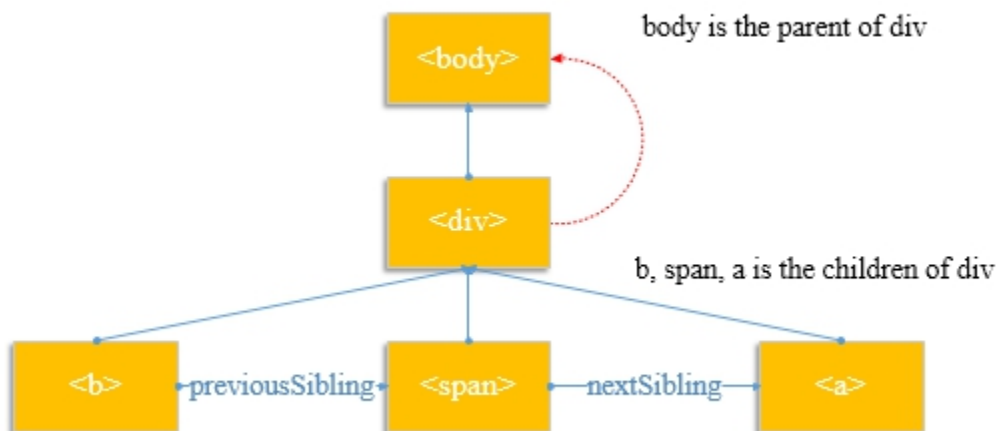


# TP répertoire

*DOM, Node, Element, create, append, remove element*

## Hiérarchie des éléments\*



### ElementHierarchy.html

```
<body>
  <div id="div">
    <b>Hello</b>
    <span>My name's Régis</span>
    <a href="#">portfolio</a>
  </div>
</body>
```

Rendu :

---

**Hello** My name's Régis [portfolio](#)

 Modifiez **ElementHierarchy.html** pour obtenir les nœuds parents.

`obj.parentNode` : obtenir les nœuds parents du nœud actuel

`obj.parentElement` : obtenir les éléments parents du nœud actuel

```
<script>
  const element = document.querySelector("#div");
  console.log("parent Element", element.parentElement);
  console.log("parent Node", element.parentNode);
  console.log("name", element.parentElement.nodeName);
</script>
```

Quelle est la différence entre *ParentNode* et *ParentElement* ?

# Les enfants

📁 Modifiez **ElementHierarchy.html** obtenir tous les nœuds enfants

`obj.childNodes` : obtenir tous les nœuds enfants du nœud actuel

`obj.nodeName` : obtenir le nom du nœud actuel

`obj.nodeType` : obtenir le type du nœud actuel

`obj.childElementCount` : obtenir le nombre d'enfant

`obj.children` : obtenir le nombre d'enfant

`obj.innerText` : obtenir le text html du nœud actuel

Quelle est la différence entre *children* et *childNodes* ?

```
<body>
  <div id="div">
    <b>Hello</b>
    <span>My name's Régis</span>
    <a href="#">portfolio</a>
  </div>
  <script>
    const element = document.querySelector("#div");
    console.log(element.children);
    console.log(element.childNodes);
  </script>
</body>
```

Résultat :

```
>> HTMLCollection { 0: b, 1: span, 2: a, length: 3 }
```

```
>> NodeList(7) [ #text, b, #text, span, #text, a, #text ]
```

# Node vs Element

Node :

- C'est une interface plus générale dans le DOM.
- Représente tout type de nœud dans un document (éléments, textes, commentaires, etc.).
- Tous les objets dans le DOM sont des Nodes.

Element :

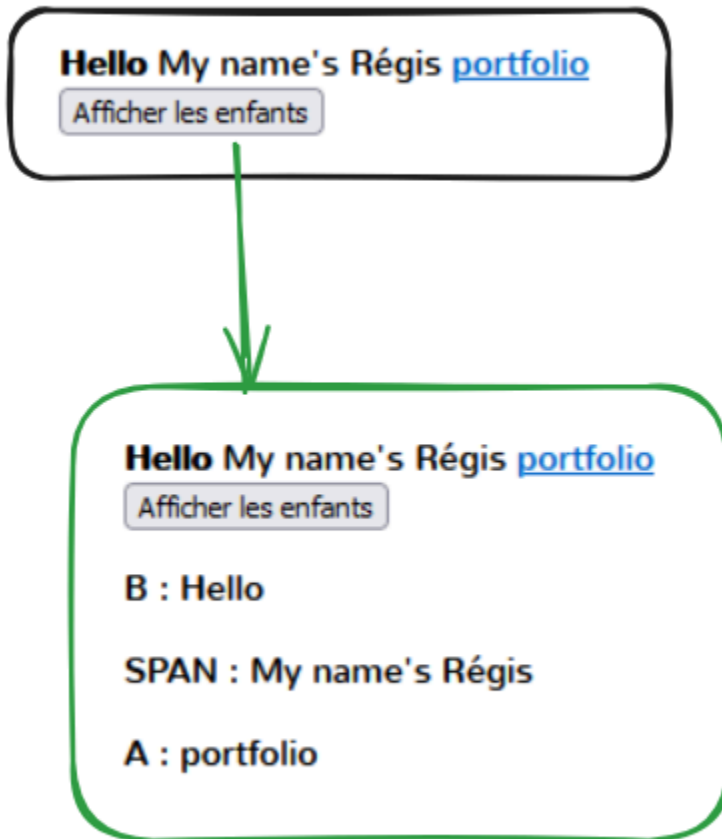
- C'est un type spécifique de Node.
- Représente un élément HTML ou XML dans le document.
- Hérite de Node, donc possède toutes ses propriétés et méthodes.

Par exemple, les nodes considèrent les textes comme des noeuds

```
<script>
  const a = document.querySelector("a");
  console.log(a.textContent);
  console.log(a.childNodes);
  let firstChild = a.childNodes[0];
  console.log(firstChild instanceof Node);
  console.log(firstChild.parentElement);
</script>
```

## Exercice :

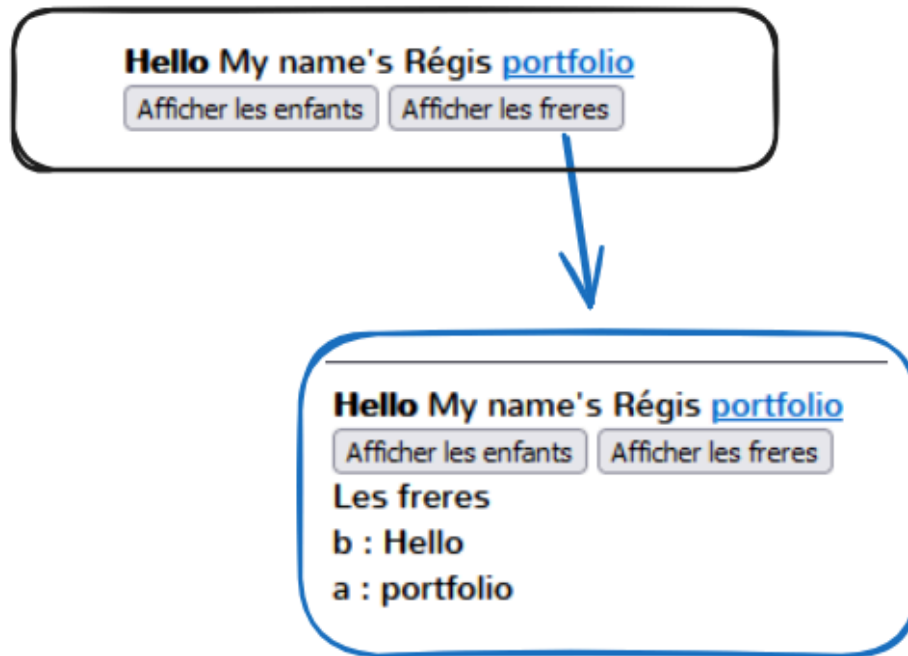
Ajouter un bouton qui au click de celui- ci affiche les enfants en précisant le nom de la balise et le contenu text de la balise.



✎ Ajouter un bouton pour obtenir le nœud précédent et suivant du nœud de “span”

`obj.previousSibling` : obtenir le nœud précédent du nœud actuel

`obj . nextSibling` : obtenir le nœud suivant du nœud actuel



# InnerHTML vs TextContent

`element.innerHTML` : récupère ou définit la syntaxe HTML décrivant les descendants de l'élément.

`element.textContent` : représente le contenu textuel d'un nœud et de ses descendants.

Exécutez le code suivant dans un fichier **textContent.html**

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="UTF-8" />
    <title>innerHTML vs textContent</title>
    <style>
      .example {
        border: 1px solid #ccc;
        padding: 10px;
        margin-bottom: 20px;
      }
      .output {
        background-color: #f0f0f0;
        padding: 5px;
        margin-top: 10px;
      }
    </style>
  </head>
  <body>
    <h1>Différence entre innerHTML et textContent</h1>

    <div class="example">
      <h2>Exemple 1: Texte simple</h2>
      <div id="example1">
        Ceci est un <strong>texte</strong> avec des balises.
      </div>
      <button onclick="showInnerHTML('example1')">Afficher
innerHTML</button>
      <button onclick="showTextContent('example1')">
        Afficher textContent
      </button>
      <div id="output1" class="output"></div>
    </div>
```

```

<div class="example">
  <h2>Exemple 2: Ajout de contenu</h2>
  <div id="example2"></div>
  <button onclick="addWithInnerHTML()">Ajouter avec innerHTML</button>
  <button onclick="addWithTextContent()">Ajouter avec
textContent</button>
</div>

<script>
  function showInnerHTML(id) {
    const element = document.getElementById(id);
    document.getElementById("output1").textContent =
      "innerHTML: " + element.innerHTML;
  }

  function showTextContent(id) {
    const element = document.getElementById(id);
    document.getElementById("output1").textContent =
      "textContent: " + element.textContent;
  }

  function addWithInnerHTML() {
    const element = document.getElementById("example2");
    element.innerHTML =
      "Ceci est du <strong>HTML</strong> ajouté avec innerHTML.";
  }

  function addWithTextContent() {
    const element = document.getElementById("example2");
    element.textContent =
      "Ceci est du <strong>HTML</strong> ajouté avec textContent.";
  }
</script>
</body>
</html>

```



## Faible de sécurité liée à innerHTML

L'utilisation de `innerHTML` peut présenter un risque de sécurité significatif, notamment lorsqu'il est utilisé avec du contenu généré par l'utilisateur ou provenant de sources non fiables.

La principale vulnérabilité est connue sous le nom d'attaque par "Cross-Site Scripting" (XSS).

Lorsque vous utilisez `innerHTML` pour insérer du contenu dans le DOM, ce contenu est analysé et exécuté comme du HTML.

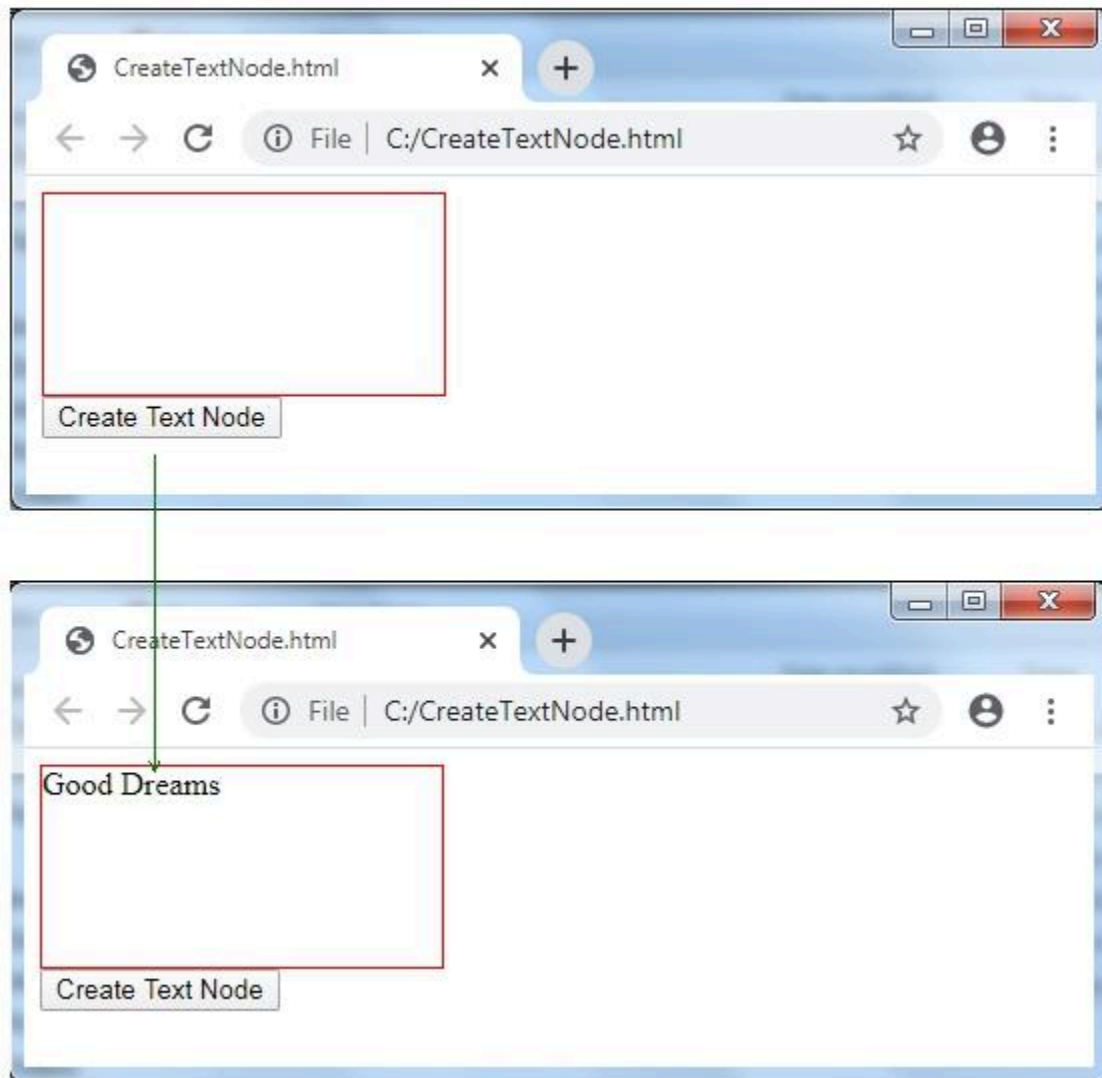
Si ce contenu contient du code JavaScript malveillant (par exemple, dans des balises `<script>` ou des attributs d'événements comme `onclick`), ce code sera exécuté dans le contexte de votre page web. Cela peut permettre à un attaquant d'injecter et d'exécuter du code arbitraire, lui donnant potentiellement accès aux données sensibles de l'utilisateur, la possibilité de modifier le contenu de la page, ou même de rediriger l'utilisateur vers des sites malveillants.

Par exemple, si un attaquant parvient à injecter

`` via `innerHTML`, ce code sera exécuté, déclenchant une alerte non désirée.

Pour éviter ces risques, il est recommandé d'utiliser des méthodes plus sûres comme `textContent` pour le texte brut, ou des API DOM comme `createElement()` et `appendChild()` pour ajouter des éléments structurés, en combinaison avec une validation et un **assainissement rigoureux** des entrées utilisateur.

# Créer un Nœud de Texte



📁 . Créez un fichier: **CreateTextNode.html**

✏️ Créer un bouton qui au clic ajoute du texte dans le carré rouge

`obj.createTextNode` : crée un nœud de texte avec le texte spécifié

`obj.appendChild` : ajoute un nœud comme dernier enfant d'un nœud

```
const divObj = document.getElementById( "div" );  
const newNode= document.createTextNode( "Good Dreams" );  
divObj.appendChild( newNode);
```

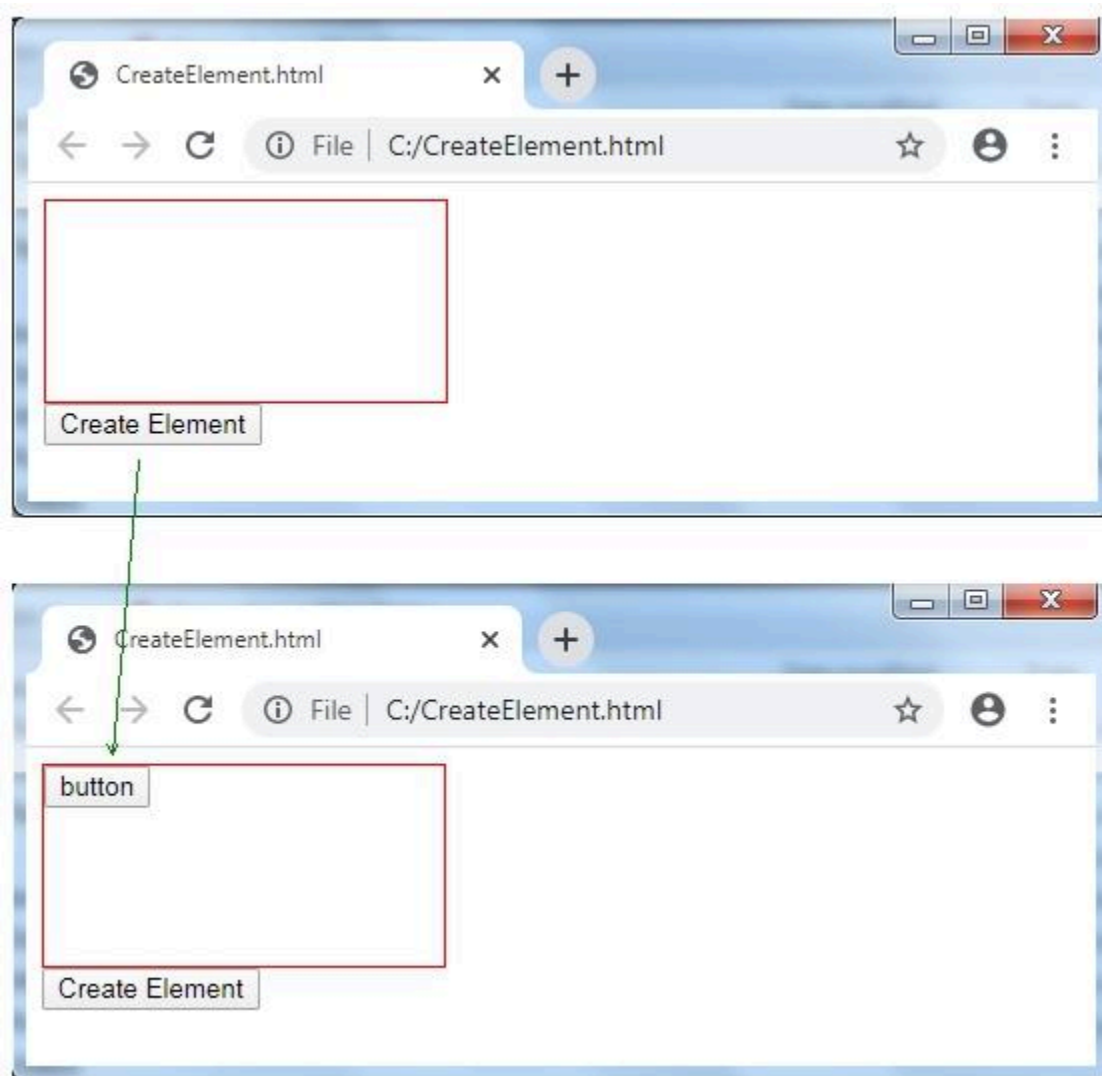
# Créer un élément

📁 Créez un fichier: **GestionElement.html**


✏️ Créer un bouton qui au clic ajoute un bouton dans le carré rouge

`obj.createElement`: crée un nœud d'élément avec le nom spécifié

`obj.appendChild`: ajoute un nœud comme dernier enfant d'un nœud




## Supprimer le Nœud

 Ajouter un bouton qui supprime le bouton à l'intérieur du carré rouge

`obj . removeChild` : supprime un nœud enfant spécifié

## Remplacer le Nœud

 Ajouter un bouton qui remplace un bouton à l'intérieur du carré rouge par une image. Un message d'erreur si pas de bouton.

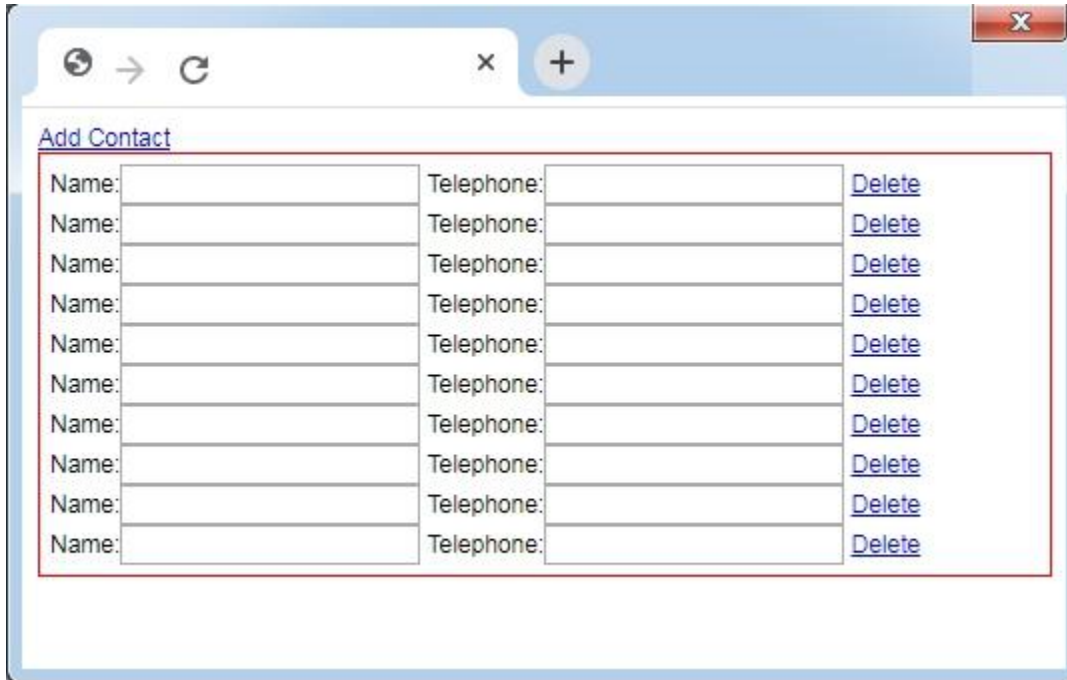
`obj . replaceChild` : remplace un nœud enfant spécifié par un autre élément

<https://picsum.photos/200/300> ( image aléatoire )

# Liste de contact

 Créer une liste de contacts :

- Ajouter un lien 'add contact" qui permet d'ajouter un contact.
- Ajouter un lien "delete" pour chaque ligne ajoutée. qui permet de supprimer la ligne correspondante.



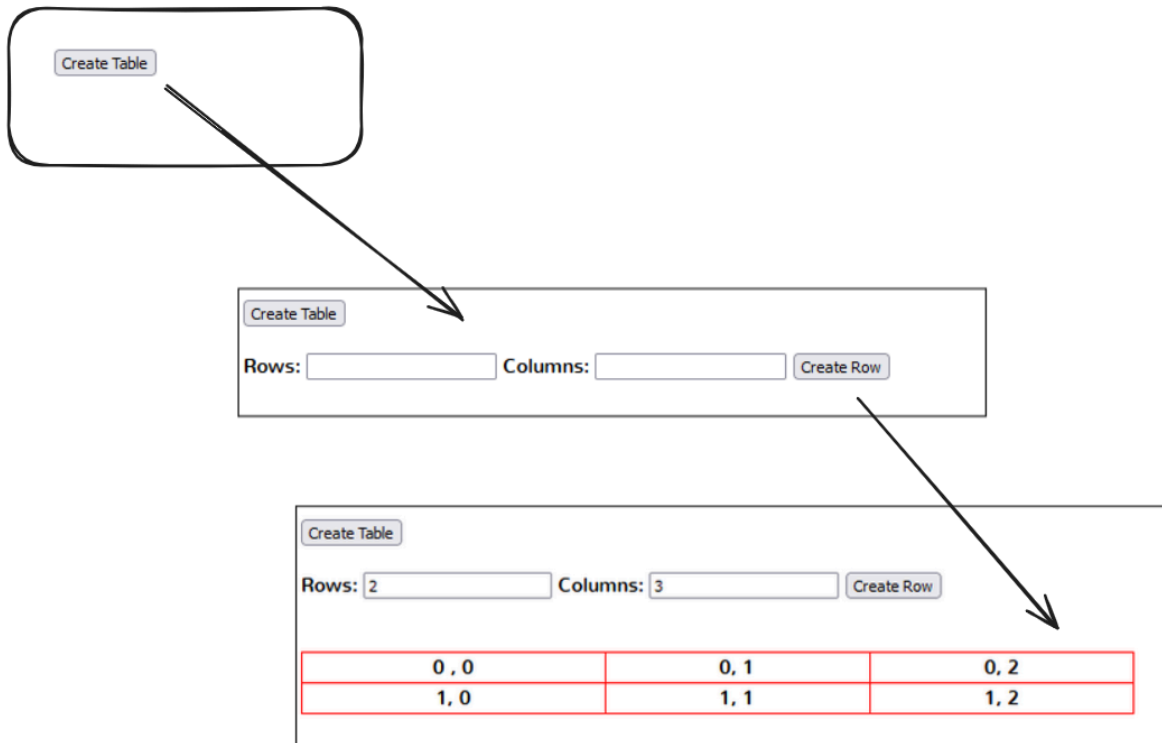
The screenshot shows a web browser window with a single tab. The address bar contains navigation icons (back, forward, refresh) and a close button. The page title is "Add Contact". The main content area contains a table with 10 rows. Each row has three columns: "Name:", "Telephone:", and a "Delete" link. The "Name:" and "Telephone:" columns contain empty text input fields. The "Delete" column contains a blue, underlined text link labeled "Delete".

Name:	Telephone:	Delete
<input type="text"/>	<input type="text"/>	<a href="#">Delete</a>
<input type="text"/>	<input type="text"/>	<a href="#">Delete</a>
<input type="text"/>	<input type="text"/>	<a href="#">Delete</a>
<input type="text"/>	<input type="text"/>	<a href="#">Delete</a>
<input type="text"/>	<input type="text"/>	<a href="#">Delete</a>
<input type="text"/>	<input type="text"/>	<a href="#">Delete</a>
<input type="text"/>	<input type="text"/>	<a href="#">Delete</a>
<input type="text"/>	<input type="text"/>	<a href="#">Delete</a>
<input type="text"/>	<input type="text"/>	<a href="#">Delete</a>
<input type="text"/>	<input type="text"/>	<a href="#">Delete</a>

# Manipulation des tables

📁 . Créez un fichier: **TableCreate.html**

## Table Créer Des Lignes Colonnes



## Instructions

### 1. Création du tableau :

- Créer un bouton "CreateTable" qui crée un tableau vide et affiche les contrôles.
- Les champs "Rows" et "Columns" permettent de spécifier le nombre de lignes et de colonnes à ajouter.

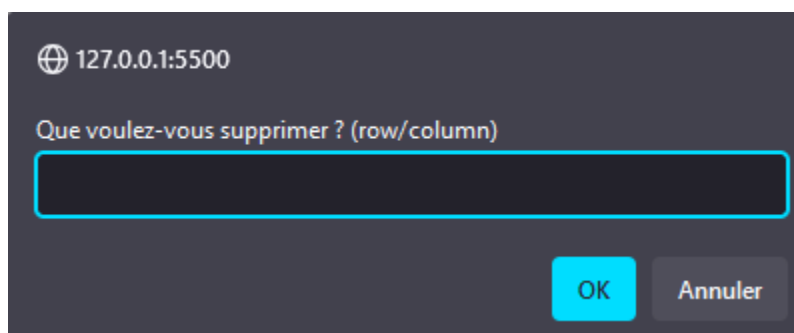
### 2. Ajout de lignes :

- Créer un bouton "Create Row" ajoute le nombre spécifié de lignes avec le nombre correct de colonnes.

- La fonction `addRow(columns)` utilise `table.insertRow()` et `row.insertCell()` pour créer les cellules.

### 3. Suppression de lignes ou de colonnes :

- Chaque cellule a un événement de clic qui appelle `handleCellClick()`.
- Cette fonction demande à l'utilisateur s'il veut supprimer une ligne ou une colonne. (prompt)
- Pour supprimer une ligne, on utilise `table.deleteRow()`.
- Pour supprimer une colonne, on parcourt toutes les lignes et on supprime la cellule à l'index spécifié. `Obj.deleteCell()`



MEMO :

<code>insertRow()</code>	insère une nouvelle ligne (<tr>) dans une <table> donnée, renvoie une référence à la nouvelle ligne.
<code>insertCell()</code>	insère un nouveau col (<td>) dans un <tr> donné, retourne une référence au nouveau col.
<code>deleteRow()</code>	supprime une rangé (<tr>)
<code>deleteCell()</code>	supprimer une cellule (<td>)

## Exercice de synthèse :

Réaliser un répertoire de contact qui permet d'afficher une liste de contact sous forme d'un tableau.

- Permettre la création, la modification et la suppression d'un contact.
- Lorsqu'un contact est sélectionné, ses informations sont affichés dans un "aside"

Voir les captures d'écrans pour vous aider.



Au chargement de la page :

#### Liste des Contacts

Nom	Téléphone	Actions
-----	-----------	---------

#### Ajouter/Modifier un Contact

<input type="text" value="Nom"/>	<input type="text" value="Téléphone"/>	<input type="text" value="Email"/>	<input type="button" value="Enregistrer"/>
----------------------------------	--	------------------------------------	--

#### Détails du Contact

## Création des users

#### Liste des Contacts

Nom	Téléphone	Actions
Régis	0612457845	<input type="button" value="Modifier"/> <input type="button" value="Supprimer"/>
John	065214521	<input type="button" value="Modifier"/> <input type="button" value="Supprimer"/>

#### Ajouter/Modifier un Contact

<input type="text" value="Nom"/>	<input type="text" value="Téléphone"/>	<input type="text" value="Email"/>	<input type="button" value="Enregistrer"/>
----------------------------------	--	------------------------------------	--

## Affichage du détails

#### Liste des Contacts

Nom	Téléphone	Actions
Régis	0612457845	<input type="button" value="Modifier"/> <input type="button" value="Supprimer"/>
John	065214521	<input type="button" value="Modifier"/> <input type="button" value="Supprimer"/>

#### Ajouter/Modifier un Contact

<input type="text" value="Nom"/>	<input type="text" value="Téléphone"/>	<input type="text" value="Email"/>	<input type="button" value="Enregistrer"/>
----------------------------------	--	------------------------------------	--

#### Détails du Contact

Nom: Régis  
Téléphone: 0612457845  
Email: regis@gmail.com