Javascript

Les variables

Une variable consiste en un espace de stockage, qui permet de garder en mémoire une valeur lors de l'exécution d'un script. Elles sont essentielles au bon fonctionnement de nos algorithme

Les types de variable

Il est possible de stocker différents types d'information dans une variable.

Une variable peut être déclarée à l'aide du mot clef **var** ou **let** et ne doivent pas contenir de caractères spéciaux à l'exception des _.

```
// chaîne de caractères
var text =  'J\'écris mon texte ici';

//Numérique
var number1 = 2;
var number2 = 3.4123;
var number3 = -509;

//booléens
var majeur = true;
var mineur = false;
```

Les types de variable

Une variable peut enfin être de type booléen (boolean), avec deux états possibles : vrai ou faux (true ou false), des tableaux et des objets.

Les opérateurs

Un opérateur est un symbole mathématique qui produit un résultat en fonction de plusieurs valeurs (la plupart du temps on utilise deux valeurs et un opérateur). Le tableau suivant liste certains des opérateurs les plus simples ainsi que des exemples que vous pouvez tester dans votre console JavaScript.

Les opérateurs

Opérateur	Explication	Symbole(s)	Exemple
Somme / Concaténation	Il peut être utilisé pour calculer la somme de deux nombres ou pour concaténer (coller) deux chaînes ensemble.	`+`	`6 + 9;` `"Coucou " + "monde !";`

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

Opérateur d'affectation	On a déjà vu cet opérateur : il permet d'affecter une valeur à une variable.	`=`	`let maVariable = 'Bob';`
Opérateur d'identité	Il permet de tester si deux valeurs sont égales et il renvoie un booléen `true`/`false` comme résultat.	`===`	`let maVariable = 3;` `maVariable === 4;`

Opérateur de négation et opérateur d'inégalité	Souvent utilisé avec l'opérateur d'égalité, l'opérateur de négation est l'équivalent, en JavaScript, d'un NON logique (il transforme la valeur `true` en `false` et vice versa)	`!`, `!==`	`let myVariable = 3;`
			`!(myVariable === 3);`
			
			On teste ici "`maVariable` n'est PAS égale à 3". Cela renvoie `false`, car elle est égale à 3.
			
			`let maVariable = 3;`
			`maVariable !== 3;`

Les fonctions alert() et prompt()

alert: Affiche un dialogue d'alerte contenant le texte spécifié.

prompt:Affiche un dialogue avec un message demandant à l'utilisateur d'entrer une réponse sous forme de texte.

```
<script type="text/javascript">

    var userName = prompt('Entrez votre prénom :');//Demande le prénom de l'utilisateur
    alert(userName); // Affiche le prénom entré par l'utilisateur

</script>
```

Exercice 1(Voir TP)

La fonction confirm()

confirm: Affiche un dialogue modal avec un message et deux boutons, OK et Annuler.

```
<script type="text/javascript">

    confirm('Voulez-vous exécuter le code Javascript de cette page ?')

</script>
```

L’instruction if

L'instruction **if** exécute une instruction si une condition donnée est vraie. Si la condition n'est pas vérifiée, il est possible d'utiliser une autre instruction.

```
if (condition1)
   instruction1
else if (condition2)
   instruction2
else if (condition3)
   instruction3
...
else
   instructionN
```

Si on indente correctement le code, on retrouve la structure exactement équivalente :

```
if (condition1)
   instruction1
else
   if (condition2)
      instruction2
   else
      if (condition3)
...
```

L’instruction if

```
<script>
    if (confirm('Pour accéder à ce site vous devez être une fille, ' +
            'cliquez sur "OK" si c\'est le cas.')) {
        alert('Vous allez être redirigé vers le site.');
    }
    else {
        alert("Désolé, vous n'avez pas accès à ce site.");
    }
</script>
```

Exercice 2(Voir TP)

La boucle while

La boucle **while** permet d'exécuter un code tant que la condition passée en paramètre n'est pas satisfaite

```
while (condition) instruction
```

```
<script>
    var number = 1;
    while (number < 10) {
        number++; // Tant que le nombre est inférieur à 10, on l'incrémente de 1
    }
</script>
```

Exercice 3, 4 et 5 (Voir TP)

La boucle for

La boucle for permet d'exécuter un code un certain nombre de fois en précisant manuellement l'intervalle pour lequel on souhaite faire la boucle. Elle présente une notation plus concise que le while :

```
for ([initialisation]; [condition]; [expression_finale])
   instruction
```

```
<p id="colourList"></p>
<script type="text/javascript">
    var colours = ["Red", "Yellow", "Blue"];
    var text = "";
    var i;
    for (i = 0; i < colours.length; i++) {
        text += colours[i] + " ";
    }
    document.getElementById("colourList").innerHTML = text;
</script>
```

Exercice 3, 4 et 5 avec la boucle for (Voir TP)

Les fonctions

La déclaration **function** permet de définir une fonction et les paramètres que celle-ci utilise.

```
function nom([param1,[, param2,[..., paramN]]]) {
   [instructions]
}
```

```
<script>
    function sayHello(name) {
        return 'Bonjour ' + name;
    }
    alert(sayHello('Bamba'));
</script>
```

Les fonctions

Les fonctions anonymes

Elles supposent la structure suivante, sans nom :

```
<script>
    function(arguments) {
    // Le code de votre fonction anonyme
    }
</script>
```

```
<script>
    var sayHello = function(name) {
        return 'Bonjour ' + name;
    };
    alert(sayHello('Bamba'));
</script>
```

Les Fonctions

IIFE pour *Immediately Invoked Function Expression* ou expression de fonction immédiatement appelée

On peut utiliser une expression de fonction pour créer une « IIFE », c'est-à-dire une expression de fonction qu'on appelle dès sa définition :

```
<script>
    var a = "coucou";
    var b = "wa tdsi";

    // IIFE
    (function(x, y) {
        console.log(x + " " + y);
    })(a, b);
</script>
```

Les fonctions

Une fonction peut être stockée dans la propriété d'un objet.

Notions de visibilité

toute variable déclarée dans une fonction n'est utilisable que dans cette même fonction. Ces variables spécifiques à une seule fonction ont un nom : les variables locales. Déclarées en dehors des fonction, on parle de variables globales.

```
<script>
    var message = 'Ici la variable globale !';
    function showMsg() {
        var message = 'Ici la variable locale !';
        alert(message); }
    showMsg(); // On affiche la variable locale
    alert(message); // Puis la variable globale
</script>
```

Javascript côté navigateur

Dom

Le DOM (Document Object Model) est une interface de programmation (ou API, Application Programming Interface) pour les documents XML et HTML. Via le Javascript, le DOM permet d'accéder au code du document ; on va alors pouvoir modifier des éléments du code HTML.

L’objet window

L'objet window représente la fenêtre elle-même. La propriété document d'un objet window pointe vers le document DOM.

Cet objet contient un ensemble de méthode et de propriétés utiles.

```
<script type="text/javascript">
    alert('Hello TDSI !');
    window.alert('Hello TDSI !');
</script>
```

```
<script type="text/javascript">
  window.setTimeout(function () {
      // Ce code sera éxécuté une fois au bout de 3 secondes (3000ms)
      console.log('code éxécuté aprés 3 secondes')
 }, 3000)
</script>
```

L’objet window

```
<script>
    // Affiche une alerte
    window.alert('Ooops');
    // Affiche une fenêtre de confirmation et renvoie un booleen
    var a = window.confirm('Sûr de sûr ?');
    // Affiche un champs qui permet de rentrer une valeur
    var nom = window.prompt('Entrez votre nom')
</script>
```

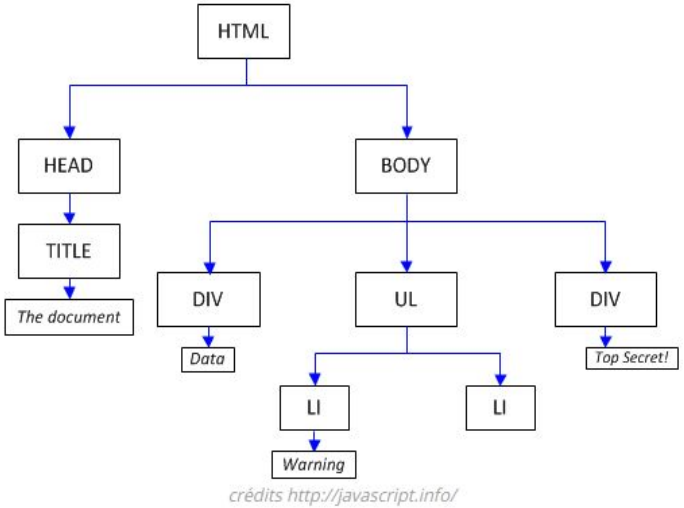
```
<script>
    window.setInterval(function () {
        // Ce code sera appellé toutes les secondes (1000ms)
    }, 1000);

    window.setTimeout(function () {
        // Ce code sera éxécuté une fois au bout de 3 secondes (3000ms)
    }, 3000);
</script>
```

L’objet document

En plus de l'objet window on a aussi accès à un objet document qui permet de récupérer des éléments HTML et de les manipuler.

Un document HTML n'est au final qu'un arbre d'éléments HTML et texte qu'il est possible de parcourir de différentes manières. Cet arbre est appellé le DOM.



crédits http://javascript.info/

L’objet document

Method	Description
document.getElementById(*id*)	Find an element by element id
document.getElementsByTagName(*name*)	Find elements by tag name
document.getElementsByClassName(*name*)	Find elements by class name

L’objet document

```
<script type="text/javascript">
    var divs = document.getElementsByTagName('div');
    for (var i = 0, c = divs.length ; i < c ; i++) {
        alert('Element n° ' + (i + 1) + ' : ' + divs[i]);
    }
</script>
```

```
<div id="menu">
    <div class="item">
        <span>Élément 1</span>
        <span>Élément 2</span>
    </div>
    <div class="publicite">
        <span>Élément 3</span>
        <span>Élément 4</span>
    </div>
</div>
<div id="contenu">
    <span>Introduction au contenu de la page...</span>
</div>
```

L’objet document

```
<script type="text/javascript">
    var query = document.querySelector('#menu .item span'),
    queryAll = document.querySelectorAll('#menu .item span');
    alert(query.innerHTML); // Affiche : "Élément 1"
    alert(queryAll.length); // Affiche : "2"
    alert(queryAll[0].innerHTML + ' - ' + queryAll[1].innerHTML);
</script>
```

On peut aussi utiliser querySelector(), qui renvoie le premier élément trouvé correspondantau sélecteur CSS spécifié, ou querySelectorAll(), qui renvoie tous les éléments (sous forme detableau) correspondant au sélecteur CSS spécifié entre parenthèses

L’objet document

Method	Description
element.innerHTML = *new html content*	Change the inner HTML of an element
element.attribute = new value	Change the attribute value of an HTML element
element.setAttribute*(attribute, value)*	Change the attribute value of an HTML element
element.style.*property = new style*	Change the style of an HTML element

L’objet document

```
<a id="myLink" href="http://www.seneweb.com">Seneweb</a>

<script type="text/javascript">
    var link = document.getElementById('myLink');
    var href = link.getAttribute('href'); // On récupère l'attribut « href »
    alert(href);
    link.setAttribute('href', 'http://www.dakaractu.com'); // on édite
</script>
```

```
<a id="myLink" href="http://www.seneweb.com">Seneweb</a>

<script type="text/javascript">
    var link = document.getElementById('myLink');
    var href = link.getAttribute('href');
    alert(href);
    link.href = 'http://www.dakaractu.com';
</script>
```

L’objet document

```
<div id="myDiv">
    <p>Un peu de texte <a>et un lien</a></p>
</div>
<script>
    // définir un nouveau contenu
    var div = document.getElementById('myDiv');
    div.innerHTML = 'Je mets une citation à la place du paragraphe';
</script>
```

```
<div id="myDiv">
    <p>Un peu de texte <a>et un lien</a></p>
</div>
<script>
    var div = document.getElementById('myDiv');
    div.innerHTML += 'que je rajoute';
</script>
```

L’objet document

Method	Description
document.createElement(*element*)	Create an HTML element
document.removeChild(*element*)	Remove an HTML element
document.appendChild(*element*)	Add an HTML element
document.replaceChild(*element*)	Replace an HTML element
document.write(*text*)	Write into the HTML output stream

Créer et insérer des éléments

Avec le DOM, l'ajout d'un élément se fait en trois temps : on crée l'élément, on lui affecte des attributs, on l'insère dans le document.

Par exemple, on crée <a> :

```
var newLink = document.createElement('a');
```

On lui affecte des attributs :

```
newLink.id = 'sdz_link';
newLink.href = 'http://blog.crdp-versailles.fr/rimbaud/';
newLink.title = 'Découvrez le blog de la Classe Actu !';
newLink.setAttribute('tabindex', '10');
```

On l'insère dans le document :

```
<div><p id="myP">Un peu de texte <a>et un lien</a></p></div>
<script>
  var newLink = document.createElement('a');
  newLink.id = 'sdz_link';
  newLink.href = 'http://blog.crdp-versailles.fr/rimbaud/';
  newLink.title = 'Découvrez le blog de la Classe Actu !';
  newLink.setAttribute('tabindex', '10');
  document.getElementById('myP').appendChild(newLink); // le nouvel élément
est le dernier enfant dans le paragraphe avec id 'myP'
  var newLinkText = document.createTextNode("Le Tonnerre de Rimbaud");
  newLink.appendChild(newLinkText); // ces deux lignes pour ajouter le texte
</script>
```

Cloner, remplacer, supprimer

Pour cloner un élément, on utilise `cloneNode()`, et on choisit avec (`true`) ou sans (`false`) ses enfants et ses attributs.

Pour remplacer un élément par un autre, on utilise `replaceChild()`, avec deux paramètres, le nouvel élément et l'élément qu'on veut remplacer :

```
<div><p id="myP">Un peu de texte <a>et un lien</a></p></div>
<script>
  var link = document.getElementsByTagName('a')[0];
  var newLabel= document.createTextNode('et un hyperlien');
  link.replaceChild(newLabel, link.firstChild);
</script>
```

Cloner, remplacer, supprimer

Pour supprimer un élément, on utilise `removeChild()`, avec le nœud enfant à retirer :

```
var link = document.getElementsByTagName('a')[0];
link.parentNode.removeChild(link);
```

Pour vérifier la présence d'éléments enfant, on utilise `hasChildNodes()` :

```
<div><p id="myP">Un peu de texte <a>et un lien</a></p></div>
<script>
   var paragraph = document.getElementsByTagName('p')[0];
   alert(paragraph.hasChildNodes()); // Affiche true
</script>
```

Pour insérer un élément avant un autre, on utilise `insertBefore()` :

```
<p id="myP">Un peu de texte <a>et un lien</a></p>
<script>
   var paragraph = document.getElementsByTagName('p')[0];
   var emphasis = document.createElement('em'),
   emphasisText = document.createTextNode(' en emphase légère ');
   emphasis.appendChild(emphasisText);
   paragraph.insertBefore(emphasis, paragraph.lastChild);
</script>
```