Le langage JavaScript permet de gérer les formulaires définis dans une page web afin de renforcer l'interactivité de cette page. Découvrons ensemble comment faire.

JavaScript et les formulaires

Rappels sur les formulaires

Les formulaires enrichissent les pages web en permettant à l'utilisateur de saisir des informations sous différentes formes : zone de texte, cases à cocher, listes déroulante, etc. À l'intérieur d'une page web, un formulaire est défini sous la forme d'une balise HTML`<form>` contenant différents champs de saisie : balises`<input>`,`<select>` ou encore`<textarea>`.

Classiquement, les données saisies par l'utilisateur dans un formulaire sont ensuite envoyées via le réseau à un **serveur web** qui va les traiter et renvoyer une réponse adaptée au navigateur client sous la forme d'une nouvelle page web. Pour réaliser cette tâche, un serveur web emploie une technologie adaptée, comme par exemple le langage PHP.

Le mécanisme client/serveur du Web

Gestion des formulaires avec JavaScript

Grâce à JavaScript, nous allons pouvoir manipuler le formulaire et ses données directement côté navigateur client, avant (éventuellement) d'envoyer ces données vers un serveur externe. Ainsi, on pourra avertir immédiatement l'utilisateur en cas de saisie erronée, ou bien lui proposer une liste de suggestions au fur et à mesure de sa frappe, et bien d'autres choses encore.

Le formulaire d'exemple

Les zones de texte

Accès à la valeur saisie

Une zone de texte permet à l'utilisateur de saisir du texte sur une ou plusieurs lignes. Une zone monoligne correspond à la balise HTML`<input type="text">`. Une zone multiligne est définie grâce à la balise`<textarea>`.

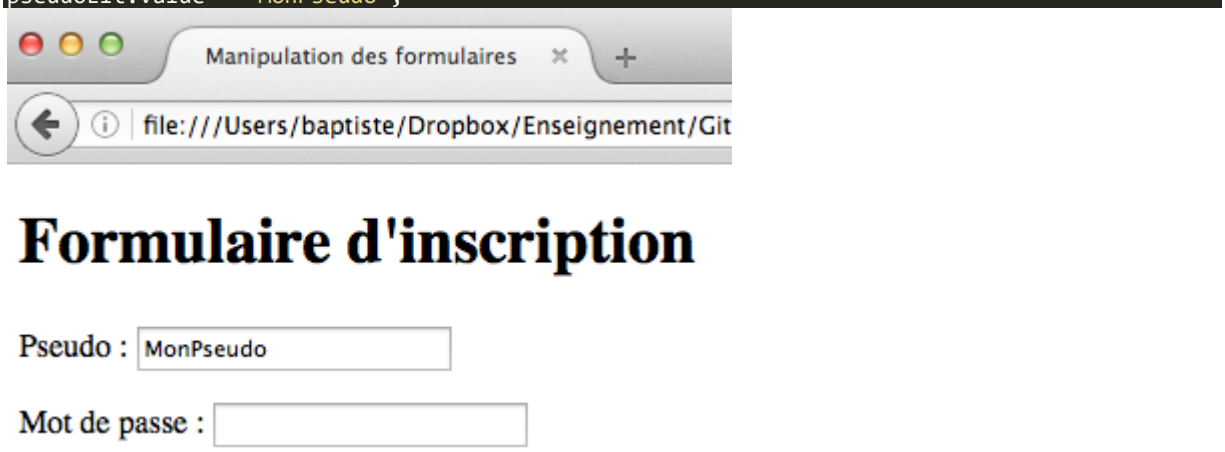
Voici l'extrait du formulaire précédent qui permet de créer la zone de saisie du pseudo.

```
<label for="pseudo">Pseudo</label> :
<input type="text" name="pseudo" id="pseudo" required>
<span id="aidePseudo"></span>
```

En JavaScript, on accède à la valeur d'une zone de texte en utilisant la propriété `value` de l'élément du DOM correspond. En donnant une nouvelle valeur à cette propriété, on modifie la valeur affichée dans la zone de texte.

L'exemple suivant donne à la zone de saisie identifiée par "pseudo" la valeur "MonPseudo".

```
var pseudoElt = document.getElementById("pseudo");
pseudoElt.value = "MonPseudo";
```



Gestion du focus

Lorsqu'une zone de texte est la cible de saisie, on dit que cette zone possède le **focus**. L'utilisateur d'un formulaire qui clique sur une zone de saisie pour y taper une information déclenche l'apparition d'un événement de type `focus`. Inversement, le changement de cible de saisie provoque l'apparition d'un événement de type `blur` sur l'ancienne zone qui avait le focus.

On peut utiliser ces événements pour afficher à l'utilisateur une aide contextuelle associée à la zone de saisie courante, comme dans l'exemple suivant.

```
// Affiche d'un message contextuel pour la saisie du pseudo
pseudoElt.addEventListener("focus", function () {
    document.getElementById("aidePseudo").textContent = "Entrez votre pseudo";
});
// Suppression du message contextuel pour la saisie du pseudo
pseudoElt.addEventListener("blur", function (e) {
    document.getElementById("aidePseudo").textContent = "";
});
```

En cliquant dans la zone de saisie du pseudo, vous obtenez l'affichage d'un message informatif dans la balise`<span>` prévue à cet effet et initialement vide.

Depuis le code JavaScript, on peut modifier la cible de saisie en appelant les méthodes `focus` (pour donner le focus) et `blur` (pour l'enlever) sur un élément du DOM.

```
// Focus sur la zone de saisie du pseudo
pseudoElt.focus();
```

Le fonctionnement des zones de texte multilignes (balises`<textarea>`) est similaire à celui des balises`<input>`.

La vérification des données saisies dans une zone de texte est abordée plus loin dans ce chapitre.

Les éléments d'options

Ces éléments permettent à l'utilisateur de faire un choix parmi plusieurs possibilités. Ils ont en commun de déclencher un événement de type `change` lorsque l'utilisateur modifie son choix.

Cases à cocher

On définit une case à cocher dans un formulaire avec la balise`<input type="checkbox">`.

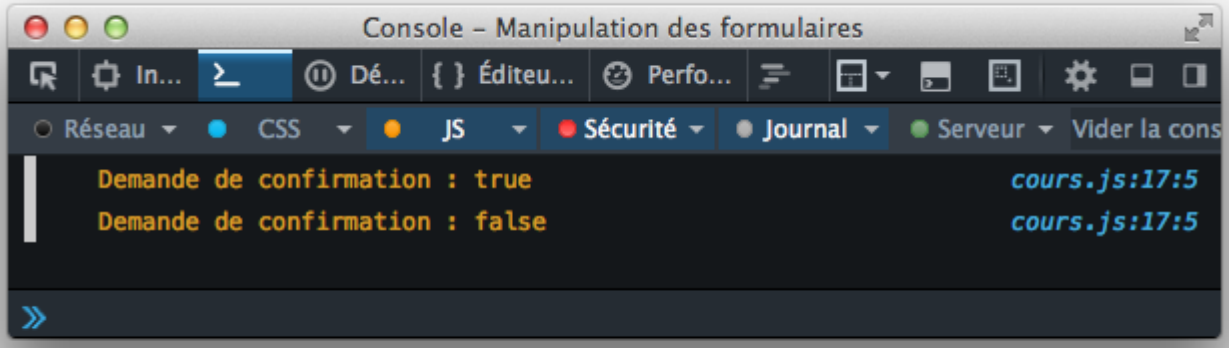
Voici l'extrait du formulaire initial qui permet de demander à l'utilisateur s'il souhaite ou non obtenir une confirmation de son inscription.

```
<input type="checkbox" name="confirmation" id="confirmation">
<label for="confirmation">M'envoyer un courriel de confirmation</label>
```

Lorsque la case change de valeur, l'objet `Event` associé à l'événement dispose d'une propriété booléenne `checked` qui indique le nouvel état de la case (cochée/décochée).

L'exemple de code ci-dessous gère les événements `change` sur la case à cocher pour afficher dans la console le choix effectué lorsque l'utilisateur clique dans la case ou sur le libellé associé.

```
// Affichage de la demande de confirmation d'inscription
document.getElementById("confirmation").addEventListener("change", function (e) {
    console.log("Demande de confirmation : " + e.target.checked);
});
```



Boutons radio

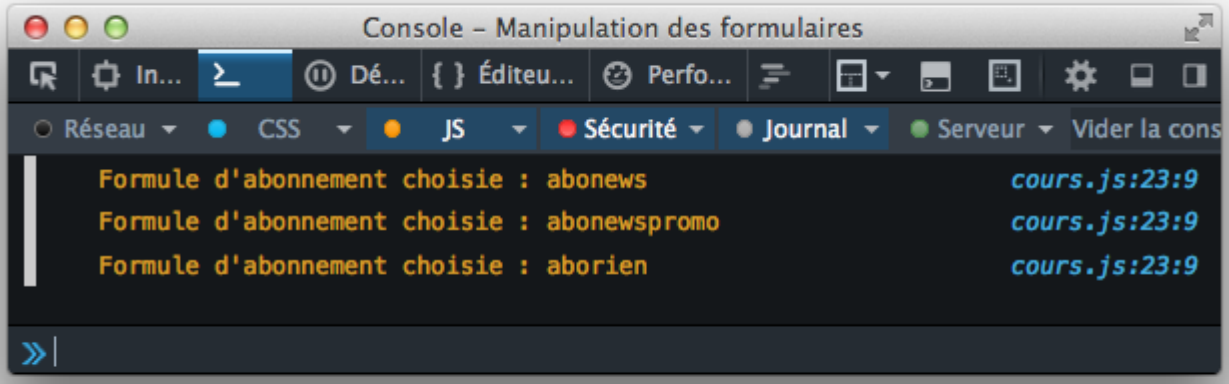
Un groupe de boutons radio permet à l'utilisateur de faire un seul choix parmi plusieurs possibilités. On crée ces boutons avec des balises`<input type="radio">` possédant le même attribut `name` et des attributs `value` différents.

Voici l'extrait du formulaire initial qui crée trois boutons radio permettant de choisir un type d'abonnement.

```
<input type="radio" name="abonnement" id="abonewspromo" value="abonewspromo">
<label for="abonewspromo">M'abonner à la newsletter et aux promotions</label>
<br>
<input type="radio" name="abonnement" id="abonews" value="abonews">
<label for="abonews">M'abonner uniquement à la newsletter</label>
<br>
<input type="radio" name="abonnement" id="aborien" value="aborien" checked>
<label for="aborien">Ne pas m'abonner</label>
<br>
```

L'exemple de code suivant ajoute le même gestionnaire pour les événements de type `change` sur chacun des boutons radio, afin d'afficher dans la console le type d'événement choisi.

```
// Affichage du type d'abonnement choisi
var aboElts = document.getElementsByName("abonnement");
for (var i = 0; i < aboElts.length; i++) {
    aboElts[i].addEventListener("change", function (e) {
        console.log("Formule d'abonnement choisie : " + e.target.value);
    });
}
```



Lors d'un changement de valeur d'un groupe de bouton radio, la propriété`e.target.value` de l'événement `change` contient la valeur de l'attribut `value` de la nouvelle balise`<input>` sélectionnée.

Listes déroulantes

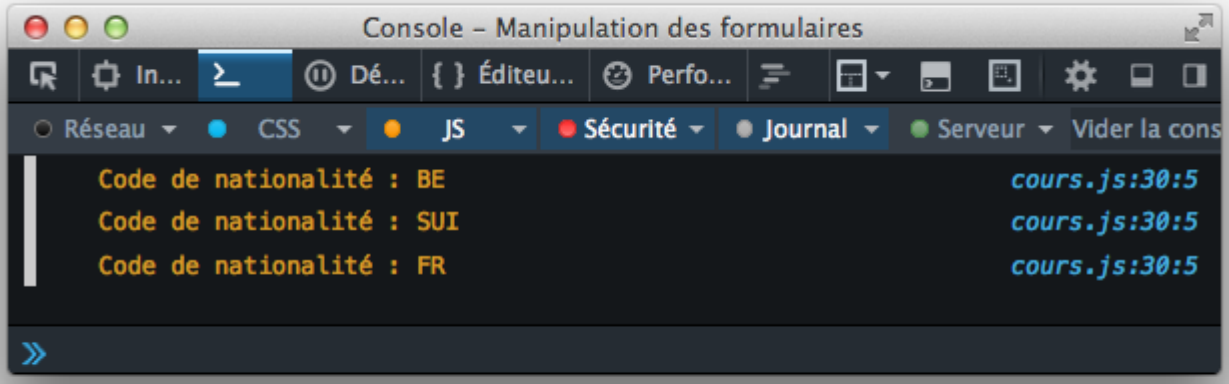
On construit une liste déroulante au moyen d'une balise`<select>` dans laquelle on ajoute une balise`<option>` par choix possible dans la liste.

Voici l'extrait du formulaire initial qui crée une liste déroulante contenant trois nationalités.

```
<label for="nationalite">Nationalité :</label>
<select name="nationalite" id="nationalite">
    <option value="FR" selected>Française</option>
    <option value="BE">Belge</option>
    <option value="SUI">Suisse</option>
    <option value="XX">Autre</option>
</select>
```

L'exemple de code suivant exploite les événements de type`change`déclenchés sur la liste déroulante pour afficher le nouveau choix (attribut`value`de la balise`<option>`associée au choix).

```
// Affichage du code de nationalité choisi
document.getElementById("nationalite").addEventListener("change", function (e) {
    console.log("Code de nationalité : " + e.target.value);
});
```



Comme pour les boutons radio, la propriété`e.target.value` de l'événement`change` contient la valeur de l'attribut`value`de la balise`<option>` associé au nouveau choix, et non pas le texte affiché dans la liste déroulante.

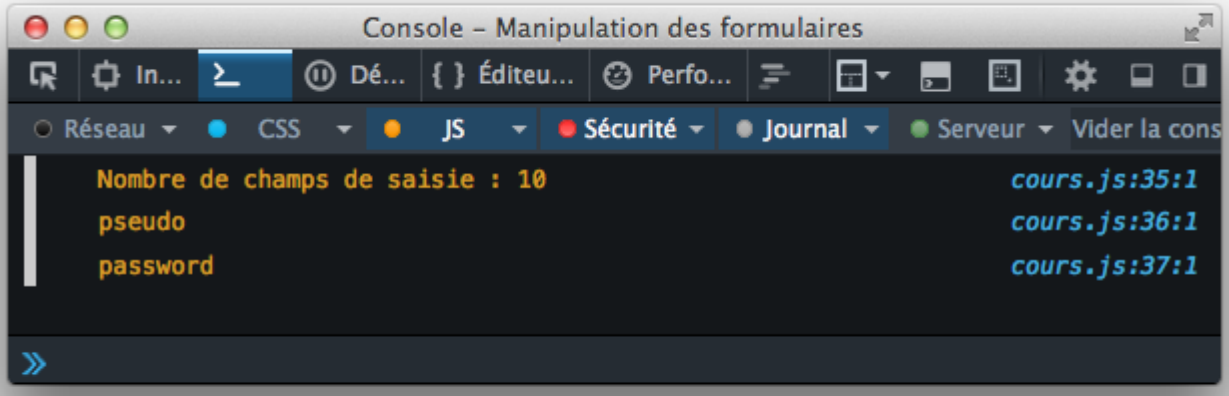
Le formulaire comme élément du DOM

Accès aux champs du formulaire

La balise`<form>` définissant un formulaire correspond à un élément du DOM. Cet élément possède une propriété `éléments` rassemblant les champs de saisie du formulaire. On peut l'utiliser pour accéder à un champ à partir de son nom (attribut`name`) ou à partir de son indice dans (ordre d'apparition dans le formulaire).

L'exemple ci-dessous affiche quelques informations sur les champs de saisie du formulaire initial.

```
var form = document.querySelector("form");
console.log("Nombre de champs de saisie : " + form.elements.length); // Affiche 10
console.log(form.elements[0].name); // Affiche "pseudo"
console.log(form.elements.mdp.type); // Affiche "password"
```



Soumission du formulaire

Un formulaire est soumis lorsque l'utilisateur clique sur le bouton d'envoi. Ce bouton correspond à une balise`<input type="submit">`. Une balise`<input type="reset">` affiche un bouton qui réinitialise les données du formulaire.

Voici l'extrait du formulaire initial qui affiche deux boutons dans le formulaire.

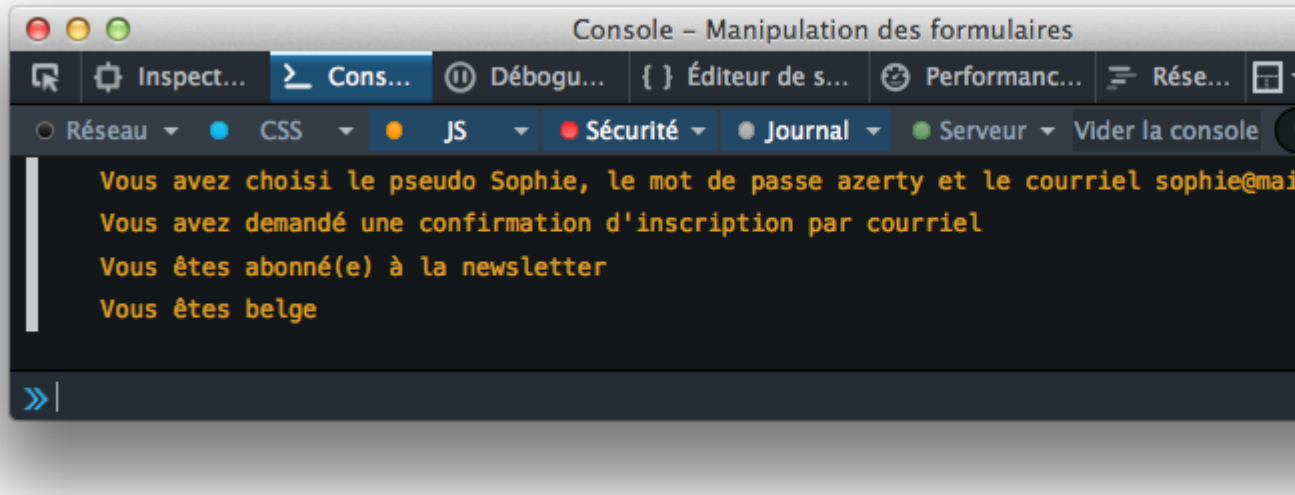
```
<input type="submit" value="Envoyer">
<input type="reset" value="Annuler">
```

En règle générale, la soumission d'un formulaire se traduit par l'envoi de ses données à la ressource identifiée par l'attribut`action`de la balise `<form>`. Mais avant cela, un événement de type`submit`est déclenché sur l'élément du DOM correspondant au formulaire. En ajoutant un gestionnaire pour ce type d'événement, on peut accéder aux données du formulaire avant leur envoi. On peut même annuler l'envoi ultérieur des données en appelant la méthode`preventDefault`sur l'objet`Event`associé à l'événement.

L'exemple de code ci-dessous affiche dans la console l'ensemble des informations saisies ou choisies par l'utilisateur, puis utilise la méthode `preventDefault`pour annuler l'envoi des données du formulaire.

```
// Affiche de toutes les données saisies ou choisies
form.addEventListener("submit", function (e) {
    var pseudo = form.elements.pseudo.value;
    var mdp = form.elements.mdp.value;
    var courriel = form.elements.courriel.value;
    console.log("Vous avez choisi le pseudo " + pseudo + ", le mot de passe " +
        mdp + " et le courriel " + courriel);
    if (form.elements.confirmation.checked) {
        console.log("Vous avez demandé une confirmation d'inscription par courriel");
    } else {
        console.log("Vous n'avez pas demandé de confirmation d'inscription par courriel");
    }
```

```
    switch (form.elements.abonnement.value) {
    case "abonewspromo":
        console.log("Vous êtes abonné(e) à la newsletter et aux promotions");
        break;
    case "abonews":
        console.log("Vous êtes abonné(e) à la newsletter");
        break;
    case "aborien":
        console.log("Vous n'êtes abonné(e) à rien");
        break;
    default:
        console.log("Erreur : code d'abonnement non reconnu");
    }
    switch (form.elements.nationalite.value) {
    case "FR":
        console.log("Vous êtes français(e)");
        break;
    case "BE":
        console.log("Vous êtes belge");
        break;
    case "SUI":
        console.log("Vous êtes suisse");
        break;
    default:
        console.log("Erreur : code de nationalité non reconnu");
    }
    e.preventDefault(); // Annulation de l'envoi des données
});
```



Validation des données saisies

Contrôler les données saisies par l'utilisateur avant de les envoyer à un serveur est l'un des grands intérêts de l'utilisation de JavaScript avec les formulaires web. Ainsi, on peut signaler immédiatement d'éventuelles erreurs de saisie, ce qui améliore l'expérience de l'utilisateur. On évite également des allers-retours réseau coûteux en temps.

Le contrôle de validité peut se faire de plusieurs manières, éventuellement combinables :

- au fur et à mesure de la saisie d'une donnée ;
- à la fin de la saisie d'une donnée ;
- au moment où l'utilisateur soumet le formulaire.

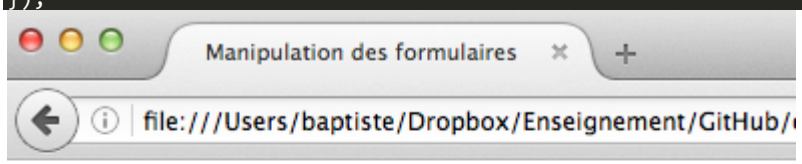
Cette dernière technique consiste simplement à ajouter des vérifications dans le gestionnaire des événements`submit`sur le formulaire. Nous allons étudier de plus près les deux autres.

Validation pendant la saisie

La validation pendant la saisie repose sur l'exploitation des événements`input`, qui se déclenchent sur une zone de saisie à chaque fois que sa valeur est modifiée.

L'exemple de code suivant ajoute un gestionnaire pour les événements`input`sur la zone de saisie du mot de passe. Ce gestionnaire vérifie la longueur (nombre de caractères) du mot de passe saisi et affiche à l'utilisateur un message ayant un contenu et une couleur appropriés.

```
// Vérification de la longueur du mot de passe saisi
document.getElementById("mdp").addEventListener("input", function (e) {
    var mdp = e.target.value; // Valeur saisie dans le champ mdp
    var longueurMdp = "faible";
    var couleurMsg = "red"; // Longueur faible => couleur rouge
    if (mdp.length >= 8) {
        longueurMdp = "suffisante";
        couleurMsg = "green"; // Longueur suffisante => couleur verte
    } else if (mdp.length >= 4) {
        longueurMdp = "moyenne";
        couleurMsg = "orange"; // Longueur moyenne => couleur orange
    }
    var aideMdpElt = document.getElementById("aideMdp");
    aideMdpElt.textContent = "Longueur : " + longueurMdp; // Texte de l'aide
    aideMdpElt.style.color = couleurMsg; // Couleur du texte de l'aide
});
```



Formulaire d'inscription

Pseudo : MonPseudo

Mot de passe : ••• Longueur : faible

Courriel : utilisateur@domaine.fr

Validation à la fin de la saisie

La fin de la saisie dans une zone de texte correspond à la perte du focus par cette zone, ce qui déclenche l'apparition d'un événement de type`blur`que l'on peut exploiter pour contrôler la données saisie.

Imaginons par exemple que l'on veuille contrôler le courriel saisi par l'utilisateur de notre formulaire. Dans un premier temps, on souhaite vérifier uniquement la présence du caractère`@`dans le courriel saisi. Pour cela, on peut utiliser la méthode JavaScript indexOf, qui permet de chercher une valeur dans une chaîne de caractères et renvoie la valeur -1 si cette valeur n'est pas trouvée.

Voici le code JavaScript associé à cette vérification.

```
// Contrôle du courriel en fin de saisie
document.getElementById("courriel").addEventListener("blur", function (e) {
    var validiteCourriel = "";
    if (e.target.value.indexOf("@") === -1) {
        // Le courriel saisi ne contient pas le caractère @
        validiteCourriel = "Adresse invalide";
    }
    document.getElementById("aideCourriel").textContent = validiteCourriel;
});
```

Utilisation d'une expression régulière

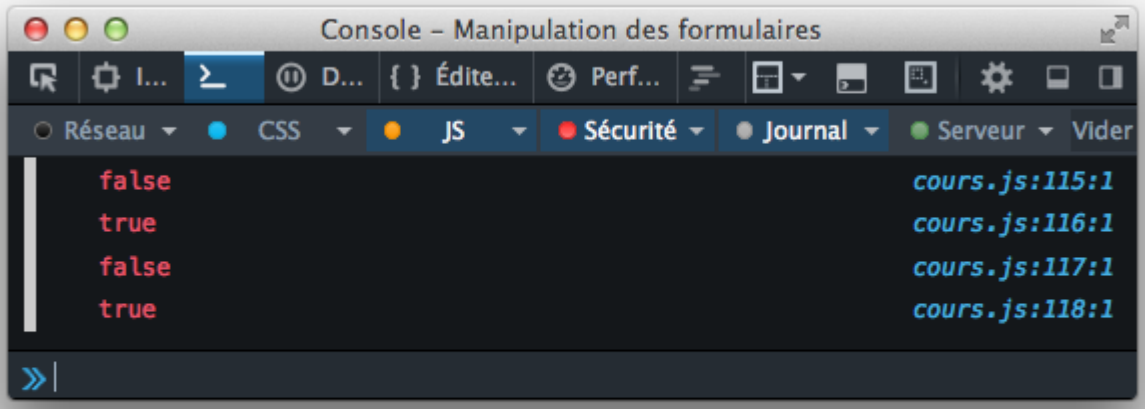
Notre vérification précédente est pour le moins primitive : il ne suffit pas qu'une chaîne contienne au moins une fois le caractère`@`pour en faire une adresse de courriel valide. Il est possible de définir des critères de contrôle beaucoup plus stricts en utilisant ce qu'on appelle une expression régulière.

Une **expression régulière**, également appelée expression rationnelle, définit un motif auquel on va comparer des chaînes de caractères pour trouver (ou non) des correspondances. La plupart des langages de programmation permettent d'exploiter les expression régulières. Elles forment

une sorte de langage à part, déconcertant au départ mais qui permet de répondre à bien des besoins.

Ce paragraphe ne constitue qu'une courte introduction au vaste monde des expressions régulières.
Ce qui suit va sans doute vous paraître un peu difficile à digérer pour le moment. Ne vous en faites pas, c'est en les utilisant qu'on apprend à maîtriser les expressions régulières, et vous avez tout le temps pour cela !
Pour commencer, essayons d'exprimer avec une expression régulière la même chose que ce que nous avons écrit plus haut avec la méthode`indexOf`: la présence du caractère`@`dans une chaîne. Voici le code JavaScript associé.

```
var regex = /@/; // La chaîne doit contenir le caractère @
console.log(regex.test("")); // Affiche false
console.log(regex.test("@")); // Affiche true
console.log(regex.test("sophie&mail.fr")); // Affiche false
console.log(regex.test("sophie@mail.fr")); // Affiche true
```



On définit une expression régulière JavaScript en plaçant son motif entre deux caractères`/`. La variable ainsi créée est un objet. Sa méthode`test` détecte la présence d'une correspondance avec le motif dans la chaîne de caractères passée en paramètre, et renvoie`true` ou`false`selon le cas.

Le tableau suivant ne présente que quelques-unes des très nombreuses possibilités offertes pour définir le motif d'une expression régulière.

Motif	Correspondance si	test() === true	test() === false
`abc`	La chaîne contient "abc"	abc, abcdef, 123abc456	abdc, 1bca, adbc, ABC
`[abc]`	La chaîne contient soit "a", soit "b", soit "c"	abc, daef, bbb, 12c34	def, xyz, 123456, BBB
`[a-z]`	La chaîne contient n'importe quelle lettre minuscule de l'alphabet	abc, 12f43, _r_	123, ABC, _-_
`[0-9]` ou `\d`	La chaîne contient un chiffre	123, ab4c, a56	abc
`a.c`	La chaîne contient "a" suivi d'un caractère (n'importe lequel) suivi de "c"	abc, acc, 12a.c34	ac, abbc, ABC
`a\.c`	La chaîne contient "a.c"	a.c, a.cdef, 12a.c34	ac, abc
`a.+c`	La chaîne contient "a" suivi d'un ou plusieurs caractères (n'importe lesquels) suivi de "c"	abc, abbc, 12a$ùc34	ac,bbc
`a.*c`	La chaîne contient "a" suivi de zéro ou plusieurs caractères (n'importe lesquels) suivi de "c"	abc, abbc, ac	ABC, bbc

L'observation de ce tableau nous conduit aux déductions suivantes :

- Les crochets `[]` définissent un intervalle de caractères. Toute chaîne contenant au moins un caractère dans cet intervalle correspondra au motif. Les motifs `[a-z]` et `[A-Z]` permettent de rechercher n'importe quelle lettre de l'alphabet, respectivement en minuscules ou en majuscules.
- Les motifs équivalents `[0-9]` et `\d` permettent de rechercher un chiffre.
- Le caractère point `.` permet de remplacer n'importe quel caractère.
- Le caractère `\` ("antislash" ou "backslash") indique que le caractère qui suit doit être recherché en tant que tel. Par exemple, `\.` permet de rechercher le caractère `.`.
- Le caractère `+` permet de rechercher une ou plusieurs occurrences de l'expression qui le précède.

- Le caractère `*` permet de rechercher zéro ou plusieurs occurrences de l'expression qui le précède.

Le site https://regex101.com permet de tester facilement ses expressions régulières. N'hésitez pas à l'utiliser pour mieux comprendre leur fonctionnement.
Revenons maintenant à la vérification du courriel saisi dans notre formulaire. Parmi les nombreuses expressions régulières utilisables pour réaliser cette vérification, je vous propose d'utiliser la suivante : `/.+@.+\..+/`.

Avant d'aller plus loin, essayez de décoder ce motif et de trouver quelles conditions doit remplir une chaîne de caractères pour lui correspondre.

Vous avez bien cherché ? Allez, voici la réponse : ce motif représente une chaîne qui :

- Commence par un ou plusieurs caractères (`.+`)
- Contient ensuite le caractère @ (`@`)
- Contient ensuite un ou plusieurs caractères (`.+`)
- Contient ensuite le caractère `.` (`\.`)
- Finit par un ou plusieurs caractères (`.+`)

Autrement dit, une chaîne devra être de la forme xxx@yyy.zzz pour correspondre à ce motif. Cela ne suffit pas à filtrer efficacement toutes les saisies, mais c'est tout de même un net progrès par rapport à la solution précédente.

Voici comment mettre en oeuvre cette technique pour notre exemple. Modifiez le gestionnaire de l'événement `blur` sur la zone de saisie du mot de passe de la manière suivante.

```
// Contrôle du courriel en fin de saisie
document.getElementById("courriel").addEventListener("blur", function (e) {
    // Correspond à une chaîne de la forme xxx@yyy.zzz
    var regexCourriel = /.+@.+\..+/;
    var validiteCourriel = "";
    if (!regexCourriel.test(e.target.value)) {
        validiteCourriel = "Adresse invalide";
    }
    document.getElementById("aideCourriel").textContent = validiteCourriel;
});
```