Un formulaire HTML est un élément clé du langage de balisage hypertexte (HTML) utilisé dans le développement web. Il permet aux utilisateurs d'interagir avec un site web en saisissant des données et en les envoyant au serveur pour traitement. Les formulaires sont couramment utilisés pour collecter des informations telles que des commentaires, des coordonnées, des informations de connexion, des sélections d'options, et bien plus encore.

<form> : C'est l'élément de base qui englobe tous les autres éléments du formulaire. Il définit un espace où les utilisateurs peuvent saisir et soumettre des données. Les attributs importants incluent action (l'URL où les données seront envoyées), method (la méthode HTTP utilisée pour l'envoi, généralement GET ou POST), et d'autres attributs pour la personnalisation.

Cette balise contient des attributs :

- **action** : Cet attribut spécifie l'URL ou le point de terminaison vers lequel les données du formulaire seront envoyées lorsqu'il est soumis. Par exemple :

```
<form action="traitement.php" method="POST">
```

- **method** : Cet attribut indique la méthode HTTP à utiliser pour envoyer les données du formulaire. Les valeurs typiques sont "GET" et "POST". "GET" ajoute les données au niveau de l'URL, tandis que "POST" les envoie dans le corps de la requête. Par exemple :

```
<form action="traitement.php" method="POST">
```

- **target** : Cet attribut détermine où la réponse du serveur après la soumission du formulaire sera affichée. Les valeurs typiques sont "_self" pour la même fenêtre et "_blank" pour une nouvelle fenêtre ou un nouvel onglet. Par exemple :

```
<form action="traitement.php" method="POST" target="_blank">
```

- **enctype** : Cet attribut indique comment les données du formulaire sont encodées lorsqu'elles sont envoyées. La valeur par défaut est généralement "application/x-www-form-urlencoded", mais vous pouvez également utiliser "multipart/form-data" pour télécharger des fichiers. Par exemple :

```
<form action="traitement.php" method="POST" enctype="multipart/form-
data">
```

- **autocomplete** : Cet attribut contrôle la fonctionnalité de remplissage automatique (auto-complétion) du navigateur pour les champs de saisie du formulaire. Les valeurs courantes sont "on" (activé) et "off" (désactivé). Par exemple :

```
<form action="traitement.php" method="POST" autocomplete="off">
```

- **name** : Cet attribut donne un nom au formulaire, ce qui peut être utile pour le cibler avec du CSS ou JavaScript. Par exemple :

```
<form action="traitement.php" method="POST" name="monFormulaire">
```

- **novalidate** : Cet attribut désactive la validation HTML5 native du navigateur. Utile si vous prévoyez de valider le formulaire côté serveur ou avec JavaScript. Par exemple:

```
<form action="traitement.php" method="POST" novalidate>
```

- **onsubmit** : Cet attribut permet d'ajouter une fonction JavaScript qui sera exécutée lorsque le formulaire est soumis. Par exemple :

```
 <form action="traitement.php" method="POST" onsubmit="return
validerFormulaire()">
```

- **class et id** : Ces attributs permettent de définir des classes ou des identifiants CSS pour cibler le formulaire avec du style ou du JavaScript.
  Par exemple :

```
<form action="traitement.php" method="POST" class="formulaire"
id="monFormulaire">
```

Le formulaire html doit être rédigé dans la balise < form> < /form>

Exemple :

```
<p>avant le formulaire</p>

    <form method="get" action="">

        <p>à l'intérieur du formulaire</p>

    </form>

<p>après le formulaire</p>
```

Introduction au formulaire

La balise <input>

La balise HTML <input> est une balise autonome que nous rencontrerons à plusieurs reprises au cours de ce chapitre. Fréquemment, c'est la valeur de son attribut type qui variera en fonction du type de champ que nous souhaitons inclure, que ce soit un champ de texte monoligne, une adresse e-mail, une date, et ainsi de suite.

La balise <input type="text">

Cette balise permet d'ajouter une zone de texte monoligne. Cette balise a un attribut name pour avoir un nom à cette zone de texte.

```
<form method="get" action="">
```

```
    <input type="text" name="nom">
```

```
</form>
```

D'autres attributs peuvent être appliqués à cette balise pour adapter son comportement :

- Pour augmenter la taille du champ, on peut utiliser l'attribut size.
- On peut restreindre le nombre de caractères pouvant être saisis avec l'attribut maxlength.
- pré-remplir le champ avec une valeur par défaut, l'attribut value est utile.
- Pour donner une indication sur le contenu du champ, on peut recourir à l'attribut placeholder. Cette indication s'effacera dès que l'utilisateur cliquera dans le champ.

Comment le visiteur connait qu'il doit renseigner son nom sans avoir une indication dans la zone du texte ?

La balise <label>

```
<form method="get" action="">
```

```
  <label for="nom">Votre nom</label>  : <input type="text"
```

```
name="nom" id="nom" >
```

```
</form>
```

Nous avons ajouté une étiquette (<label>) pour la zone de texte pour fournir une description du champ.

- L'attribut for de l'étiquette est associé à l'attribut id de l'input, ce qui permet aux utilisateurs de cliquer sur l'étiquette pour sélectionner le champ. Il est donc unique pour cet input
- L'attribut name est utilisé pour identifier le champ lors de l'envoi du formulaire.

Exemple :

```
<form method="get" action="">
  <p>
    <label for="nom">Votre nom :</label>
    <input type="text" name="nom" id="nom" placeholder="Ecrivez votre nom" size="20" maxlength="10">
  </p>
</form>
```

Dans cet exemple, la zone de saisie est accompagnée d'une indication pour comprendre ce qui doit être saisi. Le champ a une longueur de 20 caractères, cependant, il n'est possible d'insérer qu'un maximum de 10 caractères à l'intérieur.

Vous pouvez tester et modifier en ligne ce code sur codepen. :

La balise <textarea>

La balise <textarea> est un élément HTML utilisé pour créer une zone de texte multi-lignes dans une page web. Contrairement à la balise <input> qui est utilisée pour les champs de texte simples, la balise <textarea> permet aux utilisateurs de saisir du texte sur plusieurs lignes.

Exemple :

```
<label for="description">Description : </label> <br>
```

```
<textarea rows="4" cols="50">
```

```
   C'est ici que vous pouvez saisir du texte sur plusieurs lignes.
```

```
</textarea>
```

Dans cet exemple, l'attribut rows définit le nombre de lignes visibles dans la zone de texte, et l'attribut cols définit le nombre de colonnes (largeur) de la zone de texte. Les utilisateurs peuvent saisir du texte dans cette zone en utilisant plusieurs lignes et en ajustant automatiquement la taille de la zone de texte en fonction du contenu.

Vous pouvez tester et modifier en ligne ce code en utilisant codepen.

La balise <input type= "button">

Les éléments <input type="button"> ne possèdent pas de comportement particulier. Pour qu'un bouton <input type="button"> puisse avoir un effet, il est nécessaire d'écrire quelques lignes JavaScript.

```
<input type="button" value="Cliquer ici" />
```

L'attribut value indique le texte qui s'affiche sur le bouton.

On peut avoir d'autres attributs comme disabled pour désactiver un bouton.

```
<input type="button" value="Désactivé" disabled />
```

Vous pouvez tester et modifier en ligne ce code : https://codepen.io/Formation-DCLIC/pen/mdaVVjG

La balise <input type= "checkbox">

L'élément HTML <input> avec l'attribut type="checkbox" est utilisé pour créer des cases à cocher dans les formulaires sur les pages web. Une case à cocher permet aux utilisateurs de sélectionner ou de désélectionner une option spécifique.

```
<input type="checkbox" id="newsletter" name="subscribe" value="yes">
```

```
<label for="newsletter">S'abonner à la newsletter</label>
```

Dans cet exemple, une case à cocher est créée avec l'identifiant (id) "newsletter", le nom (name) "subscribe" et la valeur (value) "yes". La balise <label> est utilisée pour afficher un libellé associé à la case à cocher. Le libellé est lié à la case à cocher en utilisant l'attribut for qui pointe vers l'identifiant de la case à cocher.

Lorsqu'un utilisateur coche cette case, le navigateur envoie les données du formulaire avec le nom "subscribe" et la valeur "yes" au serveur lors de la soumission du formulaire. Si la case n'est pas cochée, aucune donnée n'est envoyée pour cet élément.

Si on voulait que le ckeckbox est coché par défaut, on ajoute l'attribut checked

```
<input type="checkbox" id="newsletter" name="subscribe" value="yes"
```

```
checked>
```

```
<label for="newsletter">S'abonner à la newsletter</label>
```

Vous pouvez tester et modifier en ligne ce code en utilisant codepen.

La balise <input type="number">

L'élément HTML <input> de type "number" est utilisé pour créer un champ de saisie numérique

```
<label for="quantity">Quantité :</label>
```

```
<input type="number"
       id="quantity"
       name="quantity"
       min="0"
       max="100"
       step="1"
       value="0">
```

- **type="number"** : Définit le type d'entrée comme un champ de saisie numérique.
- **id** : Fournit un identifiant unique pour l'élément, qui peut être associé à une balise <label> pour une meilleure expérience utilisateur.
- **name** : Définit le nom de l'élément, utilisé pour identifier le champ lors de la soumission du formulaire.
- **min** : Définit la valeur minimale autorisée pour la saisie numérique.
- **max** : Définit la valeur maximale autorisée pour la saisie numérique.
- **step** : Définit l'incrément (ou la valeur par saut) pour la modification de la valeur numérique lorsque les flèches sont utilisées ou lorsque l'utilisateur fait des ajustements.
- **value** : Définit la valeur par défaut affichée dans le champ de saisie.
- **<label for="quantity">** : Lie le libellé à l'élément de saisie en utilisant l'attribut for qui correspond à l'identifiant de l'élément.

Vous pouvez tester et modifier en ligne ce code : https://codepen.io/Formation-DCLIC/pen/BavjjEe

La balise <input type="number">

```
<label for="email">Adresse e-mail :</label>
```

```
<input type="email"
```

```
       id="email"
```

```
       name="email"
```

```
       required
```

```
       placeholder="votre@email.com">
```

Explications des attributs :

- **type="email"** : Définit le type d'entrée comme un champ de saisie d'adresse e-mail.
- **id** : Fournit un identifiant unique pour l'élément, qui peut être associé à une balise <label> pour une meilleure expérience utilisateur.
- **name** : Définit le nom de l'élément, utilisé pour identifier le champ lors de la soumission du formulaire.
- **required** : Indique que le champ de saisie est obligatoire et ne peut pas être laissé vide .
- **placeholder** : Affiche un exemple ou une indication dans le champ de saisie avant que l'utilisateur n'y entre une valeur.
- **<label for="email">** : Lie le libellé à l'élément de saisie en utilisant l'attribut for qui correspond à l'identifiant de l'élément.

D'autres attributs supplémentaires:

- **mutiple**
- **pattern**

Vous pouvez tester et modifier en ligne ce code : https://codepen.io/Formation-DCLIC/pen/GRPoobz

La balise <input type="password">

```
<label for="password">Mot de passe :</label>
```

```
<input type="password"
```

```
       id="password"
```

```
       name="password"
```

```
       required
```

```
       minlength="8"
```

```
       maxlength="20"
```

```
       placeholder="Entrez votre mot de passe">
```

- **id** : Fournit un identifiant unique pour l'élément, qui peut être associé à une balise <label> pour une meilleure expérience utilisateur.
- **name** : Définit le nom de l'élément, utilisé pour identifier le champ lors de la soumission du formulaire.
- **required** : Indique que le champ de saisie est obligatoire et ne peut pas être laissé vide.
- **minlength** : Définit le nombre minimum de caractères requis pour le mot de passe.
- **maxlength** : Définit le nombre maximum de caractères autorisés pour le mot de passe.
- **placeholder** : Affiche une indication de ce qui est attendu dans le champ de saisie.
- **<label for="password">** : Lie le libellé à l'élément de saisie en utilisant l'attribut for qui correspond à l'identifiant de l'élément.

Vous pouvez tester et modifier en ligne ce code en utilisant codepen.

La balise <input type="radio">

Les boutons radio permettent aux utilisateurs de sélectionner une option parmi plusieurs options mutuellement exclusives.

```
<input type="radio"
       id="option1"
       name="options"
       value="option1"
       checked>
<label for="option1">Option 1</label>

<input type="radio"
       id="option2"
       name="options"
       value="option2">
<label for="option2">Option 2</label>
```

- **id** : Fournit un identifiant unique pour l'élément, qui peut être associé à une balise <label> pour une meilleure expérience utilisateur.

- **name** : Définit le nom du groupe de boutons radio. Les boutons radio ayant le même nom font partie du même groupe et sont mutuellement exclusifs.
- **value** : Définit la valeur associée au bouton radio lorsqu'il est sélectionné. Cette valeur est envoyée au serveur lors de la soumission du formulaire.
- **checked** : Indique que le bouton radio est initialement sélectionné lorsque la page est chargée.

Cet exemple montre comment créer deux boutons radio, "Option 1" et "Option 2", faisant partie du même groupe "options". L'élément radio avec l'attribut checked est initialement sélectionné lorsque la page est chargée.

Les boutons radio sont souvent utilisés pour les choix exclusifs, comme sélectionner le sexe (masculin/féminin), le mode de paiement, ou d'autres options où l'utilisateur ne peut choisir qu'une seule parmi plusieurs.

Vous pouvez tester et modifier en ligne ce code en utilisant codepen.

La balise <input type="number">

L'élément HTML <input> de type "range" est utilisé pour créer un curseur glissant (slider) dans un formulaire sur une page web. Les curseurs glissants permettent aux utilisateurs de sélectionner une valeur numérique dans une plage définie

```
<label for="volume">Volume :</label>
```

```
<input type="range"
```

```
       id="volume"
```

```
       name="volume"
```

```
       min="0"
```

```
       max="100"
```

```
       value="50">
```

- **min** : Définit la valeur minimale autorisée pour le curseur glissant.
- **max** : Définit la valeur maximale autorisée pour le curseur glissant.

Vous pouvez tester et modifier en ligne ce code en utilisant codepen.

La balise <input type="date">

```
<label for="birthdate">Date de naissance :</label>
```

```
<input type="date"
       id="birthdate"
       name="birthdate"
       min="1900-01-01"
       max="2023-08-25"
       value="2000-01-01">
```

D'autres balises de date. Il faut vérifier que le navigateur autorise ce genre de date

- **<input type="time">** pour l'heure ;
- **<input type="week">** pour la semaine ;
- **<input type="month">** pour le mois ;
- **<input type="datetime">** pour la date et l'heure (avec gestion du décalage horaire) ;
- **<input type="datetime-local">** pour la date et l'heure (sans gestion du décalage horaire).

Vous pouvez tester et modifier en ligne ce code en utilisant codepen.

La balise <input type="url">

L'élément HTML <input> de type "url" est utilisé pour créer un champ de saisie d'URL (lien) dans un formulaire sur une page web. Cela permet aux utilisateurs de saisir des adresses web valides

```
<label for="website">Site Web :</label>
```

```
<input type="url"
```

```
       id="website"
```

```
       name="website"
```

```
       placeholder="https://www.example.com"
```

```
       required>
```

Vous pouvez tester et modifier en ligne ce code en utilisant codepen.

La balise <input type="submit">

Cette balise est utilisée pour créer un bouton de soumission dans un formulaire sur une page web. Ce bouton permet aux utilisateurs de soumettre les données saisies dans le formulaire au serveur pour traitement.

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

La balise <input type="reset">

Cette balise est utilisée pour créer un bouton de réinitialisation dans un formulaire sur une page web. Ce bouton permet aux utilisateurs de rétablir les valeurs du formulaire aux valeurs par défaut.

```
<input type="reset" value="Réinitialiser">
```

Vous pouvez tester et modifier en ligne ce code en utilisant codepen.

La balise <select>

La balise <select></select> est employée pour délimiter le commencement et la fin d'une liste déroulante. L'attribut "name" est inclus dans la balise pour attribuer un nom à cette liste.

Ensuite, à l'intérieur de la balise <select></select>, plusieurs balises <option></option> (une pour chaque choix possible) sont disposées. Chacune de ces balises <option></option> est dotée d'un attribut "value" qui permet d'identifier le choix effectué par le visiteur.

```
<label for="cars">Choisissez une voiture :</label>
```

```
<select id="cars" name="cars">
```

```
  <option value="volvo">Volvo</option>
```

```
  <option value="saab">Saab</option>
```

```
  <option value="mercedes">Mercedes</option>
```

```
  <option value="audi">Audi</option>
```

```
</select>
```

Dans cet exemple, nous avons une liste déroulante de voitures. La balise <select></select> englobe toute la liste, tandis que chaque voiture est représentée par une balise <option></option>. Chaque balise <option></option> a un attribut "value" qui identifie la valeur associée à cette option. L'attribut "name" de la balise <select></select> permet d'identifier la liste dans le formulaire lors de la soumission.

Vous pouvez tester et modifier en ligne ce code en utilisant codepen.

Quelle balise HTML est utilisée pour créer un formulaire ?

- <label>
- <input>
- <select>
- <form>

Les formulaires HTML

Les èvènements

Les événements en HTML peuvent être représentés sous forme d'attributs en ajoutant des attributs spécifiques à un élément HTML.
Ces attributs sont appelés "attributs d'événement" et sont utilisés pour spécifier les actions à exécuter lorsque l'événement correspondant se produit sur l'élément.
Voici comment cela fonctionne et quelques-uns des attributs d'événement les plus couramment utilisés :

Syntaxe générale :
Pour ajouter un attribut d'événement à un élément HTML, vous utilisez la syntaxe suivante :

```
<element attribut="codeJavaScript">
```

- element : L'élément HTML sur lequel vous voulez définir l'attribut d'événement.
- attribut : Le nom de l'attribut d'événement (par exemple, onclick, onmouseover, etc.).
- codeJavaScript : Le code JavaScript à exécuter lorsque l'événement se produit.

Liste des d'attributs d'événement les plus utilisés:

- onclick : Définit une action à exécuter lorsque l'utilisateur clique sur l'élément.

- onmouseover : Définit une action à exécuter lorsque le curseur de la souris survole l'élément.
- onmouseout : Définit une action à exécuter lorsque le curseur de la souris quitte l'élément.
- onkeydown : Définit une action à exécuter lorsque l'utilisateur appuie une touche du clavier.
- onkeyup : Définit une action à exécuter lorsque l'utilisateur relâche une touche du clavier.
- onchange : Définit une action à exécuter lorsque la valeur d'un élément de formulaire change.
- onsubmit : Définit une action à exécuter lorsque le formulaire est soumis.
- onload : Définit une action à exécuter lorsque la page web se charge.

Utilisation :
Vous pouvez ajouter ces attributs d'événement directement aux éléments HTML dans votre code.
Lorsque l'événement se produit, le code JavaScript spécifié dans l'attribut sera exécuté.

Exemple :
Voici un exemple d'utilisation de l'attribut `onclick` pour exécuter une fonction JavaScript lorsque l'utilisateur clique sur un bouton :

```
<button onclick="afficherMessage();">Cliquez-moi</button>

<script>

    function afficherMessage() {

        alert("Bouton cliqué !");

    }

</script>
```

En utilisant des attributs d'événement, vous pouvez associer directement des actions JavaScript aux interactions de l'utilisateur avec les éléments de la page.

Cela rend les interactions simples à mettre en œuvre, mais pour des scénarios plus complexes, l'utilisation de `addEventListener` et JavaScript externe peut être préférable.
La méthode `addEventListener` de l'objet Document sera présenté dans le niveau intermidiaire.

La liste des èvènements

- **submit**

L'événement submit se produit lorsqu'un formulaire est soumis, généralement en cliquant sur un bouton "Envoyer". Il peut être utilisé pour valider les données du formulaire avant leur soumission.

Exemple : Lorsque l'utilisateur clique sur le bouton "Envoyer", l'événement submit est déclenché. La fonction validerFormulaire() vérifie si le champ de texte est vide. Si c'est le cas, elle empêche l'envoi du formulaire en retournant false.

```
<!DOCTYPE html>

<html>

<head>

    <title>Exemple d'événement submit</title>

</head>

<body>

    <form name="myForm" onsubmit="return validerFormulaire();">

        <input type="text" name="username" placeholder="Nom
d'utilisateur">

        <button type="submit">Envoyer</button>

    </form>



    <script>

        function validerFormulaire() {

            const username = document.myForm.username.value;
```

```
            if (username === "") {
                alert("Veuillez entrer un nom d'utilisateur.");
                return false; // Empêcher l'envoi du formulaire
            }
            return true; // Envoyer le formulaire
        }
    </script>
</body>
</html>
```

- **reset**

L'événement `reset` se produit lorsque le bouton "Réinitialiser" d'un formulaire est cliqué. Il peut être utilisé pour effectuer des actions spécifiques lors de la réinitialisation du formulaire.

Exemple : Lorsque l'utilisateur clique sur le bouton "Réinitialiser", l'événement `reset` est déclenché. La fonction `afficherMessageReset()` affiche simplement un message dans une boîte de dialogue "alert".

```
<!DOCTYPE html>
<html>
<head>
    <title>Exemple d'événement reset</title>
</head>
```

```
<body>

    <form name="myForm" onreset="afficherMessageReset();">

        <input type="text" name="username" placeholder="Nom
d'utilisateur">

        <button type="reset">Réinitialiser</button>

    </form>


    <script>

        function afficherMessageReset() {

            alert("Formulaire réinitialisé !");

        }

    </script>

</body>

</html>
```

- **change**

L'événement change se produit lorsque la valeur d'un élément de formulaire, comme une liste déroulante, est modifiée et que l'élément perd le focus.

Exemple : Lorsque l'utilisateur sélectionne une option différente dans une liste déroulante et déplace le focus ailleurs, l'événement change est déclenché. La fonction afficherSelection() affiche la sélection dans une boîte de dialogue "alert".

```
<!DOCTYPE html>

<html>

<head>

    <title>Exemple d'événement change</title>

</head>

<body>

    <form name="myForm">

        <select name="ville" onchange="afficherSelection();">

            <option value="paris">Paris</option>

            <option value="newyork">New York</option>

            <option value="london">Londres</option>

        </select>

    </form>


    <script>

        function afficherSelection() {

            const selectedCity = document.myForm.ville.value;
```

```
            alert(`Ville sélectionnée : ${selectedCity}`);

        }

    </script>

</body>

</html>
```

- **input**

L'événement input se produit à chaque fois que la valeur d'un élément de formulaire change, généralement en réponse à une saisie de l'utilisateur.

Exemple : Lorsque l'utilisateur tape du texte dans un champ de texte, l'événement input est déclenché. La fonction afficherSaisie() affiche la saisie en cours dans une boîte de dialogue "alert".

```
<!DOCTYPE html>

<html>

<head>

    <title>Exemple d'événement input</title>

</head>

<body>

    <form name="myForm">

        <input type="text" name="message" placeholder="Entrez un
message" oninput="afficherSaisie();">
```

```
    </form>


    <script>

        function afficherSaisie() {

            const message = document.myForm.message.value;

            alert(`Saisie en cours : ${message}`);

        }

    </script>

</body>

</html>
```

- **focus et blur**

L'événement focus se produit lorsque l'élément reçoit le focus, c'est-à-dire lorsque l'utilisateur clique ou navigue jusqu'à cet élément. L'événement blur se produit lorsque l'élément perd le focus.

Exemple : Lorsque l'utilisateur clique dans un champ de texte, l'événement focus est déclenché. Lorsqu'il clique ensuite ailleurs, l'événement blur est déclenché. Les fonctions afficherFocus() et afficherBlur() montrent un message dans une boîte de dialogue "alert" en fonction des événements.

```
<!DOCTYPE html>

<html>

<head>
```

```
    <title>Exemple d'événements focus et blur</title>

</head>

<body>

    <form name="myForm">

        <input type="text" name="username" placeholder="Nom
d'utilisateur"

               onfocus="afficherFocus();" onblur="afficherBlur();">

    </form>


    <script>

        function afficherFocus() {

            alert("Champ de texte en focus !");

        }


        function afficherBlur() {

            alert("Champ de texte hors focus !");

        }

    </script>
```

```
</body>

</html>
```

- **keydown, keyup et keypress**

Les événements keydown, keyup et keypress se produisent lorsqu'une touche du clavier est enfoncée, relâchée ou pressée (combinaison de keydown suivi de keyup).

Exemple : Lorsque l'utilisateur appuie sur une touche, l'événement keydown est déclenché. Lorsqu'il relâche la touche, l'événement keyup est déclenché. La fonction afficherTouche() affiche la touche appuyée lors de l'événement keypress.

```
<!DOCTYPE html>

<html>

<head>

    <title>Exemple d'événements keydown, keyup et keypress</title>

</head>

<body>

    <form name="myForm">

        <input type="text" name="message" placeholder="Appuyez sur
une touche"

               onkeydown="afficherTouche(event, 'down');"
onkeyup="afficherTouche(event, 'up');"

               onkeypress="afficherTouche(event, 'press');">
```

```
    </form>
```

```
    <script>
```

```
        function afficherTouche(event, type) {
```

```
            const touche = event.key;
```

```
            alert(`Touche ${touche} ${type}`);
```

```
        }
```

```
    </script>
```

```
</body>
```

```
</html>
```

- **mousedown, mouseup et click**

Les événements mousedown, mouseup et click se produisent lorsqu'un bouton de la souris est enfoncé, relâché et cliqué respectivement.

Exemple : Lorsque l'utilisateur clique sur un élément, l'événement mousedown est déclenché. Lorsqu'il relâche le bouton de la souris, l'événement mouseup est déclenché. Si l'utilisateur a enfoncé et relâché le bouton rapidement, l'événement click est déclenché. Les fonctions afficherClic() et afficherRelachement() montrent des messages dans des boîtes de dialogue "alert" en fonction des événements.

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
    <title>Exemple d'événements mousedown, mouseup et click</title>
</head>
<body>
    <form name="myForm">
        <button type="button" onmousedown="afficherClic();"
onmouseup="afficherRelachement();">Cliquez ici</button>
    </form>

    <script>
        function afficherClic() {
            alert("Bouton de la souris enfoncé !");
        }

        function afficherRelachement() {
            alert("Bouton de la souris relâché !");
        }
    </script>
</body>
```

```
</html>
```

- **contextmenu**

L'événement contextmenu se produit lorsque l'utilisateur tente d'ouvrir le menu contextuel (clic droit) sur un élément.

Exemple : Lorsque l'utilisateur clique avec le bouton droit sur un élément, l'événement contextmenu est déclenché. La fonction afficherMenuContextuel() affiche un message dans une boîte de dialogue "alert".

```
<!DOCTYPE html>

<html>

<head>

    <title>Exemple d'événement contextmenu</title>

</head>

<body>

    <form name="myForm">

        <div oncontextmenu="afficherMenuContextuel();">Cliquez avec
le bouton droit ici</div>

    </form>



    <script>

        function afficherMenuContextuel() {
```

```
            alert("Menu contextuel affiché !");

        }

    </script>

</body>

</html>
```

- **select**

L'événement select se produit lorsque l'utilisateur sélectionne du texte dans un champ de texte ou un élément <textarea>.

Exemple : Lorsque l'utilisateur sélectionne du texte en cliquant et en faisant glisser le curseur dans un champ de texte, l'événement select est déclenché. La fonction afficherSelection() affiche la sélection dans une boîte de dialogue "alert".

```
<!DOCTYPE html>

<html>

<head>

    <title>Exemple d'événement select</title>

</head>

<body>

    <form name="myForm">

        <input type="text" name="selection" value="Sélectionnez-moi"

onselect="afficherSelection();">
```

```
    </form>

    <script>
        function afficherSelection() {
            const selectedText = window.getSelection().toString();
            alert(`Texte sélectionné : ${selectedText}`);
        }
    </script>
</body>
</html>
```

- **invalid**

L'événement invalid se produit lorsqu'un élément de formulaire échoue à la validation native du navigateur lors de la soumission du formulaire.

Exemple : Lorsque l'utilisateur tente de soumettre un formulaire avec des données invalides, l'événement invalid peut être déclenché. La fonction afficherMessageErreur() personnalise le message d'erreur affiché dans une boîte de dialogue "alert".

```
<!DOCTYPE html>
<html>
<head>
    <title>Exemple d'événement invalid</title>
```

```
</head>

<body>

    <form name="myForm" oninvalid="afficherMessageErreur();">

        <input type="email" name="email" required

placeholder="Adresse e-mail">

        <button type="submit">Envoyer</button>

    </form>


    <script>

        function afficherMessageErreur() {

            alert("Veuillez entrer une adresse e-mail valide.");

        }

    </script>

</body>

</html>
```

Les évènements

L'introduction aux objets en JavaScript est un concept essentiel pour comprendre comment le langage fonctionne et comment interagir avec les données et les fonctionnalités dans un environnement web. Les objets fournissent un moyen puissant de représenter et d'organiser des données complexes, ainsi que de créer des interactions et des fonctionnalités interactives sur les pages web. Voici une introduction de base aux objets en JavaScript :

Qu'est-ce qu'un Objet en JavaScript ?

En JavaScript, un objet est une structure de données qui regroupe des valeurs (propriétés) et des fonctions (méthodes) associées. Les propriétés sont des variables qui contiennent des données, et les méthodes sont des fonctions qui effectuent des actions ou des opérations liées à l'objet. Les objets sont utilisés pour modéliser des entités du monde réel et pour encapsuler des fonctionnalités connexes.

Les objets préconstruits en JavaScript sont des objets intégrés au langage et qui offrent des fonctionnalités essentielles pour manipuler différents types de données et effectuer des opérations courantes. Ces objets sont disponibles dès le chargement de JavaScript dans un navigateur ou dans un environnement d'exécution.

Nous allons introduire les objets les plus utilisés dans Javascript.

- **Object :**
  Tout Objet Javascipt hérite de cette classe Object

- **Array :**
  Cet objet est utilisé pour stocker des collections ordonnées d'éléments. Il propose une gamme de méthodes pour ajouter, supprimer, parcourir et manipuler des éléments dans un tableau.

- **Boolean**
  L'objet Boolean est associé aux valeurs booléennes true (vrai) et false (faux). Bien que les valeurs booléennes soient généralement utilisées sans l'objet Boolean, cet objet offre quelques méthodes utiles pour travailler avec ces valeurs.

- **Date :**
  L'objet Date est utilisé pour travailler avec les dates et les heures. Il offre plusieurs méthodes pour créer, manipuler et formater les dates et les heures.

- **String**
  L'objet String est utilisé pour travailler avec des chaînes de caractères.

- **Math :**
  Un objet Javascript posséde des méthodes standards et aussi trigométrie

- **Document**
  L'objet document fournit une interface pour interagir avec les éléments HTML de la page, modifier son contenu et répondre aux événements utilisateur.

Introduction aux objets JavaScript

Les tableaux peuvent être créés en utilisant la notation littérale entre crochets [ ] ou en utilisant le constructeur Array() :

```
const fruits = ["pomme", "banane", "orange"];
```

```
const numbers = new Array(1, 2, 3, 4, 5);
```

L'objet Array a comme propriété : length

L'objet Array a comme méthodes :

- La méthode push(element) : number

- Cette méthode permet d'ajouter un élément à la fin du tableau et renvoie la nouvelle taille du tableau

```
const fruits = ["pomme", "banane", "orange"];
```

```
fruits.push("ananas");
```

Après appeler la méthode push, le tableau fruits sera changé et le résultat est :
["pomme", "banane", "orange","ananas"];

- La méthode pop()

- Cette méthode permet de supprimer le dernier élément du tableau et le renvoie. Si le tableau est vide la méthode retourne undefined et le tableau ne se change pas.

```
const fruits = ["pomme", "banane", "orange"];
```

```
console.log(fruits.pop()); // orange
```

```
console.log(fruits); // [ 'pomme', 'banane' ]
```

- ### La méthode shift()

- Cette méthode permet de supprimer le premier élément du tableau et le renvoie. Si le tableau est vide la méthode retourne undefined et le tableau ne se change pas.

```
const fruits = ["pomme", "banane", "orange"];
```

```
console.log(fruits.shift()); // pomme
```

```
console.log(fruits); // [ 'banane', 'orange' ]
```

- ### La méthode unshift(element)

- Cette méthode permet d'ajouter un élément au début du tableau. Cet élément peut être un seul élément ou un tableau.

```
const fruits = ["pomme", "banane", "orange"];
```

```
console.log(fruits.shift()); // pomme
```

```
console.log(fruits); // [ 'banane', 'orange' ]
```

Dans cet exemple, on ajoute un tableau au début de notre tableau fruits.

Le résultat est un tableau fruits imbriqué, qui contient au début le tableau inséré puis le reste du tableau.

- ## La méthode concat(array)

- Cette méthode permet de fusionner deux tableaux en un nouveau tableau. Les deux tableaux initiaux ne seront pas modifiés

```
const fruits = ["pomme", "banane", "orange"];
```

```
console.log(fruits.concat(["ananas", "raisin"])); // [ 'pomme',
```

```
'banane', 'orange', 'ananas', 'raisin' ] un autre tableau
```

```
console.log(fruits);  // [ 'pomme', 'banane', 'orange' ]
```

Dans cet example, le tableau initial fruits ne se change pas et la méthode concat retourne un nouveau tableau qui la concaténation de deux tableaux.

- ## La méthode slice(start, end)

- Cette méthode permet d'extraire une partie du tableau et crée un nouveau tableau.

  Un paramètre négatif peut être utilisé pour un intervalle depuis la fin du tableau

```
const fruits = ["pomme", "banane", "orange", "ananas", "raisins" ];
```

```
console.log(fruits.slice(2,4)); //['orange', 'ananas']
```

```
console.log(fruits.slice(-2)); //['ananas', 'raisin']
```

```
console.log(fruits.slice(-3,-2)); // ['orange']
```

```
console.log(fruits);
```

- La méthode splice(start, deleteCount ?, ...items):

- Cette méthode permet de retirer des éléments d'un tableau et, si nécessaire, insère de nouveaux éléments à leur place, renvoyant les éléments supprimés et retourne un tableau contenant les éléments qui ont été supprimés.

- start – L'emplacement basé sur zéro dans le tableau à partir duquel commencer à retirer les éléments.
- deleteCount – Le nombre d'éléments à supprimer.

```
let noms = ["Alice", "Bob", "Charlie", "David", "Emma"];
```

```
// Ajouter un élément à la position 2
```

```
noms.splice(2, 0, "Frank");
```

```
// Remplacer un élément à la position 1
```

```
noms.splice(1, 1, "Eva");
```

```
// Supprimer deux éléments à partir de la position 3
```

```
noms.splice(3, 2);
```

```
console.log(noms); //[ 'orange', 'ananas', 'raisin' ]
```

```
Vous avez un tableau noms contenant plusieurs noms : ["Alice", "Bob',
```

```
"Charlie", "David", "Emma"].
```

Utilisation de noms.splice(2, 0, "Frank") :
L'index 2 indique que l'opération commencera à partir de la troisième position.
Le 0 indique que vous ne souhaitez supprimer aucun élément.
"Frank" est l'élément que vous souhaitez insérer à la position 2.
Le résultat après cette opération sera ["Alice", "Bob", "Frank", "Charlie", "David", "Emma"].

Utilisation de noms.splice(1, 1, "Eva") :
L'index 1 indique que l'opération commencera à partir de la deuxième position.
Le 1 indique que vous souhaitez supprimer un élément.
"Eva" est l'élément que vous souhaitez insérer à la position 1.
Le résultat après cette opération sera ["Alice", "Eva", "Frank", "Charlie", "David", "Emma"].

Utilisation de noms.splice(3, 2) :
L'index 3 indique que l'opération commencera à partir de la quatrième position.
Le 2 indique que vous souhaitez supprimer deux éléments à partir de cette position.
Le résultat après cette opération sera ["Alice", "Eva", "Frank"].

- La méthode IndexOf(element, startIndex)

- Cette méthode permet de renvoyer l'indice de la première occurrence d'un élément dans le tableau.

```
const noms = ["Alice", "Bob", "Charlie", "David", "Emma"];
```

```
const indiceCharlie = noms.indexOf("Charlie");
```

```
const indiceEva = noms.indexOf("Eva");
```

```
console.log("Indice de Charlie :", indiceCharlie); // Affiche
```

```
"Indice de Charlie : 2"
```

```
console.log("Indice de Eva :", indiceEva); // Affiche "Indice de
```

```
Eva : -1"
```

noms.indexOf("Charlie") : La méthode indexOf() recherche la première occurrence de l'élément "Charlie" dans le tableau.
Comme "Charlie" est à la position 2 dans le tableau, l'indice renvoyé sera 2.

noms.indexOf("Eva") : La méthode indexOf() recherche l'élément "Eva" dans le tableau.
Comme "Eva" n'est pas dans le tableau, la méthode renverra -1 pour indiquer que l'élément n'a pas été trouvé.

-

La méthode includes(element)

-

Cette méthode permet de vérifier si un élément existe dans le tableau.

```
const noms = ["Alice", "Bob", "Charlie", "David", "Emma"];
```

```
const charlieExiste = noms.includes("Charlie");
```

```
const evaExiste = noms.includes("Eva");
```

```
console.log("Charlie existe :", charlieExiste); // Affiche "Charlie
```

```
existe : true"
```

```
console.log("Eva existe :", evaExiste); // Affiche "Eva existe :
```

```
false"
```

Dans l'exemple 1, noms.includes("Charlie") : Comme "Charlie" est dans le tableau, la méthode renverra true. Pour l'exemple 2, noms.includes("Eva") : Comme "Eva" n'est pas dans le tableau, la méthode renverra false.

-

La méthode forEach(callback)

-

Cette méthode est utilisée pour parcourir un tableau et exécuter une fonction de rappel sur chaque élément

```
const noms = ["Alice", "Bob", "Charlie", "David", "Emma"];
```

```
noms.forEach(function(nom) {
```

```
  console.log("Bonjour, " + nom + " !");
```

```
});
```

```
/* Résultats
```

```
Bonjour, Alice !
```

```
Bonjour, Bob !
```

```
Bonjour, Charlie !
```

```
Bonjour, David !
```

```
Bonjour, Emma !
```

```
 */
```

Vous appelez la méthode forEach() sur le tableau noms. Cette méthode parcourt chaque élément du tableau et exécute la fonction de rappel sur chaque élément.
La fonction de rappel est une fonction anonyme passée en argument à forEach(). Cette fonction prend un paramètre nom qui représente l'élément en cours du tableau

À l'intérieur de la fonction de rappel, vous utilisez console.log() pour afficher un message de salutation pour chaque nom. La fonction de rappel sera exécutée pour chaque nom dans le tableau, affichant ainsi un message pour chaque nom.

-

La méthode filter(callback)

-

Cette méthode permet de créer un nouveau tableau contenant les éléments qui passent un test spécifié.

```
conso nombres = [10, 20, 25, 30, 35, 40, 45, 50];
```

```
const nombresPairs = nombres.filter(function(nombre) {
```

```
  return nombre % 2 === 0;
```

```
});
```

```
console.log(nombresPairs); // Affiche [10, 20, 30, 40, 50]
```

Vous appelez la méthode filter() sur le tableau nombres. Cette méthode crée un nouveau tableau contenant uniquement les éléments qui satisfont à la condition spécifiée dans la fonction de rappel.

La fonction de rappel est une fonction anonyme passée en argument à filter(). Cette fonction prend un paramètre nombre qui représente l'élément en cours du tableau.

À l'intérieur de la fonction de rappel, vous utilisez l'opérateur modulo % pour vérifier si le nombre est pair. Si nombre % 2 est égal à 0, alors le nombre est pair. Si la condition est vraie (le nombre est pair), l'élément est inclus dans le nouveau tableau nombresPairs. En fin de compte, nombresPairs contiendra les nombres pairs du tableau d'origine.

Quelle méthode JavaScript est utilisée pour ajouter un élément à la fin d'un tableau ?

- addToEnd()
-
- push()
-
- append()

-
-
  pop()
-

Objet Array

L'objet Boolean est associé aux valeurs booléennes true (vrai) et false (faux). Bien que les valeurs booléennes soient généralement utilisées sans l'objet Boolean, cet objet offre quelques méthodes utiles pour travailler avec ces valeurs.

- ### Le constructeur Boolean()

- Pour créer in objet de type Boolean, on utilise le constructeur Boolean()

```
const vrai = new Boolean(true);
```

```
const faux = new Boolean(false);
```

```
console.log(vrai); // Affiche [Boolean: true]
```

```
console.log(faux); // Affiche [Boolean: false]
```

Dans cet exemple, nous créons des instances de l'objet Boolean en utilisant le constructeur. Cependant, cela n'est pas couramment utilisé car il est plus simple d'utiliser les valeurs booléennes primitives (true et false) sans le constructeur.

- ### La méthode valueOf()

- Cette méthode permet de renvoyer la valeur booléenne associée à un objet Boolean.

```
const vrai = new Boolean(true);
```

```
console.log(vrai.valueOf()); // Affiche true
```

Dans cet exemple, nous créons une instance de l'objet Boolean avec la valeur true. En utilisant la méthode valueOf(), nous obtenons la valeur booléenne associée, qui est true.

- ### La méthode toString()

- Cette méthode permet de renvoyer une chaîne de caractères représentant la valeur booléenne de l'objet.

```
const faux = new Boolean(false);
```

```
console.log(faux.toString()); // Affiche "false"
```

Ici, nous créons une instance de l'objet Boolean avec la valeur false. En utilisant la méthode toString(), nous obtenons la représentation sous forme de chaîne de caractères de la valeur booléenne, qui est "false".

Objet Boolean

L'objet Date est utilisé pour travailler avec les dates et les heures. Il offre plusieurs méthodes pour créer, manipuler et formater les dates et les heures.

- ### Le constructeur Date()

- Le constructeur Date() peut être utilisé pour créer une instance de l'objet Date. Si aucun argument n'est passé, il renverra la date et l'heure actuelles.

```
const dateActuelle = new Date();
```

```
console.log(dateActuelle); // Affiche la date et l'heure actuelles
```

Dans cet exemple, nous créons une instance de l'objet Date en utilisant le constructeur sans argument. Cela crée une instance contenant la date et l'heure actuelles.

Pour créer l'objet Date on pourra utiliser aussi :

Nom_de_l_objet = new Date("jour, mois date année heures:minutes:secondes")

```
Nom_de_l_objet = new Date("10,09,2023, 11:11:11");
```

```
console.log(Nom_de_l_objet) //2023-10-09T10:11:11.000Z
```

les paramètres sont une chaîne de caractère suivant scrupuleusement la notation ci-dessus

Nom_de_l_objet = new Date(année, mois, jour)

```
const date = new Date(2023,2,2);
```

```
console.log(date); //2023-03-01T23:00:00.000Z
```

les paramètres sont trois entiers séparés par des virgules.
Les paramètres omis sont mis à zéro par défaut

Nom_de_l_objet = new Date(année, mois, jour, heures, minutes, secondes[, millisecondes])

```
const date = new Date(2023,2,2, 16,50,56);
```

```
console.log(date); //2023-03-02T15:50:56.000Z
```

les paramètres sont six entiers séparés par des virgules.
Les paramètres omis sont mis à zéro par défaut

Les dates en Javascript sont stockées de la même manière que dans le langage Java, c'est-à-dire qu'il s'agit du nombre de millisecondes depuis le 1er janvier 1970. Ainsi, toute date antérieure au 1er janvier 1970 fournira une valeur erronée.

- ## La méthode getFullYear()

- Cette méthode permet de renvoyer l'année d'une instance de l'objet Date.

```
const dateActuelle = new Date();
```

```
const annee = dateActuelle.getFullYear();
```

```
console.log(annee); // Affiche l'année actuelle (par exemple, 2023)
```

Ici, nous créons une instance de l'objet Date avec la date actuelle. En utilisant la méthode getFullYear(), nous extrayons l'année de cette instance.

- La méthode getMonth():

- Cette méthode permet de renvoyer le mois (0 pour janvier, 1 pour février, etc.) d'une instance de l'objet Date.

```
const dateActuelle = new Date();
```

```
const mois = dateActuelle.getMonth();
```

```
console.log(mois); // 7 par exemple
```

Dans cet exemple, nous obtenons le mois actuel à partir de l'instance de l'objet Date. Notez que les mois sont indexés à partir de 0, donc janvier est représenté par 0 et décembre par 11.

- La méthode toLocaleDateString()

- Cette méthode permet de renvoyer une représentation formatée de la date sous forme de chaîne.

```
const dateActuelle = new Date();
```

```
const dateFormatee = dateActuelle.toLocaleDateString();
```

```
console.log(dateFormatee); // Affiche la date formatée (par exemple,
```

```
"27/08/2023")
```

En utilisant la méthode toLocaleDateString(), nous obtenons une version formatée de la date actuelle dans le format spécifique de votre environnement.

- La méthode getHours()

- Cette méthode permet de préciser la valeur de l'heure. Le résultat est un entier (entre 0 et 23) qui correspond à l'objet Date.

- La méthode getMilliseconds()

- Cette méthode permet de récupérer le nombre de millisecondes. L'entier retourné est (entre 0 et 999) correspond aux millisecondes de l'objet passé en paramètre.

- La méthode getMinutes()

- Cette méthode permet de récupérer la valeur des minutes. L'objet retourné est un entier (entre 0 et 59) qui correspond aux minutes de l'objet Date

```
const dateActuelle = new Date();

console.log(dateActuelle) //2023-08-27T21:18:57.868Z

const hours = dateActuelle.getHours();

const minutes = dateActuelle.getMinutes();

const secondes = dateActuelle.getSeconds();

console.log(hours); // 22

console.log(minutes); // 18

console.log(secondes);  // 57
```

Dans cet exemple, l'objet DateActuelle est 2023-08-27T21:18:57.868Z
chaque méthode de getter récupère respectivement l'heure, minutes et secondes de la date.

Pour modifier la date, vous allez utiliser les méthodes (setters) comme les suivants :

- **La méthode setDate(X)**
  Cette méthode permet de fixer la valeur du jour du mois. Le paramètre est un entier (entre 1 et 31) qui correspond au jour du mois

- **La méthode setHours(X)**
  Cette méthode permet de fixer la valeur de l'heure. Le paramètre est un entier (entre 0 et 23) qui correspond à l'heure

- **La méthode setMinutes(X)**
  Cette méthode permet de fixer la valeur des minutes. Le paramètre est un entier (entre 0 et 59) qui correspond aux minutes

- **La méthode setMonth(X)**
  Cette méthode permet de fixer le numéro du mois. Le paramètre est un entier (entre 0 et 11) qui correspond au mois :

- **La méthode setTime(X)**
  Cette méthode permet d'assigner la date. Le paramètre est un entier représentant le nombre de millisecondes depuis le 1er janvier 1970

```
const date = new Date();
```

```
console.log(date) //2023-08-27T21:35:29.120Z
```

```
date.setDate(9);
```

```
date.setHours(7);
```

```
console.log(date); //2023-08-09T07:35:29.120Z
```

Tout d'abord, nous avons affiché la date actuelle, puis on modifie le mois, l'heure. Notre objet date créé a subi les changements.

Quelle est la méthode utilisée pour créer une nouvelle instance de l'objet Date en JavaScript ?

- createDate()
-
-

new DateInstance()

-
-

Date.create()

-
-

new Date()

-

Objet Date

L'objet String est utilisé pour travailler avec des chaînes de caractères. Voici quelques-unes de ses méthodes les plus couramment utilisées :

- La méthode length()

- Cette méthode permet de retourner la longueur de la chaîne de caractères.

```
const myString = "Hello, world!";
```

```
const length = myString.length;
```

```
console.log(length); // Affiche 13
```

Tout caractère est calculé y compris l'espace, les points de ponctuation.

- La méthode charAt(index):

- Cette méthode permet de retourner le caractère à l'index spécifié dans la chaîne.

```
const myString = "Hello";
```

```
const char = myString.charAt(1);
```

```
console.log(char); // Affiche "e"
```

- **La méthode substring(start, end):**

Cette méthode permet d'extraire une sous-chaîne en fonction des indices de début et de fin spécifiés.

```
const myString = "Hello, world!";
```

```
const subString = myString.substring(7, 12);
```

```
console.log(subString); // Affiche "world"
```

- La méthode concat(str)

- Cette méthode permet de concaténer la chaîne actuelle avec une autre chaîne.

```
const str1 = "Hello";
```

```
const str2 = " world!";
```

```
const combined = str1.concat(str2);
```

```
console.log(combined); // Affiche "Hello world!"
```

- La méthode toUpperCase()

- Cette méthode permet de retourner une nouvelle chaîne avec tous les caractères en majuscules.

```
const myString = "Hello";
```

```
const upperCaseString = myString.toUpperCase();
```

```
console.log(upperCaseString); // Affiche "HELLO"
```

- ### La méthode toLowerCase()

- Cette méthode permet de retourner une nouvelle chaîne avec tous les caractères en minuscules

```
const myString = "Hello";
```

```
const lowerCaseString = myString.toLowerCase();
```

```
console.log(lowerCaseString); // Affiche "hello"
```

Si la chaine est initialement en minuscules. Elle reste inchangée

- ### La méthode replace(oldStr, newStr)

- Cette méthode permet de remplacer toutes les occurrences d'une sous-chaîne par une autre dans la chaîne.

```
const myString = "Hello, world!, Bonjour, world!";
```

```
const newString = myString.replace("world", "universe");
```

```
console.log(newString); // Hello, universe!, Bonjour, world!
```

- ### La méthode trim()

- Cette méthode permet de retourner une nouvelle chaîne en supprimant les espaces en début et en fin de chaîne.

```
const myString = "   Hello   ";
```

```
const trimmedString = myString.trim();
```

```
console.log(trimmedString); // Affiche "Hello"
```

- ## La méthode indexOf(str)

- Cette méthode permet de retourner l'index de la première occurrence de la sous-chaîne spécifiée.

```
const myString = "Hello, world!";
```

```
const index = myString.indexOf("world");
```

```
console.log(index); // Affiche 7
```

Dans cet exemple, le mot « world » commence à la position 7 de la chaine initiale. IndexOf retourne la première position de la première lettre de la chaine en paramètre.

- ## La méthode split(delimiter)

- Cette méthode permet de diviser la chaîne en un tableau de sous-chaînes en utilisant le délimiteur spécifié.

```
const myString = "apple,banana,orange";
```

```
const fruits = myString.split(",");
```

```
console.log(fruits); // Affiche ["apple", "banana", "orange"]
```

Le résultat est un array qui contient des sous-chaines délimités par le délimiteur "," mis en paramètre.

L'objet String possède aussi des méthodes qui résultent des chaines de caractères contenant des balises html. Par exemple, anchor() qui rend la chaine de caractère un lien. La méthode bold() rend le texte en gras. Vous pouvez consulter les exemples ci-dessous ainsi le résultat affiché en commentaires.

```
// anchor("nom_a_donner")
```

```
const linkText = "Cliquez ici";
```

```
const link = linkText.anchor("https://www.example.com");

console.log(link); // <a name="https://www.example.com">Cliquez
ici</a>


// bold()

const text = "Texte en gras";

const boldText = text.bold();

console.log(boldText); // <strong>Texte en gras</strong>


// fontcolor(couleur)

const message = "Hello, world!";

const coloredMessage = message.fontcolor("blue");

console.log(coloredMessage); // <font color="blue">Hello,
world!</font>


// fontsize(Size)

const content = "Texte de taille différente";

const largeText = content.fontsize(5);
```

```
console.log(largeText); // <font size="5">Texte de taille différente</font>

// italics()
const emphasis = "Texte en italique";
const italicText = emphasis.italics();
console.log(italicText); // <em>Texte en italique</em>

// link(URL)
const siteName = "Visitez notre site";
const siteLink = siteName.link("https://www.example.com");
console.log(siteLink); // <a href="https://www.example.com">Visitez notre site</a>

// small()
const note = "Note importante";
const smallNote = note.small();
console.log(smallNote); // <small>Note importante</small>
```

```
// strike()

const deletedText = "Ce texte a été supprimé";

const strikethroughText = deletedText.strike();

console.log(strikethroughText); // <strike>Ce texte a été
supprimé</strike>


// sub()

const chemicalFormula = "H2O";

const subscriptFormula = chemicalFormula.sub();

console.log(subscriptFormula); // <sub>H2O</sub>
```


Objet String

Un objet Javascript posséde des méthodes standards et aussi trigométrie

- La méthode Math.random()
- Cette méthode permet de renvoyer un nombre décimal pseudo-aléatoire entre 0 (inclus) et 1 (exclus).

```
const randomValue = Math.random();
```

```
console.log(randomValue); // Affiche un nombre aléatoire entre 0 et 1
```

Math.random() génère un nombre décimal aléatoire entre 0 et 1 (0 inclus, 1 exclus).

- ### La méthode Math.floor()

- Cette méthode permet de renvoyer le plus grand entier inférieur ou égal à un nombre.

```
const number = 7.85;
```

```
const floorValue = Math.floor(number);
```

```
console.log(floorValue); // Affiche 7
```

Cette méthode Math.floor(7.85) permet de renvoyer l'entier le plus grand ou égal à 7.85, ce qui est 7.

- ### La méthode Math.ceil()

- Cette méthode renvoie le plus petit entier supérieur ou égal à un nombre.

```
const number = 3.14;
```

```
const ceilValue = Math.ceil(number);
```

```
console.log(ceilValue); // Affiche 4
```

Cette méthode Math.ceil(3.14) permet de renvoyer l'entier le plus petit ou égal à 3.14, ce qui est 4.

- ### La méthode Math.round()

- Cette méthode permet de renvoyer l'entier le plus proche d'un nombre donné.

```
const number = 5.6;
```

```
const roundedValue = Math.round(number);
```

```
console.log(roundedValue); // Affiche 6
```

Math.round(5.6) renvoie l'entier le plus proche de 5.6, ce qui est 6.

- ### Les méthodes Math.max() et Math.min()

- Ces méthodes permettent de renvoyer le plus grand et le plus petit élément, respectivement, parmi un ensemble d'arguments numériques.

```
const maxValue = Math.max(10, 5, 20, 15);
```

```
const minValue = Math.min(10, 5, 20, 15);
```

```
console.log(maxValue); // Affiche 20
```

```
console.log(minValue); // Affiche 5
```

Math.max(10, 5, 20, 15) renvoie la valeur maximale (20) parmi les arguments fournis. Math.min(10, 5, 20, 15) renvoie la valeur minimale (5).

- ### La méthode Math.pow()

- Cette méthode permet de renvoyer la valeur d'un nombre élevé à une puissance donnée.

```
const result = Math.pow(2, 3);
```

```
console.log(result); // Affiche 8 (2^3)
```

Math.pow(2, 3) calcule 2 élevé à la puissance 3, ce qui donne 8.

Il y aussi d'autres méthodes comme PI, sin(valeur), cos(valeur),tan(valeur),atan(valeur), etc .. qui retournent respectivement le valeur pi, le sinus,le cosinus, la tangente, l'atengente . . ..

Ces méthodes sont utilisées dans des exemples précises de Trigonométrie.

Objet Math

L'objet document fournit une interface pour interagir avec les éléments HTML de la page, modifier son contenu et sa structure, ainsi que répondre aux événements utilisateur.

Les méthodes de la classe Window

- **alert(message)**

La méthode `alert(message)` est utilisée pour afficher une boîte de dialogue modale d'alerte dans le navigateur.
Cette boîte de dialogue contient un message spécifié par l'utilisateur et un bouton "OK".
Lorsque la boîte de dialogue s'affiche, elle bloque l'interaction avec la page jusqu'à ce que l'utilisateur clique sur le bouton "OK" ou ferme la boîte de dialogue.

Paramètres :

- `message` : Une chaîne de caractères qui spécifie le message à afficher dans la boîte de dialogu

```
const message = "Ceci est un message d'alerte !";
```

```
window.alert(message);
```

Dans cet exemple, lorsque le code est exécuté, une boîte de dialogue d'alerte s'affichera avec le message "Ceci est un message d'alerte !".
L'utilisateur devra cliquer sur le bouton "OK" pour fermer la boîte de dialogue et reprendre l'interaction avec la page.

- **confirm(message)**

- La méthode `confirm(message)` est utilisée pour afficher une boîte de dialogue modale de confirmation dans le navigateur.
  Cette boîte de dialogue contient un message spécifié par l'utilisateur, ainsi que deux boutons : "OK" et "Annuler".
  Elle permet à l'utilisateur de confirmer ou d'annuler une action ou une décision.

  Paramètres :

- `message` : Une chaîne de caractères qui spécifie le message à afficher dans la boîte de dialogue.

```
const actionMessage = "Voulez-vous continuer ?";

const userConfirmed = window.confirm(actionMessage);



if (userConfirmed) {

    console.log("L'utilisateur a confirmé l'action.");

} else {

    console.log("L'utilisateur a annulé l'action.");

}
```

Dans cet exemple, lorsque le code est exécuté, une boîte de dialogue de confirmation s'affichera avec le message "Voulez-vous continuer ?".
Si l'utilisateur clique sur "OK", la variable `userConfirmed` sera `true`, et le message "L'utilisateur a confirmé l'action." sera affiché dans la console.
Si l'utilisateur clique sur "Annuler", la variable `userConfirmed` sera `false`, et le message "L'utilisateur a annulé l'action." sera affiché dans la console.

-

prompt(message, defaultText)

- La méthode `prompt(message, defaultText)` est utilisée pour afficher une boîte de dialogue modale avec un champ de saisie.
  Cette boîte de dialogue permet à l'utilisateur de saisir une valeur, généralement en réponse à une question ou à une demande spécifique.

  Paramètres :

  - `message` : Une chaîne de caractères qui spécifie le message à afficher dans la boîte de dialogue.
  - `defaultText` : (Facultatif) Une chaîne de caractères qui sera pré-remplie dans le champ de saisie.

```
const userName = window.prompt("Entrez votre nom :", "John Doe");
```

```
if (userName !== null) {
```

```
    console.log("Bonjour, " + userName + " !");
```

```
} else {
```

```
    console.log("L'utilisateur a annulé.");
```

```
}
```

  Dans cet exemple, lorsque le code est exécuté, une boîte de dialogue avec le message "Entrez votre nom :" et un champ de saisie sera affichée.
  Si l'utilisateur saisit un nom et clique sur "OK", le nom saisi sera stocké dans la variable `userName`, et un message de salutation sera affiché dans la console.
  Si l'utilisateur clique sur "Annuler", la variable `userName` sera `null`, et le message "L'utilisateur a annulé." sera affiché.

- **open(url, target, features)**

La méthode open(url, target, features) est utilisée pour ouvrir une nouvelle fenêtre ou un nouvel onglet dans le navigateur avec l'URL spécifiée et des paramètres de personnalisation tels que la taille, la position et les barres d'outils.

Paramètres :

- url : Une chaîne de caractères qui spécifie l'URL à ouvrir dans la nouvelle fenêtre.
- target : (Facultatif) Une chaîne de caractères qui indique comment la nouvelle fenêtre doit être affichée. Les valeurs possibles sont _blank, _self, _parent, _top ou le nom d'une fenêtre existante.
- features : (Facultatif) Une chaîne de caractères qui spécifie les fonctionnalités de la nouvelle fenêtre, telles que la taille, la position, les barres d'outils, etc.

```
const newWindow = window.open("https://www.example.com", "_blank",
"width=500,height=300");
```

Dans cet exemple, lorsque le code est exécuté, une nouvelle fenêtre ou un nouvel onglet sera ouvert avec l'URL "https://www.example.com".
La nouvelle fenêtre aura une largeur de 500 pixels et une hauteur de 300 pixels, comme spécifié dans le paramètre features.

- **close()**

La méthode close() est utilisée pour fermer la fenêtre actuelle (ou l'onglet) dans le navigateur.

Cela peut être utile lorsque vous avez créé une nouvelle fenêtre ou un nouvel onglet à l'aide de la méthode open() et que vous souhaitez fermer cette fenêtre programmématiquement.

```
const openWindow = window.open("https://www.example.com", "_blank",
"width=500,height=300");
```

```
setTimeout(() => {
    openWindow.close();
}, 5000);
```

Dans cet exemple, le code ouvre une nouvelle fenêtre ou un nouvel onglet avec l'URL "https://www.example.com" et les dimensions spécifiées. Ensuite, une minuterie est définie pour fermer cette fenêtre après 5 secondes à l'aide de la méthode `close`.

- **setTimeout(callback, delay)**

- La méthode `setTimeout(callback, delay)` est utilisée pour planifier l'exécution d'une fonction (appelée rappel ou callback) après un délai spécifié.
  Cette fonction est exécutée une seule fois après le délai écoulé.

  Paramètres :

  - `callback` : Une fonction (rappel) à exécuter après le délai écoulé.
  - `delay` : Le délai en millisecondes avant que la fonction `callback` soit exécutée

```
function showMessage() {
    console.log("Le délai est écoulé !");
}

const delay = 3000; // 3 secondes
const timeoutId = window.setTimeout(showMessage, delay);
```

- **clearTimeout(timeoutId)**

- Annule l'exécution planifiée d'une fonction avec `setTimeout()`.

```
const timeoutId = setTimeout(() => {
```

```
console.log("Cette fonction ne sera pas exécutée.");

}, 5000);

clearTimeout(timeoutId);
```

Dans cet exemple, le code définit une fonction showMessage qui affiche un message dans la console. Ensuite, il utilise la méthode setTimeout pour planifier l'exécution de cette fonction après un délai de 3 secondes.
Après 3 secondes, le message "Le délai est écoulé !" sera affiché dans la console.

- **setInterval(callback, delay)**

La méthode setInterval(callback, delay) est utilisée pour planifier l'exécution répétitive d'une fonction (appelée rappel ou callback) avec un intervalle de temps fixe entre chaque exécution.
La fonction est exécutée plusieurs fois à des intervalles réguliers jusqu'à ce que l'intervalle soit annulé avec clearInterval().

Paramètres :

- callback : Une fonction (rappel) à exécuter périodiquement.
- delay : L'intervalle en millisecondes entre chaque exécution de la fonction callback.

```
function printTime() {

    console.log(new Date().toLocaleTimeString());

}

const intervalId = window.setInterval(printTime, 1000);

// Exécuter toutes les 1 seconde
```

Dans cet exemple, le code définit une fonction printTime qui affiche l'heure actuelle dans la console au format de l'heure locale.

Ensuite, il utilise la méthode setInterval pour planifier l'exécution répétitive de cette fonction toutes les 1 seconde (1000 millisecondes).
L'heure sera affichée dans la console chaque seconde.

- **clearInterval(intervalId)**

- La méthode clearInterval(intervalId) est utilisée pour annuler un intervalle défini précédemment avec la méthode setInterval().
Lorsque vous appelez clearInterval() avec l'identifiant de l'intervalle, cet intervalle cesse d'exécuter la fonction de rappel périodiquement.

Paramètres :

- intervalId : L'identifiant de l'intervalle à annuler, qui a été retourné lors de l'appel à setInterval().

```
function printCounter() {
    console.log("Compteur :", counter);
    counter++;
}

let counter = 1;
const intervalId = window.setInterval(printCounter, 1000); // Exécuter toutes les 1 seconde

// Annuler l'intervalle après 5 secondes
setTimeout(() => {
```

```
    window.clearInterval(intervalId);

    console.log("Intervalle annulé.");

}, 5000);
```

Dans cet exemple, le code définit une fonction printCounter qui affiche la valeur d'un compteur dans la console.
Ensuite, il utilise la méthode setInterval pour planifier l'exécution répétitive de cette fonction toutes les 1 seconde.
Après 5 secondes, l'intervalle est annulé à l'aide de la méthode clearInterval, ce qui arrête l'exécution de la fonction printCounter.

- **requestAnimationFrame(callback)**

- La méthode requestAnimationFrame(callback) est utilisée pour planifier l'exécution d'une fonction de rappel (callback) avant le prochain rafraîchissement de l'écran.
  Cela permet de réaliser des animations fluides et optimisées en exploitant le taux de rafraîchissement du moniteur.

  Paramètres :

- callback : Une fonction de rappel à exécuter avant le prochain rafraîchissement de l'écran

```
function animate(timestamp) {

    // Code d'animation ici

    console.log("Animation en cours à", timestamp);

    // Planifier la prochaine animation

    const nextRequestId = window.requestAnimationFrame(animate);
```

```
}
```

```
const initialRequestId = window.requestAnimationFrame(animate);
```

Dans cet exemple, le code définit une fonction `animate` qui sert d'animation.
Cette fonction est planifiée avec `requestAnimationFrame` pour être exécutée avant chaque rafraîchissement d'écran.
Elle reçoit le `timestamp` qui indique le moment où elle est appelée.
À l'intérieur de la fonction, vous pouvez placer le code d'animation et prévoir la prochaine animation en appelant `requestAnimationFrame` à nouveau.

- **cancelAnimationFrame(requestId)**

La méthode `cancelAnimationFrame(requestId)` est utilisée pour annuler une demande de rafraîchissement d'animation précédemment planifiée à l'aide de la méthode `requestAnimationFrame()`.
Cela permet d'arrêter une animation en cours ou d'empêcher qu'une animation planifiée ne soit exécutée.

Paramètres :

- `requestId` : L'identifiant de demande retourné par la méthode `requestAnimationFrame()` lors de la planification de l'animation.

```
let animationRequestId;
```

```
function animate(timestamp) {
```

```
    // Code d'animation ici
```

```
    console.log("Animation en cours à", timestamp);
```

```
    // Planifier la prochaine animation
```

```
    animationRequestId = window.requestAnimationFrame(animate);
}

// Démarrer l'animation
animationRequestId = window.requestAnimationFrame(animate);

// Arrêter l'animation après 5 secondes
setTimeout(() => {
    window.cancelAnimationFrame(animationRequestId);
    console.log("Animation annulée.");
}, 5000);
```

Dans cet exemple, le code définit une fonction `animate` pour simuler une animation.
L'animation est planifiée avec `requestAnimationFrame`. Un identifiant de demande est stocké lorsque l'animation est planifiée.
Après 5 secondes, l'animation est annulée en utilisant `cancelAnimationFrame`, ce qui empêche les appels futurs à la fonction `animate`.

- **fetch(input, init)**

- La méthode `fetch(input, init)` est utilisée pour effectuer des requêtes réseau vers des ressources (comme des API) et récupérer des données.
  Elle fournit une interface moderne pour effectuer des appels HTTP et gérer les réponses de manière asynchrone.

Paramètres :

- input : Une URL (chaîne de caractères) ou un objet Request représentant la ressource cible à récupérer.
- init : (Facultatif) Un objet de configuration pour personnaliser la requête (headers, méthodes, corps, etc.).

```
fetch("https://api.example.com/data")
    .then(response => {
        if (!response.ok) {
            throw new Error("La réponse n'est pas OK");
        }
        return response.json();
    })
    .then(data => {
        console.log("Données récupérées :", data);
    })
    .catch(error => {
        console.error("Erreur :", error);
    });
```

Dans cet exemple, le code utilise la méthode fetch pour récupérer des données à partir de l'URL "https://api.example.com/data".
La première étape est de vérifier si la réponse est OK (response.ok).

Si la réponse est OK, elle est transformée en JSON à l'aide de `response.json()`, puis les données sont affichées dans la console. En cas d'erreur, elle est gérée dans la section `catch`.

- **moveTo(x, y)**

La méthode `moveTo(x, y)` est utilisée pour déplacer la fenêtre du navigateur à une position spécifique sur l'écran.
Les coordonnées `(x, y)` spécifient la nouvelle position de la fenêtre en pixels par rapport au coin supérieur gauche de l'écran.

Paramètres :

- `x` : La position horizontale (abscisse) en pixels par rapport au coin supérieur gauche de l'écran.
- `y` : La position verticale (ordonnée) en pixels par rapport au coin supérieur gauche de l'écran.

```
// Déplacer la fenêtre à la position (100, 200) sur l'écran
```

```
window.moveTo(100, 200);
```

Dans cet exemple, le code utilise la méthode `moveTo` pour déplacer la fenêtre à la position `(100, 200)` sur l'écran.
Cela signifie que le coin supérieur gauche de la fenêtre sera positionné à 100 pixels du bord gauche de l'écran et à 200 pixels du bord supérieur.

- **moveBy(x, y)**

La méthode `moveBy(x, y)` est utilisée pour déplacer la fenêtre du navigateur par un certain nombre de pixels dans les directions horizontale et verticale.
Les valeurs `x` et `y` spécifient le décalage en pixels à ajouter aux positions actuelles de la fenêtre.

Paramètres :

- `x` : Le décalage horizontal en pixels à ajouter à la position actuelle de la fenêtre.
- `y` : Le décalage vertical en pixels à ajouter à la position actuelle de la fenêtre.

```
// Déplacer la fenêtre de 50 pixels vers la droite et de 30 pixels
vers le bas

window.moveBy(50, 30);
```

Dans cet exemple, le code utilise la méthode moveBy pour déplacer la fenêtre du navigateur de 50 pixels vers la droite et de 30 pixels vers le bas à partir de sa position actuelle.
Cela signifie que la position horizontale de la fenêtre sera augmentée de 50 pixels et la position verticale sera augmentée de 30 pixels.

- **resizeTo(width, height)**

La méthode resizeTo(width, height) est utilisée pour redimensionner la fenêtre du navigateur à des dimensions spécifiques en pixels.
Les valeurs width et height spécifient la nouvelle largeur et hauteur de la fenêtre.

Paramètres :

- width : La nouvelle largeur en pixels de la fenêtre.
- height : La nouvelle hauteur en pixels de la fenêtre.

```
// Redimensionner la fenêtre à une largeur de 800 pixels et une
hauteur de 600 pixels

window.resizeTo(800, 600);
```

Dans cet exemple, le code utilise la méthode resizeTo pour redimensionner la fenêtre du navigateur à une largeur de 800 pixels et une hauteur de 600 pixels.
Après l'appel à la méthode, la fenêtre aura les nouvelles dimensions spécifiées.

- **resizeBy(widthDelta, heightDelta)**

La méthode `resizeBy(widthDelta, heightDelta)` est utilisée pour redimensionner la fenêtre du navigateur en ajoutant ou en soustrayant un certain nombre de pixels à sa largeur et à sa hauteur actuelles.
Les valeurs `widthDelta` et `heightDelta` spécifient les ajustements à apporter aux dimensions actuelles.

Paramètres :

- `widthDelta` : L'ajustement de largeur en pixels à ajouter (positif) ou à soustraire (négatif) à la largeur actuelle de la fenêtre.
- `heightDelta` : L'ajustement de hauteur en pixels à ajouter (positif) ou à soustraire (négatif) à la hauteur actuelle de la fenêtre.

```
// Ajuster la largeur de la fenêtre de +100 pixels et la hauteur de
-50 pixels

window.resizeBy(100, -50);
```

Dans cet exemple, le code utilise la méthode `resizeBy` pour ajuster la largeur de la fenêtre de +100 pixels et la hauteur de -50 pixels par rapport à leurs dimensions actuelles.
Cela signifie que la largeur de la fenêtre sera augmentée de 100 pixels et la hauteur sera réduite de 50 pixels.

-

scrollTo(x, y)

-

La méthode `scrollTo(x, y)` est utilisée pour faire défiler le contenu d'une page web vers une position spécifique dans les directions horizontale et verticale.
Les valeurs `x` et `y` spécifient les nouvelles coordonnées de défilement, en pixels, par rapport au coin supérieur gauche de la page.

Paramètres :

- `x` : La position horizontale (abscisse) en pixels par rapport au coin supérieur gauche de la page.
- `y` : La position verticale (ordonnée) en pixels par rapport au coin supérieur gauche de la page.

```
// Faire défiler la page pour que les coordonnées (200, 300) soient
visibles
```

```
window.scrollTo(200, 300);
```

- **scrollBy(x, y)**

- Fait défiler la fenêtre par la quantité spécifiée en pixels.

```
window.scrollBy(0, 100);
```

Dans cet exemple, le code utilise la méthode `scrollTo` pour faire défiler la page web de manière à ce que les coordonnées `(200, 300)` soient visibles dans la fenêtre du navigateur.
Cela signifie que le contenu de la page sera ajusté pour que le point `(200, 300)` soit positionné à l'emplacement le plus haut possible dans la fenêtre.

- **scroll(x, y)**

- La méthode `scroll(x, y)` est utilisée pour faire défiler le contenu d'une page web d'une quantité spécifique dans les directions horizontale et verticale.
Les valeurs `x` et `y` spécifient le nombre de pixels à défiler dans chaque direction.

Paramètres :

- `x` : Le nombre de pixels à faire défiler horizontalement.
- `y` : Le nombre de pixels à faire défiler verticalement.

```
// Faire défiler de +100 pixels horizontalement et de -50 pixels
verticalement

window.scroll(100, -50);
```

Dans cet exemple, le code utilise la méthode `scroll` pour faire défiler le contenu de la page de +100 pixels horizontalement et de -50 pixels verticalement.
Cela signifie que le contenu de la page sera déplacé vers la droite de 100 pixels et vers le haut de 50 pixels.

- **print()**

- La méthode `print()` est utilisée pour ouvrir la boîte de dialogue d'impression du navigateur, permettant à l'utilisateur d'imprimer le contenu actuel de la page web affichée.

```
// Ouvrir la boîte de dialogue d'impression lorsque le bouton est cliqué
const printButton = document.getElementById("print-button");
printButton.addEventListener("click", () => {
    window.print();
});
```

Dans cet exemple, le code utilise la méthode `print` pour ouvrir la boîte de dialogue d'impression lorsque l'utilisateur clique sur un bouton avec l'ID "print-button".
Lorsque l'utilisateur interagit avec la boîte de dialogue d'impression, il peut configurer les paramètres d'impression selon ses préférences.

- **matchMedia(mediaQuery)**

- La méthode `matchMedia(mediaQuery)` est utilisée pour vérifier si une requête media spécifiée dans la `mediaQuery` correspond aux caractéristiques du périphérique et du navigateur actuels.
Une requête media est généralement utilisée pour déterminer le type de périphérique, la largeur de l'écran, l'orientation, etc., et adapter le style ou le comportement en conséquence.

Paramètres :

- `mediaQuery` : Une chaîne de caractères représentant une requête media, par exemple `"screen and (max-width: 768px)"`.

```
// Vérifier si l'écran est de petite taille (inférieure à 768px)

const smallScreenQuery = window.matchMedia("(max-width: 768px)");


// Vérifier si la requête est satisfaite

if (smallScreenQuery.matches) {

    console.log("Écran de petite taille");

} else {

    console.log("Écran de taille normale");

}
```

Dans cet exemple, le code utilise la méthode `matchMedia` pour créer un objet `MediaQueryList` basé sur la requête media `"max-width: 768px"`.
Ensuite, il vérifie si la requête est satisfaite en accédant à la propriété `matches` de l'objet.
Si la condition est vraie, cela signifie que l'écran est de petite taille.

- **postMessage(message, targetOrigin, transfer)**

La méthode `postMessage(message, targetOrigin, transfer)` est utilisée pour envoyer un message entre fenêtres ou onglets qui ont été ouverts par le même domaine.
Cette méthode est couramment utilisée pour la communication entre une fenêtre parente et une fenêtre enfant, ou entre différentes fenêtres ou onglets d'une même application web.

Paramètres :

- `message` : Les données à envoyer à la fenêtre cible. Cela peut être une chaîne de caractères, un objet JSON, etc.
- `targetOrigin` : L'origine de la fenêtre cible. Cela indique à quelle fenêtre le message doit être envoyé, pour des raisons de sécurité.
- `transfer` : (Facultatif) Un tableau d'objets transférables (tels que des objets ArrayBuffer) à partager entre les fenêtres. Ces objets sont déplacés plutôt que copiés.

Fenêtre parente :

```
// Ouvrir une fenêtre enfant

const childWindow = window.open("child.html");



// Envoyer un message à la fenêtre enfant

const message = { text: "Hello from parent!" };

childWindow.postMessage(message, "*");
```

Fenêtre enfeant (child.html)

```
// Écouter les messages de la fenêtre parente

window.addEventListener("message", event => {

    if (event.origin === "http://example.com") {

        const receivedMessage = event.data;

        console.log("Message received:", receivedMessage);

    }
```

```
});
```

Dans cet exemple, la fenêtre parente envoie un message à la fenêtre enfant à l'aide de postMessage.
La fenêtre enfant écoute les messages à l'aide de l'événement message et vérifie l'origine du message pour des raisons de sécurité.
Si l'origine correspond, elle reçoit et traite le message.

La méthode postMessage est couramment utilisée pour créer des fonctionnalités de communication entre fenêtres, comme la synchronisation d'état entre une fenêtre parente et une fenêtre enfant dans une application web.

- **addEventListener(type, listener, options)**

La méthode addEventListener(type, listener, options) est utilisée pour ajouter un écouteur d'événement à un objet (dans ce cas, la fenêtre) afin de détecter quand un événement spécifique se produit.
L'écouteur exécute une fonction (le listener) en réponse à cet événement.

Paramètres :

- type : Le type d'événement que vous souhaitez écouter, comme "click", "load", "keydown", etc.
- listener : La fonction (ou la référence à la fonction) qui sera appelée lorsque l'événement se produit.
- options : (Facultatif) Un objet d'options pour spécifier des détails sur la gestion de l'écouteur, tels que la capture et l'utilisation passive.

```
// Ajouter un écouteur pour l'événement "click"

const button = document.getElementById("my-button");

button.addEventListener("click", function(event) {

    console.log("Le bouton a été cliqué !");

});
```

Dans cet exemple, le code ajoute un écouteur d'événement "click" au bouton avec l'ID "my-button". Lorsque le bouton est cliqué, la fonction de rappel est exécutée, affichant "Le bouton a été cliqué !" dans la console.

- **removeEventListener(type, listener, options)**

La méthode `removeEventListener(type, listener, options)` est utilisée pour retirer un écouteur d'événement précédemment ajouté à un objet (dans ce cas, la fenêtre) à l'aide de la méthode `addEventListener`. Cela permet de cesser de surveiller un certain type d'événement avec une fonction de rappel spécifique.

Paramètres :

- `type` : Le type d'événement pour lequel vous souhaitez retirer l'écouteur, par exemple `"click"`, `"load"`, `"keydown"`, etc.
- `listener` : La fonction de rappel que vous avez précédemment ajoutée avec `addEventListener` et que vous souhaitez retirer.
- `options` : (Facultatif) Les mêmes options que vous avez utilisées lors de l'ajout de l'écouteur. Ces options doivent correspondre à celles utilisées lors de l'ajout.

```
// Ajouter un écouteur pour l'événement "click"

const button = document.getElementById("my-button");


function clickHandler(event) {

    console.log("Le bouton a été cliqué !");

}


button.addEventListener("click", clickHandler);


// Retirer l'écouteur après un certain temps
```

```
setTimeout(() => {

    button.removeEventListener("click", clickHandler);

}, 5000);
```

Dans cet exemple, le code ajoute un écouteur d'événement "click" au bouton avec l'ID "my-button". Après un délai de 5 secondes, l'écouteur est retiré à l'aide de `removeEventListener`.
Cela signifie que la fonction `clickHandler` ne sera plus exécutée en réponse aux clics sur le bouton.

Recupération des données à partir d'un formulaire

Lorsque vous souhaitez accéder à un objet, simplement connaître son nom ne suffit pas. Vous devez également spécifier le "chemin d'accès" pour atteindre cet objet au sein de la structure.

Dans le contexte du navigateur et du JavaScript, cette idée est également applicable. Si vous voulez faire référence à une fenêtre, un document ou un élément spécifique, il est important de spécifier le chemin complet pour y accéder, surtout si vous traitez avec plusieurs contextes ou objets imbriqués.

Cependant, dans certains cas, il existe des abréviations et des conventions qui facilitent l'accès aux objets couramment utilisés :

- **Fenêtre par défaut** : Si vous n'indiquez pas explicitement le nom de la fenêtre, le navigateur utilisera par défaut la fenêtre courante. Cela simplifie l'accès à la fenêtre actuellement affichée.
-
-
- **L'objet document** : Dans la plupart des cas, vous pouvez omettre `window.document` et simplement utiliser `document`. Cela fonctionne car il n'y a généralement qu'un seul objet `document` dans la fenêtre.
-

Ces abréviations et conventions aident à rendre le code plus lisible et plus succinct, en évitant de devoir spécifier le chemin complet à chaque fois.

Le concept de "chemin d'accès" est crucial pour accéder aux objets au sein de structures arborescentes. Cela s'applique également à la manipulation d'objets dans le contexte du navigateur et du JavaScript, où des abréviations pratiques sont souvent utilisées pour simplifier l'accès aux objets courants

Voici des exemples pour recupérer les données à partir d'un formulaire:

- **< input type="text">**

```
<!DOCTYPE html>

<html>

<head>

    <title>Accès aux objets de formulaire</title>

</head>

<body>

    <form name="monFormulaire">

        <input type="text" name="nom" placeholder="Votre nom">

        <input type="text" name="prenom" placeholder="Votre prénom">

        <button type="button" onclick="afficherInfos()">Afficher les
informations</button>

    </form>



    <script>

        function afficherInfos() {

            // Accéder aux valeurs des champs en utilisant les noms
d'éléments
```

```
            const nomValue = window.document.monFormulaire.nom.value;

            const prenomValue =
window.document.monFormulaire.prenom.value;



            // Afficher le résultat dans une boîte de dialogue
"alert"

            const message = `Nom : ${nomValue}, Prénom :
${prenomValue}`;

            alert(message);

        }

    </script>

</body>

</html>
```

Lorsque le bouton est cliqué, la fonction afficherInfos() est déclenchée.
Dans la fonction afficherInfos(), nous utilisons les noms des champs de saisie (nom et prenom) pour accéder à leurs valeurs en utilisant la notation window.document.monFormulaire.nom.value et window.document.monFormulaire.prenom.value.
Nous construisons un message en utilisant les valeurs récupérées.
Enfin, nous affichons le message dans une boîte de dialogue "alert" en utilisant la fonction JavaScript alert(message).

- **<textarea> & <input type="checkbox">**

Lorsque vous cliquez sur le bouton, il affichera les informations saisies dans la zone de texte et vérifiera si la case à cocher est cochée ou non, tout cela sera affiché dans une boîte de dialogue "alert" :

```
<!DOCTYPE html>

<html>

<head>

    <title>Exemple avec <textarea> et <input type="checkbox"></title>

</head>

<body>

    <form name="monFormulaire">

        <textarea name="commentaire" placeholder="Votre commentaire"></textarea>

        <br>

        <label>

            <input type="checkbox" name="avis" value="positif"> Avis positif

        </label>

        <br>

        <button type="button" onclick="afficherInfos()">Afficher les informations</button>
```

```
    </form>


    <script>

        function afficherInfos() {

            // Accéder à la valeur de la zone de texte et de la case
à cocher

            const commentaireValue =
window.document.monFormulaire.commentaire.value;

            const avisChecked =
window.document.monFormulaire.avis.checked;


            // Construire le message en fonction de la case cochée
ou non

            let message = `Commentaire : ${commentaireValue}`;

            if (avisChecked) {

                message += "\nAvis positif : Oui";

            } else {

                message += "\nAvis positif : Non";
```

```
            }
```

```
            // Afficher le résultat dans une boîte de dialogue
"alert"
            alert(message);
        }
    </script>
</body>
</html>
```

Lorsque le bouton "Afficher les informations" est cliqué, la fonction afficherInfos() est appelée.
Dans cette fonction, nous accédons à la valeur du commentaire en utilisant window.document.monFormulaire.commentaire.value.
Nous vérifions si la case à cocher est cochée en utilisant window.document.monFormulaire.avis.checked.
En fonction de l'état de la case à cocher, nous construisons un message.
Nous affichons ensuite le message dans une boîte de dialogue "alert" en utilisant alert(message).

- **< input type="radio">**

Lorsque vous cliquez sur le bouton, il affichera les informations saisies dans le champ de texte, vérifiera l'option de bouton radio sélectionnée et affichera le tout dans une boîte de dialogue "alert" :

```
<!DOCTYPE html>
<html>
<head>
    <title>Exemple avec <input type="radio"></title>
```

```
</head>

<body>

    <form name="monFormulaire">

        <input type="text" name="nom" placeholder="Votre nom">

        <br>

        <label>

            <input type="radio" name="genre" value="homme"> Homme

        </label>

        <label>

            <input type="radio" name="genre" value="femme"> Femme

        </label>

        <br>

        <button type="button" onclick="afficherInfos()">Afficher les
informations</button>

    </form>



    <script>
```

```
        function afficherInfos() {

            // Accéder à la valeur du champ de texte

            const nomValue = window.document.monFormulaire.nom.value;



            // Trouver l'option de bouton radio sélectionnée

            const radioOptions = window.document.monFormulaire.genre;

            let genreValue = "";

            for (let i = 0; i < radioOptions.length; i++) {

                if (radioOptions[i].checked) {

                    genreValue = radioOptions[i].value;

                    break;

                }

            }



            // Construire le message en fonction de l'option de
bouton radio sélectionnée

            let message = `Nom : ${nomValue}\nGenre : ${genreValue}`;
```

```
            // Afficher le résultat dans une boîte de dialogue
"alert"
            alert(message);
        }
    </script>
</body>
</html>
```

Lorsque le bouton "Afficher les informations" est cliqué, la fonction afficherInfos() est appelée.
Nous accédons à la valeur du champ de texte en utilisant window.document.monFormulaire.nom.value.
Pour trouver l'option de bouton radio sélectionnée, nous parcourons toutes les options radio avec une boucle et vérifions quelle option est cochée.
En fonction de l'option de bouton radio sélectionnée, nous construisons un message.
Nous affichons le message dans une boîte de dialogue "alert" en utilisant alert(message).

- **<input type="number">, <input type="date">, <input type="range"> et <input type="url">.**

Lorsque vous cliquez sur le bouton, il affichera les informations saisies dans ces champs dans une boîte de dialogue "alert" :

```
<!DOCTYPE html>
<html>
<head>
    <title>Exemple avec différents types d'entrées</title>
</head>
```

```
<body>

    <form name="monFormulaire">

        <input type="number" name="age" placeholder="Âge">

        <br>

        <input type="date" name="dateNaissance">

        <br>

        <input type="range" name="niveau" min="1" max="10">

        <br>

        <input type="url" name="siteWeb" placeholder="URL du site
web">

        <br>

        <button type="button" onclick="afficherInfos()">Afficher les
informations</button>

    </form>


    <script>

        function afficherInfos() {

            // Accéder aux valeurs des différents types d'entrées
```

```
            const ageValue = window.document.monFormulaire.age.value;

            const dateNaissanceValue =
window.document.monFormulaire.dateNaissance.value;

            const niveauValue =
window.document.monFormulaire.niveau.value;

            const siteWebValue =
window.document.monFormulaire.siteWeb.value;



            // Construire le message avec les valeurs saisies

            const message = `Âge : ${ageValue}\nDate de naissance :
${dateNaissanceValue}\nNiveau : ${niveauValue}\nSite Web :
${siteWebValue}`;



            // Afficher le résultat dans une boîte de dialogue
"alert"

            alert(message);

        }

    </script>

</body>
```

```
</html>
```

Lorsque le bouton "Afficher les informations" est cliqué, la fonction afficherInfos() est appelée.
Nous accédons aux valeurs des champs en utilisant les noms d'éléments correspondants, par exemple window.document.monFormulaire.age.value pour l'âge.
Nous construisons un message en utilisant les valeurs récupérées.
Nous affichons le message dans une boîte de dialogue "alert" en utilisant alert(message).

- **<select>**

Lorsque vous sélectionnez une option et cliquez sur le bouton, il affichera l'option sélectionnée dans une boîte de dialogue "alert" :

```
<!DOCTYPE html>

<html>

<head>

    <title>Exemple avec la balise <select></title>

</head>

<body>

    <form name="monFormulaire">

        <select name="pays">

            <option value="fr">France</option>

            <option value="us">États-Unis</option>

            <option value="ca">Canada</option>
```

```
            <option value="uk">Royaume-Uni</option>

        </select>

        <br>

        <button type="button" onclick="afficherInfos()">Afficher le
pays sélectionné</button>

    </form>


    <script>

        function afficherInfos() {

            // Accéder à la valeur de l'option sélectionnée

            const paysValue =
window.document.monFormulaire.pays.value;


            // Construire le message avec l'option sélectionnée

            const message = `Pays sélectionné : ${paysValue}`;


            // Afficher le résultat dans une boîte de dialogue
"alert"

            alert(message);
```

```
        }
```

```
    </script>
```

```
</body>
```

```
</html>
```

Lorsque vous sélectionnez une option dans la liste déroulante et cliquez sur le bouton "Afficher le pays sélectionné", la fonction afficherInfos() est appelée.
Nous accédons à la valeur de l'option sélectionnée en utilisant window.document.monFormulaire.pays.value.
Nous construisons un message en utilisant la valeur récupérée.
Nous affichons le message dans une boîte de dialogue "alert" en utilisant alert(message).

Objet Document