SÉLECTION ET ACCÈS AUX ÉLÉMENTS DE FORMULAIRES

DOCUMENT.GETELEMENTBYID

Cette méthode permet de sélectionner un élément **HTML** en fonction de son attribut `id`.

```
const element = document.getElementById('monElement');
console.log(element);
```

Attribut	Description
`id`	Identifiant unique de l'élément HTML

SYNTAXE

```
var element = document.getElementById("id");
```

EXEMPLES D'UTILISATION

Supposons que notre formulaire HTML contient un élément avec l'ID "input-name":

```
<input type="text" id="input-name">
```

Pour sélectionner cet élément en JavaScript:

```
var inputName = document.getElementById("input-name");
```

DOCUMENT.GETELEMENTSBYNAME

Cette méthode permet de sélectionner des éléments **HTML** possédant un certain attribut `name`.

```
var elements = document.getElementsByName('exemple');
```

Attribut	Description
`exemple`	Le nom de l'attribut à sélectionner

SYNTAXE

```
var elements = document.getElementsByName("name");
```

EXEMPLES D'UTILISATION

Supposons que notre **formulaire HTML** contient plusieurs éléments avec le nom **"input-group"**:

```
<input type="text" name="input-group">
<input type="text" name="input-group">
```

Pour sélectionner ces éléments en **JavaScript**:

```
var inputGroup = document.getElementsByName("input-group");
```

DOCUMENT.QUERYSELECTOR

Cette méthode permet de sélectionner le **premier élément HTML** correspondant à un **sélecteur CSS**.

```
// Exemple pour sélectionner l'élément ayant l'ID 'exemple'
const element = document.querySelector('#exemple');

// Exemple pour sélectionner le premier élément de la classe 'exemple'
const element = document.querySelector('.exemple');
```

SYNTAXE

```
var element = document.querySelector("selector");
```

EXEMPLES D'UTILISATION

Supposons que notre formulaire HTML contient un élément avec la classe **"input-email"**:

```
<input type="email" class="input-email">
```

Pour sélectionner cet élément en JavaScript:

```
var inputEmail = document.querySelector(".input-email");
```

DOCUMENT.QUERYSELECTORALL

Cette méthode permet de sélectionner tous les **éléments HTML** correspondant à un **sélecteur CSS**.

```
const elements = document.querySelectorAll('.maClasse');

elements.forEach(function(element) {
  console.log(element.textContent);
});
```

SYNTAXE

```
var elements = document.querySelectorAll("selector");
```

EXEMPLES D'UTILISATION

Supposons que notre **formulaire HTML** contient plusieurs éléments avec la classe "**input-checkbox**":

```
<input type="checkbox" class="input-checkbox">
<input type="checkbox" class="input-checkbox">
```

Pour sélectionner ces éléments en **JavaScript**:

```
var inputCheckboxes = document.querySelectorAll(".input-checkbox");
```

GESTION DES ÉVÉNEMENTS

ÉVÉNEMENT SUBMIT

L'événement `submit` est déclenché lorsqu'un **formulaire** est soumis. Il est souvent utilisé pour **valider** et **envoyer** les données du formulaire.

```
document.querySelector("form").addEventListener("submit", function(event) {
  event.preventDefault(); // Empêche le comportement par défaut de soumission
  // Valider et envoyer les données ici
});
```

SYNTAXE

```
form.addEventListener("submit", function(event) {
  // Code à exécuter lors de la soumission
});
```

EXEMPLES D'UTILISATION

```
document.querySelector("form").addEventListener("submit", function(event) {
  alert("Formulaire soumis");
});
```

PRÉVENTION DU RECHARGEMENT DE LA PAGE

Par défaut, la **soumission d'un formulaire** entraîne le **rechargement de la page**. Pour empêcher cela, utilisez `event.preventDefault()`.

```
form.addEventListener("submit", function(event) {
  event.preventDefault();
  // Code à exécuter lors de la soumission
});
```

ÉVÉNEMENT CHANGE

L'événement `change` est déclenché lorsque la **valeur** d'un élément de formulaire est **modifiée**.

Exemple :

```
document.querySelector("#monElement").addEventListener("change", function() {
  console.log("La valeur a été modifiée");
});
```

Éléments associés	Description
`<input>`	Événement déclenché lorsque l'utilisateur modifie la valeur
`<select>`	Événement déclenché lorsque l'utilisateur change l'option sélectionnée
`<textarea>`	Événement déclenché lorsque l'utilisateur modifie le contenu du champ

SYNTAXE

```
element.addEventListener("change", function(event) {
  // Code à exécuter lors d'un changement de valeur
});
```

EXEMPLES D'UTILISATION

```
document.querySelector("input[type='text']").addEventListener("change", function(event) {
  alert("Valeur modifiée");
});
```

UTILISATION AVEC DIFFÉRENTS TYPES D'ÉLÉMENTS

L'événement `change` peut être utilisé avec différents types d'éléments de **formulaires**, tels que les champs texte, les cases à cocher, et les listes déroulantes.

Type d'élément	Exemple d'utilisation
Champ texte	`document.querySelector("input[type='text']")`
Case à cocher	`document.querySelector("input[type='checkbox']")`
Liste déroulante	`document.querySelector("select")`

```
document.querySelector("input[type='checkbox']").addEventListener("change", function() {
  alert("Case à cocher modifiée");
});
```

ÉVÉNEMENT FOCUS

L'événement `focus` est déclenché lorsqu'un **élément de formulaire** reçoit le **focus**.

```
// Exemple d'utilisation de l'événement focus
document.getElementById("monInput").addEventListener("focus", function() {
  console.log("L'élément input a reçu le focus.");
});
```

SYNTAXE

```
element.addEventListener("focus", function() {
  // Code à exécuter lors du focus
});
```

EXEMPLES D'UTILISATION

```
document.getElementById("prenom").addEventListener("focus", function() {
  alert("Champ prénom a le focus");
});
```

CAS D'UTILISATION FRÉQUENTS

- Afficher des informations ou des conseils lors du **focus** sur un champ
- Appliquer un **style particulier** lors du focus

ÉVÉNEMENT BLUR

L'événement `blur` est déclenché lorsqu'un élément de **formulaire** perd le **focus**.

```
element.addEventListener('blur', function() {
  console.log('L\'élément a perdu le focus');
});
```

SYNTAXE

```
element.addEventListener("blur", function() {
  // Code à exécuter lors de la perte de focus
});
```

EXEMPLES D'UTILISATION

```
document.getElementById("prenom").addEventListener("blur", function() {
  alert("Champ prénom a perdu le focus");
});
```

CAS D'UTILISATION FRÉQUENTS

- Valider un champ de formulaire lors de la **perte de focus**
- Réinitialiser un **style particulier** lors de la perte de focus

MANIPULATION DES DONNÉES DE FORMULAIRES

VALEURS DES CHAMPS

Pour accéder à la valeur d'un champ de formulaire, utilisez la propriété **`value`** de l'élément.

```
document.getElementById("monInput").value; // Récupère la valeur du champ avec l'ID "monInput"
```

SYNTAXE (ÉLÉMENT.VALUE)

```
var champ = document.getElementById("monChamp");
var valeur = champ.value;
```

Note : La propriété `value` permet d'accéder à la valeur saisie par l'utilisateur dans un champ de formulaire HTML.

- Utilisable avec des éléments tels que :
  - `<input>`
  - `<select>`
  - `<textarea>`
- Syntaxe : `élément.value;`

EXEMPLES D'UTILISATION

```
var nomUtilisateur = document.getElementById("username").value;
var email = document.getElementById("email").value;
var motDePasse = document.getElementById("password").value;
```

MODIFICATION DES VALEURS

Pour modifier la **valeur** d'un champ de formulaire, affectez une nouvelle valeur à la propriété **`value`** de l'élément.

```
document.getElementById("monChamp").value = "Nouvelle valeur";
```

SYNTAXE (ÉLÉMENT.VALUE = NOUVELLEVALEUR)

```
var champ = document.getElementById("monChamp");
champ.value = "Nouvelle valeur";
```

EXEMPLES D'UTILISATION

```
document.getElementById("username").value = "NouvelUtilisateur";
document.getElementById("email").value = "nouvel@email.com";
document.getElementById("password").value = "NouveauMotDePasse";
```

RESET D'UN FORMULAIRE

Pour réinitialiser un formulaire, utilisez la méthode **`reset()`** sur l'élément **form**.

```
document.getElementById("monFormulaire").reset();
```

SYNTAXE (FORM.RESET())

```
var monFormulaire = document.getElementById("monFormulaire");
monFormulaire.reset();
```

EXEMPLES D'UTILISATION

```
document.getElementById("monFormulaire").reset();
```

VALIDATION DE FORMULAIRES

VALIDATION HTML5 NATIVE

La **validation HTML5 native** permet de définir des **contraintes** sur les champs de formulaire directement dans le code **HTML**.

Attribut	Description
required	Indique que le champ est obligatoire
pattern	Applique une expression régulière pour validation
maxlength	Limite la longueur maximale du texte entré
min et max	Pour les types number/date, contraint la valeur à un intervalle

Exemple :

```
<form>
  <label for="email">Email:</label>
  <input type="email" id="email" name="email" required>

  <label for="phone">Téléphone:</label>
  <input type="tel" id="phone" name="phone" pattern="[0-9]{10}" required>

  <label for="password">Mot de passe:</label>
  <input type="password" id="password" name="password" minlength="8" required>

  <input type="submit" value="Envoyer">
</form>
```

SYNTAXE

```
<input type="text" required pattern="[A-Za-z]{3,}" title="3 caractères min.">
```

EXEMPLES D'UTILISATION

```
<form>
  <input type="email" required placeholder="Adresse email">
  <input type="password" required minlength="8" placeholder="Mot de passe">
  <button type="submit">Envoyer</button>
</form>
```

VALIDATION JAVASCRIPT PERSONNALISÉE

La validation JavaScript personnalisée permet de créer des **règles de validation** plus complexes et adaptées à des cas spécifiques.

Techniques de validation	Description
Validation des champs obligatoires	S'assurer que tous les champs obligatoires sont remplis, sinon afficher un message d'erreur.
Validation par expressions régulières	Utiliser des **expressions régulières** pour valider les données saisies de manière plus complexe (ex: emails, formats de dates).

SYNTAXE

```
function validation() {
  // Code de validation
}

form.addEventListener("submit", (e) => {
  if (!validation()) {
    e.preventDefault();
  }
});
```

EXEMPLES D'UTILISATION

```
function validerEmail(email) {
  // RegExp pour vérifier la syntaxe de l'email
  const regExp = /^(([^<>()[\]\\.,;:\s@"]+(\.[^<>()[\]\\.,;:\s@"]+)*)|(".+"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\])
  return regExp.test(email);
}

form.addEventListener("submit", (e) => {
  const email = document.getElementById("email").value;
  if (!validerEmail(email)) {
    e.preventDefault();
    // Afficher un message d'erreur
  }
});
```

GESTION DES ERREURS

La gestion des erreurs permet d'afficher des **messages d'erreur** adaptés en cas de non-conformité des données soumises.

```
function afficherErreur(element, message) {
  const span = document.createElement("span");
  span.className = "erreur";
  span.textContent = message;
  element.parentNode.appendChild(span);
}

function supprimerErreur(element) {
  const erreur = element.parentNode.querySelector(".erreur");
  if (erreur) {
    erreur.remove();
  }
}

form.addEventListener("submit", (e) => {
```

VALIDATION CÔTÉ SERVEUR

La validation **côté serveur** permet de vérifier les données soumises une fois qu'elles ont été envoyées au serveur, afin de garantir leur **conformité**.

Avantages	Inconvénients
Sécurité accrue	Temps de réponse plus long
Prise en charge des erreurs	Charge de travail

SYNTAXE

```
fetch("/api/validation", {
  method: "POST",
  body: JSON.stringify(formData),
}).then((response) => {
  // Gérer la réponse du serveur
});
```

EXEMPLES D'UTILISATION

```
form.addEventListener("submit", (e) => {
  e.preventDefault();

  const formData = new FormData(form);
  fetch("/api/validation", {
    method: "POST",
    body: formData,
  }).then((response) => response.json())
    .then((data) => {
      if (data.success) {
        // Rediriger vers la page de succès
      } else {
        // Afficher les erreurs côté serveur
      }
    });
```

COMMUNICATION AVEC LE SERVEUR VIA AJAX

Pour communiquer avec le serveur sans recharger la page, on utilise **AJAX** (Asynchronous JavaScript And XML) qui permet de transmettre les données de manière **asynchrone**.

```
const xhr = new XMLHttpRequest();
xhr.onreadystatechange = function () {
  if (xhr.readyState == 4 && xhr.status == 200) {
    // Gérer la réponse du serveur
  }
};

xhr.open("POST", "/api/validation", true);
xhr.send(formData);
```

ENVOI DE FORMULAIRES

ENVOI CLASSIQUE

L'envoi classique d'un formulaire consiste à soumettre les données à une page du serveur définie par l'attribut **`action`** du formulaire.

SYNTAXE

```
<form action="/path/to/server/page" method="POST">
  <!-- Champs du formulaire -->
  <button type="submit">Envoyer</button>
</form>
```

EXEMPLES D'UTILISATION

```
<form action="/register" method="POST">
  <input type="text" name="username" placeholder="Nom d'utilisateur">
  <input type="password" name="password" placeholder="Mot de passe">
  <button type="submit">S'inscrire</button>
</form>
```

ENVOI VIA AJAX

L'envoi de formulaires via **AJAX** permet de soumettre les données sans recharger la page, offrant une meilleure **expérience utilisateur**.

```
// Exemple d'envoi de formulaire via AJAX
document.getElementById("form").addEventListener("submit", function(event) {
  event.preventDefault(); // Empêche le rechargement de la page

  // Récupération des données du formulaire
  var data = new FormData(event.target);

  // Envoi des données via AJAX
  var xhr = new XMLHttpRequest();
  xhr.open("POST", "https://exemple.com/api/formulaire");
  xhr.onreadystatechange = function() {
    if (xhr.readyState === 4 && xhr.status === 200) {
      console.log("Réponse reçue : " + xhr.responseText);
    }
  };
```

SYNTAXE

```
const form = document.getElementById("myForm");
form.addEventListener("submit", (event) => {
  event.preventDefault();

  const formData = new FormData(form);

  fetch("/path/to/server/page", {
    method: "POST",
    body: formData,
  })
  .then((response) => /* Gérer la réponse du serveur */)
  .catch((error) => /* Gérer l'erreur */);
});
```

EXEMPLES D'UTILISATION

```html
<form id="ajaxForm">
  <input type="text" name="username" placeholder="Nom d'utilisateur">
  <input type="password" name="password" placeholder="Mot de passe">
  <button type="submit">Se connecter</button>
</form>
```

```javascript
const ajaxForm = document.getElementById("ajaxForm");
ajaxForm.addEventListener("submit", (event) => {
  event.preventDefault();

  const formData = new FormData(ajaxForm);

  fetch("/login", {
    method: "POST",
    body: formData,
  })
  .then((response) => {
    if (response.status === 200) {
      alert("Connexion réussie");
    } else {
      alert("Erreur lors de la connexion");
```

GESTION DES RÉPONSES DU SERVEUR

Lors de l'envoi d'un **formulaire via AJAX**, il est important de gérer adéquatement les réponses du serveur pour informer l'utilisateur du résultat de l'opération.

- Vérifier le **code de statut**
- Gérer les erreurs éventuelles
- Mettre à jour l'**interface utilisateur** en conséquence

```
fetch('api/formulaire', {
  method: 'POST',
  body: JSON.stringify(data),
  headers: { 'Content-Type': 'application/json' },
})
  .then((response) => {
    if (response.ok) {
      return response.json(); // succès
    } else {
      throw new Error("Erreur lors de l'envoi du formulaire");
    }
  })
  .then((data) => {
    console.log('Succès:', data);
  })
```

EXEMPLE : TRAITEMENT D'UNE RÉPONSE JSON

```
fetch("/login", {
  method: "POST",
  body: formData,
})
.then((response) => response.json())
.then((data) => {
  if (data.success) {
    alert("Connexion réussie");
  } else {
    alert(`Erreur : ${data.message}`);
  }
})
.catch((error) => {
  alert("Erreur lors de l'envoi du formulaire");
});
```

BIBLIOTHÈQUES ET FRAMEWORKS

JQUERY ET PLUGINS POUR FORMULAIRES

jQuery est une **bibliothèque JS populaire** qui facilite la manipulation des données de formulaires, les événements et les **requêtes AJAX**.

Exemple : Sélection et manipulation d'un champ input

```
$('input[name="username"]').val("nouveauNom");
```

REACT ET GESTION DES FORMULAIRES

React est une **bibliothèque JS** pour les interfaces utilisateur qui utilise une approche **contrôlée** pour les champs de formulaire.

Exemple : Component React avec un champ input contrôlé

```
class MyForm extends React.Component {
  state = { inputValue: "" };

  handleChange = (event) => {
    this.setState({ inputValue: event.target.value });
  };

  render() {
    return (
      <input
        type="text"
        value={this.state.inputValue}
        onChange={this.handleChange}
      />
    );
```

VUE.JS ET GESTION DES FORMULAIRES

Vue.js est un **framework JS** pour les interfaces utilisateur qui utilise la directive **`v-model`** pour la **liaison bidirectionnelle** des données de formulaires.

Exemple : Component Vue.js avec champs de formulaire

```
<template>
  <input type="text" v-model="username" />
</template>

<script>
export default {
  data() {
    return { username: "" };
  },
};
</script>
```

AUTRES BIBLIOTHÈQUES ET FRAMEWORKS

Il existe de nombreux autres outils pour gérer les **formulaires** en JavaScript, tels que **Angular**, **Ember.js** et **Backbone.js**. Chacun a sa propre approche pour manipuler les données de formulaires, les validations et les envois.