JavaScript

Variables

Les variables sont des conteneurs dans lesquels on peut stocker des valeurs. Pour commencer, il faut déclarer une variable avec le mot-clé `let` en le faisant suivre de son nom:
exemple : `let myVariable;`

Une fois une variable déclarée, vous pouvez lui donner une valeur :
`myVariable = 'Bob';  ou alors sur la même ligne let myVariable = 'Bob';`

Notez que les variables peuvent contenir des types différents de données:

Variable	Explication	Exemple
Chaîne de caractères	Une suite de caractères connue sous le nom de chaîne. Pour indiquer que la valeur est une chaîne, il faut la placer entre guillemets.	`let myVariable = 'Bob';`
Nombre	Un nombre. Les nombres ne sont pas entre guillemets.	`let myVariable = 10;`
Booléen	Une valeur qui signifie vrai ou faux. `true`/`false` sont des mots-clés spéciaux en JS, ils n'ont pas besoin de guillemets.	`let myVariable = true;`
Tableau	Une structure qui permet de stocker plusieurs valeurs dans une seule variable.	`let myVariable = [1,'Bob','Étienne',10];` Référez-vous à chaque élément du tableau ainsi : `myVariable[0]`, `myVariable[1]`, etc.
Objet	À la base de toute chose. Tout est un objet en JavaScript et peut être stocké dans une variable. Gardez cela en mémoire tout au long de ce cours.	`let myVariable = document.querySelector('h1');` tous les exemples au dessus sont aussi des objets.

Opérateurs

Un opérateur est un symbole mathématique qui produit un résultat en fonction de deux valeurs (ou variables). Le tableau suivant liste certains des opérateurs les plus simples ainsi que des exemples que vous pouvez tester dans votre console JavaScript.

Opérateur	Explication	Symbole(s)	Exemple
Addition	Utilisé pour ajouter deux nombres ou concaténer (accoler) deux chaînes.	`+`	`6 + 9;` `"Bonjour " + "monde !";`

Soustraction, multiplication, division	Les opérations mathématiques de base.	`-`, `*`, `/`	`9 - 3;`
			`8 * 2; // pour multiplier, on utilise un astérisque`
			`9 / 3;`

Assignation	On a déjà vu cet opérateur : il affecte une valeur à une variable.	`=`	`let myVariable = 'Bob';`
Égalité	Teste si deux valeurs sont égales et renvoie un booléen `true`/`false` comme résultat.	`===`	`let myVariable = 3;` `myVariable === 4;`

Négation , N'égale pas	Renvoie la valeur logique opposée à ce qu'il précède ; il change `true` en `false`, etc. Utilisé avec l'opérateur d'égalité, l'opérateur de négation teste que deux valeurs *ne sont pas* égales.	`!`, `!==`	L'expression de base est vraie, mais la comparaison renvoie `false` parce que nous la nions :
			`let myVariable = 3;`
			`!(myVariable === 3);`
			On teste ici que "`myVariable`n'est PAS égale à 3". Cela renvoie `false`, car elle est égale à 3.
			`let myVariable = 3;`
			`myVariable !== 3;`

Structures conditionnelles

Les structures conditionnelles sont des éléments du code qui permettent de tester si une expression est vraie ou non et d'exécuter des instructions différentes selon le résultat. La structure conditionnelle utilisée la plus fréquemment est `if ... else`.

Par exemple :

```
let iceCream = 'chocolat';
if (iceCream === 'chocolat') {
  alert("J'adore la glace au chocolat !");
} else {
  alert("Ooooh, mais j'aurais préféré au chocolat.");
}
```

L'expression contenue dans `if ( ... )` correspond au test — Ici, on utilise l'opérateur d'identité (décrit plus haut) pour comparer la variable `iceCream` avec la chaîne de caractères `chocolat` pour voir si elles sont égales. Si cette comparaison renvoie `true`, le premier bloc de code sera exécuté. Sinon, le premier bloc est sauté et c'est le code du second bloc, celui présent après `else`, qui est exécuté.

Fonctions

Les fonctions sont un moyen de compacter des fonctionnalités en vue de leur réutilisation. Quand vous avez besoin de la procédure, vous pouvez appeler une fonction, par son nom, au lieu de ré écrire la totalité du code chaque fois. Nous avons déjà utilisé des fonctions plus haut, par exemple :

```
1.let myVariable = document.querySelector('h1');
2.alert('Bonjour !');
```

Ces fonctions (`querySelector` et `alert`) sont disponibles dans le navigateur et vous pouvez les utiliser où bon vous semble.
Si vous voyez quelque chose qui ressemble à un nom de variable et qui est suivi de parenthèses — `()`—, c'est probablement une fonction. Les fonctions prennent souvent des arguments — bouts de données dont elles ont besoin pour faire leur travail. Ils sont placés entre parenthèses, séparés par des virgules s'il y en a plusieurs.
Par exemple, la fonction `alert()` fait apparaître une fenêtre de pop-up dans la fenêtre du navigateur, mais vous devez donner une chaîne comme argument pour indiquer à la fonction ce que l'on souhaite écrire dans cette fenêtre.
La bonne nouvelle est que vous pouvez définir vos propres fonctions —

par exemple, vous pouvez écrire une fonction toute simple qui prend deux arguments et les multiplie :

```
function multiply(num1,num2) {
  let result = num1 * num2;
  return result;
}
```

Vous pouvez déclarer cette fonction dans la console avant de l'utiliser plusieurs fois :

```
multiply(4, 7);
multiply(20, 20);
multiply(0.5, 3);
```

Événements

Pour qu'un site web soit vraiment interactif, vous aurez besoin d'événements. Les événements sont des structures de code qui « écoutent » ce qui se passe dans le navigateur et déclenchent du code en réponse. Le meilleur exemple est l'événement cliquer, déclenché par le navigateur quand vous cliquez sur quelque chose avec la souris. À titre de démonstration, saisissez ces quelques lignes dans la console, puis cliquez sur la page en cours :

```
document.querySelector('html').addEventListener('click', function() {
    alert('Aïe, arrêtez de cliquer !!');
});
```

Il existe plein de méthodes pour « attacher » un événement à un élément. Dans cet exemple, nous avons sélectionné l'élément HTML concerné et nous avons défini un gestionnaire `onclick` qui est une propriété qui est égale à une fonction anonyme (sans nom) qui contient le code à exécuter quand l'utilisateur clique.
On pourra noter que :

```
document.querySelector('html').addEventListener('click', function()
{});
```

est équivalent à :

```
let myHTML = document.querySelector('html');
myHTML.addEventListener('click', function() {});
```

La première syntaxe est simplement plus courte.

Autre fonctionnalité:

Ajouter un changeur d'image

Dans cette section, nous allons incorporer une autre image au site en utilisant quelques fonctionnalités de l'API DOM et un peu de JavaScript pour alterner entre les deux images lorsqu'on clique sur l'image affichée.

1.Tout d'abord, trouvez une autre image à afficher sur le site. Assurez-vous qu'elle soit de même taille que la première, ou le plus possible approchante.

2.Enregistrez cette image dans votre dossier `images`.

3.Renommez cette image « firefox2.png » (sans les guillemets).

4.Dans le fichier `main.js`, entrez ce code JavaScript (si votre code « Bonjour, monde » est toujours là, supprimez- le) :

```
let myImage = document.querySelector('img');

myImage.addEventListener('click', function() {
    let mySrc = myImage.getAttribute('src');
    if (mySrc === 'images/firefox-icon.png') {
      myImage.setAttribute('src', 'images/firefox2.png');
    } else {
      myImage.setAttribute('src', 'images/firefox-icon.png');
    }
});
```

5.Sauvegardez tous les fichiers puis chargez `index.html` dans le navigateur. Maintenant, si vous cliquez sur l'image, elle doit changer pour l'autre !

Dans cet exemple, nous utilisons une référence vers l'élement `<img>` grâce à la variable `myImage`. Ensuite, nous égalons la propriété du gestionnaire d'événement `onclick` de cette variable à une fonction sans nom (une fonction anonyme). Maintenant chaque fois que cet élément est cliqué :

1.nous récupèrons la valeur de l'attribut `src` de l'image.
2.nous utilisons une structure conditionnelle pour voir si la valeur de `src` est égale au chemin de l'image originale :
1.si c'est le cas, nous changeons la valeur de `src` et indiquons le chemin vers la seconde image, forçant le chargement de cette image dans l'élément `<img>`.
2.si ce n'est pas le cas (ce qui signifie que l'image a déjà été changée), la valeur de `src` revient à sa valeur initiale.

Ajouter un message d'accueil personnalisé

Continuons en ajoutant encore un peu de code pour changer le titre de la page afin d'inclure un message d'accueil personnalisé pour le visiteur du site. Ce message d'accueil sera conservé quand l'utilisateur quittera le site et s'il y revient — nous le sauvegarderons avec l'API Web Storage. Nous ajouterons également une option pour pouvoir changer l'utilisateur et le message d'accueil si besoin.
1.Dans le fichier `index.html`, ajoutons la ligne suivante avant l'élément `<script>` :

```
<button>Changer d'utilisateur</button>
```

2.Dans le fichier `main.js`, ajoutons le code suivant à la fin du fichier. Cela fait référence au nouveau bouton ajouté et à l'élément de titre puis enregistrons ces références dans des variables :

```
let myButton = document.querySelector('button');
let myHeading = document.querySelector('h1');
```

3.Ajoutons maintenant les fonctionnalités pour avoir ce message d'accueil personnalisé (cela ne servira pas immédiatement mais un peu plus tard) :

```
function setUserName() {
  let myName = prompt('Veuillez saisir votre nom.');
  localStorage.setItem('nom', myName);
  myHeading.textContent = 'Mozilla est cool, ' + myName;
}
```

Cette fonction utilise la fonction `prompt()` qui affiche une boîte de dialogue, un peu comme `alert()` sauf qu'elle permet à l'utilisateur de saisir des données et de les enregistrer dans une variable quand l'utilisateur clique sur OK. Dans notre exemple, nous demandons à l'utilisateur de saisir son nom. Ensuite, nous appelons une API appelée `localStorage`. Cette API permet de stocker des données dans le navigateur pour pouvoir les réutiliser ultérieurement. Nous utilisons la fonction `setItem()` de cette API pour stocker la donnée qui nous intéresse dans un conteneur appelé `'nom'`. La valeur

stockée ici est la valeur de la variable `myName` qui contient le nom saisi par l'utilisateur. Enfin, on utilise la propriété `textContent` du titre pour lui affecter un nouveau contenu.
4.Ajoutons ensuite ce bloc `if ... else`. Ce code correspond à l'étape d'initialisation car il sera utilisé la première fois que la page est chargée par l'utilisateur :

```
if (!localStorage.getItem('nom')) {
  setUserName();
} else {
  let storedName = localStorage.getItem('nom');
  myHeading.textContent = 'Mozilla est cool, ' + storedName;
}
```

Ce bloc utilise l'opérateur de négation (NON logique, représenté avec le !) pour vérifier si le navigateur possède une donnée enregistrée appelée `nom`. Si non, la fonction `setUserName()` est appelée pour créer cette donnée. Si elle existe (ce qui correspond au cas où l'utilisateur est déjà venu sur la page), on la récupère avec la méthode `getItem()` et on définit le contenu de `textContent` pour le titre avec une chaîne suivie du nom de l'utilisateur, comme nous le faisons dans `setUserName()`.

5.Enfin, on associe le gestionnaire `onclick` au bouton. De cette façon, quand on clique sur le bouton, cela déclenchera l'exécution de la fonction `setUserName()`. Ce bouton permet à l'utilisateur de modifier la valeur s'il le souhaite:

```
myButton.addEventListener('click', function() {
  setUserName();
});
```

Récapitulons : la première fois que l'utilisateur viste le site, il sera invité à saisir un nom d'utilisateur. Ce nom sera utilisé pour afficher un message d'accueil personnalisé. Si l'utilisateur souhaite changer son nom, il peut cliquer sur le bouton. En supplément, le nom est enregistré pour plus tard grâce à l'API `localStorage`, cela permet à l'utilisateur de retrouver son nom, même s'il a fermé la page et/ou le navigateur et qu'il revient plus tard !