numeri in un array

Specifiche funzionali:

- Creare una funzione che accetta un array di numeri.
- Restituire la somma di tutti i numeri nell'array.

Analisi Top-Down:

1. Verifica che l'input sia un array valido.
2. Usa un ciclo per iterare sui numeri.
3. Accumula la somma.
4. Restituisci il risultato.

Analisi Bottom-Up:

1. Usa un ciclo per iterare su un array.
2. Calcola la somma con una variabile accumulativa.
3. Gestisci casi particolari (array vuoto).

Codice:

```
function sumArray(numbers) {
    // Inizializza la somma a 0
    let sum = 0;

    // Itera su ogni numero nell'array
    for (let i = 0; i < numbers.length; i++) {
        sum += numbers[i];
    }

    // Restituisci la somma
    return sum;
}

// Test della funzione
const nums = [10, 20, 30, 40];
console.log(`Somma: ${sumArray(nums)}`); // Output: Somma: 100
```

2. Filtrare oggetti in base a una proprietà

Specifiche funzionali:

- Creare una funzione che accetta un array di oggetti.
- Filtrare gli oggetti in base a una proprietà specifica.

Analisi Top-Down:

1. Itera sull'array di oggetti.
2. Controlla il valore della proprietà di ciascun oggetto.
3. Restituisci un nuovo array contenente solo gli oggetti che soddisfano la condizione.

Codice:

```
function filterObjectsByProperty(array, property, value) {
    // Usa il metodo filter per selezionare gli oggetti
    return array.filter(obj => obj[property] === value);
}

// Test della funzione
const people = [
    { name: 'Mario', age: 30 },
    { name: 'Luigi', age: 25 },
    { name: 'Peach', age: 30 },
];
const filtered = filterObjectsByProperty(people, 'age', 30);
console.log(filtered);
// Output: [ { name: 'Mario', age: 30 }, { name: 'Peach', age: 30 } ]
```

3. Creare un array di oggetti da un altro array

Specifiche funzionali:

- Partire da un array di stringhe.
- Creare un array di oggetti, dove ogni stringa diventa una proprietà.

Analisi Top-Down:

1. Itera sull'array di stringhe.
2. Trasforma ogni stringa in un oggetto.
3. Aggiungi ogni oggetto a un nuovo array.

Codice:

```
function createObjectArray(names) {
    // Trasforma ogni nome in un oggetto
    return names.map(name => ({ name }));
}

// Test della funzione
const names = ['Mario', 'Luigi', 'Peach'];
const objects = createObjectArray(names);
console.log(objects);
```

```
// Output: [ { name: 'Mario' }, { name: 'Luigi' }, { name: 'Peach' } ]
```

4. Raggruppare oggetti in base a una proprietà

Specifiche funzionali:

- Creare una funzione che raggruppa oggetti in base a una proprietà.
- Restituire un oggetto dove le chiavi sono i valori della proprietà e i valori sono array di oggetti.

Analisi Top-Down:

1. Itera sull'array di oggetti.
2. Controlla il valore della proprietà.
3. Aggiungi l'oggetto alla chiave corrispondente in un nuovo oggetto.

Codice:

```
function groupBy(array, property) {
    return array.reduce((result, obj) => {
        // Usa il valore della proprietà come chiave
        const key = obj[property];
        if (!result[key]) {
            result[key] = [];
        }
        result[key].push(obj);
        return result;
    }, {});
}

// Test della funzione
const items = [
    { type: 'frutta', name: 'Mela' },
    { type: 'frutta', name: 'Banana' },
    { type: 'verdura', name: 'Carota' },
];
const grouped = groupBy(items, 'type');
console.log(grouped);
// Output:
// { frutta: [ { type: 'frutta', name: 'Mela' }, { type: 'frutta', name:
'Banana' } ],
//   verdura: [ { type: 'verdura', name: 'Carota' } ] }
```

5. Trovare il massimo valore in un array di oggetti

Specifiche funzionali:

- Creare una funzione che trova l'oggetto con il massimo valore in una proprietà specifica.

Analisi Top-Down:

1. Itera sull'array.
2. Confronta i valori della proprietà specificata.

3. Restituisci l'oggetto con il valore massimo.

Codice:

```
function findMaxByProperty(array, property) {
    // Usa reduce per trovare l'oggetto con il valore massimo
    return array.reduce((max, obj) =>
        (obj[property] > (max[property] || -Infinity) ? obj : max), {});
}

// Test della funzione
const scores = [
    { name: 'Mario', score: 85 },
    { name: 'Luigi', score: 92 },
    { name: 'Peach', score: 88 },
];
const maxScore = findMaxByProperty(scores, 'score');
console.log(maxScore);
// Output: { name: 'Luigi', score: 92 }
```

Riepilogo

Questi esercizi mostrano diverse tecniche per gestire array e oggetti in contesti pratici:

1. **Somma di numeri**: Operazioni matematiche su array.
2. **Filtrare oggetti**: Estrazione di sottogruppi da un array.
3. **Trasformazione di array**: Creazione di strutture dati da array di base.
4. **Raggruppamento di oggetti**: Organizzazione di dati in categorie.
5. **Trovare valori massimi**: Identificazione di elementi specifici.