INTRODUCTION
AUX LANGAGES DE DÉVELOPPEMENT POUR LE WEB

JAVASCRIPT ET PHP

Jean Clément
Département Sciences de l'information – Paris 8

INTRODUCTION À JAVASCRIPT

Qu'est-ce que le Javascript?

Le Javascript est un langage de script incorporé dans un document HTML. Historiquement il s'agit même du premier langage de script pour le Web. Ce langage est un langage de programmation qui permet d'apporter des améliorations au langage HTML en permettant d'exécuter des commandes du côté client, c'est-à-dire au niveau du navigateur et non du serveur web.
Ainsi le langage Javascript est fortement dépendant du navigateur appelant la page web dans laquelle le script est incorporé, mais en contrepartie il ne nécessite pas de compilateur, contrairement au langage Java, avec lequel il a longtemps été confondu.

Les navigateurs et javascript

JavaScript 1.0 Netscape	Navigator 2.0, Internet Explorer 3.0
JavaScript 1.1	Netscape Navigator 3.0
Javascript 1.2	Netscape Navigator 4.0/4.05, Internet Explorer 4.0
Javascript 1.3	Netscape Navigator 4.06, Internet Explorer 5.0
Javascript 1.4	Netscape Navigator 6.0, Internet Explorer 5.5
Javascript 1.5	Netscape Navigator 6.0

Il ne faut pas confondre JavaScript et Java. En effet contrairement au langage Java, le code Javascript est directement écrit dans la page HTML, c'est un langage peu évolué qui ne permet aucune confidentialité au niveau des codes (ceux-ci sont effectivement visibles).
D'autre part l'applet Java (le programme) a été préalablement compilée, et une machine virtuelle permettant d'interpréter le pseudo-code doit être chargée en mémoire (du côté du client) à chaque chargement de la page, d'où un important ralentissement pour les applets Java contrairement au JavaScript.
Le Javascript est *case sensitive* (en français sensible à la casse), c'est-à-dire qu'il fait une différence entre un nom de variable contenant ou non des majuscules. Ainsi la fonction bonjour(); n'est pas la même fonction que Bonjour();.
Enfin, comme en langage C, chaque instruction se termine par un point-virgule (;).

A quoi ressemble un script?

Un script est une portion de code qui vient s'insérer dans une page HTML. Le code du script n'est toutefois pas visible dans la fenêtre du navigateur car il est compris entre des balises (ou tags) spécifiques qui signalent au navigateur qu'il s'agit d'un script écrit en langage JavaScript.
Les balises annonçant un code Javascript sont les suivantes:

```
<SCRIPT language="Javascript">
Placez ici le code de votre script
<SCRIPT>
```

Masquage du script pour les anciens browsers

Ce code est ainsi invisible du point de vue du navigateur c'est-à-dire que ce dernier n'affiche pas dans sa fenêtre le code Javascript. Toutefois, d'anciens navigateurs, créés avant l'arrivée du Javascript, ne connaissent pas ces balises et donc les ignorent...
Le code de votre Javascript risque donc de s'afficher sur votre belle page web et venir gâcher votre travail. L'astuce consiste donc à ajouter des balises de commentaires à l'intérieur même des balises de script. Ainsi les anciens navigateurs ignoreront tout simplement l'intégralité du script, tandis que les navigateurs récents l'interprèteront (comme il se le doit) comme du Javascript!
Voici ce que donne le script une fois "masqué" pour les anciens navigateurs:

```
<SCRIPT language="Javascript"> <!--
Placez ici le code de votre script
// -->
</SCRIPT>
```

Ajouter des commentaires dans votre code

Comme dans tout langage de programmation, il est bon d'ajouter des commentaires dans les scripts.
Il ne faut pas confondre les balises de commentaires du langage HTML (destinées à masquer le script pour certains browsers) et les caractères de commentaires Javascript (permettant de documenter son script.
Pour écrire des commentaires, Javascript utilise les conventions utilisées en langage C/C++

- Pour mettre en commentaires tout une ligne on utilise le double slash:
  // Tous les caractères derrière le // sont ignorés
- Pour mettre en commentaire une partie du texte (éventuellement sur plusieurs lignes) on utilise le /* et le */:
  /* Toutes les lignes comprises entre ces repères sont ignorées par l'interpréteur code */

Il faut veiller à ne pas imbriquer des commentaires, au risque de provoquer une erreur lors de l'exécution du code!

Un exemple de javascript

Pour commencer nous allons afficher une boîte de dialogue suite au chargement d'une page HTML. Dans ce cas le script est totalement inutile voire ennuyeux pour vos visiteurs... Cet exemple montre

ce que l'abus de Javascript peut donner... Il faudra apprendre à se servir du Javascript avec modération!

```
<HTML>
<HEAD>
<TITLE> Voici une page contenant du Javascript</TITLE>
</HEAD>
<SCRIPT language="Javascript">
<!--
alert("Voici un message d'alerte!");
// -->
</BODY>
</HTML>
```

À quel emplacement insérer le Javascript dans votre page HTML?

Il existe plusieurs façons d'inclure des scripts dans les pages HTML :
• Grâce à la balise `<SCRIPT>`
• En mettant le code dans un fichier
• Grâce aux événements

Dans la balise <SCRIPT>

Le code Javascript peut être inséré où vous le désirez dans votre page Web, vous devez toutefois veiller à ce que le navigateur ait entièrement chargé votre script avant d'exécuter une instruction. En effet, lorsque le navigateur charge votre page Web, il la traite de haut en bas, de plus vos visiteurs (souvent impatients) peuvent très bien interrompre le chargement d'une page, auquel cas si l'appel d'une fonction se situe avant la fonction dans votre page il est probable que cela génèrera une erreur si cette fonction n'a pas été chargée.
Ainsi, on place généralement le maximum d'éléments du script dans la balise d'en-tête (ce sont les éléments située entre les balises `<HEAD>` et `</HEAD>`). Les événements Javascript seront quant à eux placés dans le corps de la page (entre les balises `<BODY>` et `</BODY>`) comme attribut d'une commande HTML.
L'argument de la balise `<SCRIPT>` décrit le langage utilisé. Il peut être "`JavaScript`" "`JavaScript1.1`" "`JavaScript1.2`".
On peut ainsi (en passant un argument différent de "`JavaScript`") utiliser d'autres langages de programmation que celui-ci (par exemple le `VbScript`).
Pour utiliser différentes versions de JavaScript tout en conservant une certaine compatibilité, il suffit de déclarer plusieurs balises `SCRIPT` ayant chacune comme paramètre la version du JavaScript correspondante.

Dans un fichier externe

Il est possible de mettre les codes de JavaScript en annexe dans un fichier (à partir de Netscape 3.0 uniquement). Le code à insérer est le suivant :

```
<SCRIPT LANGUAGE="Javascript" SRC="url/fichier.js"> </SCRIPT>_
```

Où `url/fichier.js` correspond au chemin d'accès au fichier contenant le code en JavaScript, sachant que si celui-ci n'existe pas le navigateur exécutera le code inséré entre les deux balises.

Grâce aux événements

On appelle événement une action de l'utilisateur, comme le clic d'un des boutons de la souris. Le code dans le cas du résultat d'un événement s'écrit :

```
<balise eventHandler="code Javascript à insérer>
```

`eventHandler` (gestionnaire d'événement) représente le nom de l'événement.

Les objets javascript

Le but de cette section n'a pas pour ambition de vous faire connaître la programmation orientée objet (POO) mais de vous donner une idée de ce qu'est un objet, concept nécessaire à la création de scripts Javascript.

Le Javascript traite les éléments qui s'affichent dans votre navigateur comme des objets, c'est à dire des éléments

• classés selon une hiérarchie pour pouvoir les désigner précisément

• auxquels on associe des propriétés.

Cette notion semble floue pour l'instant mais voyons cela sur un exemple concret:

Imaginez un arbre dans un jardin comportant une branche sur laquelle se trouve un nid. On suppose la hiérarchie d'objets est définie comme ceci:

```
jardin
	arbre
		branche
			feuille
			nid
				largeur: 20
				couleur: jaune
				hauteur: 4
		tronc
		racine
	salade
	balançoire
		trapèze
		corde
		nid
			largeur: 15
			couleur: marron
			hauteur: 6
```

Le nid sur l'arbre est donc désigné comme suit:

```
jardin.arbre.branche.nid
```

Contrairement au nid situé sur la balançoire:

```
Jardin.balançoire.nid
```

Imaginez maintenant que l'on veuille changer la couleur du nid (dans l'arbre) pour le peindre en vert, il suffirait de taper une commande du genre:

```
jardin.arbre.nid.couleur=vert
```

C'est donc ainsi que l'on représente les objets en Javascript, à la seule différence que ce n'est pas un jardin qui est représenté sous forme d'objets mais la fenêtre de votre navigateur...

Les objets du navigateur

Javascript divise la page du navigateur en objets, afin de vous permettre d'accéder à n'importe lequel d'entre eux et de pouvoir les manipuler par l'intermédiaire de leurs propriétés.
On commence généralement par l'objet le plus grand (celui contenant tous les autres) puis on descend dans la hiérarchie jusqu'à arriver à l'objet voulu !

- L'objet le plus grand est l'objet fenêtre (les objets en javascript ont leur dénomination en anglais, donc dans le cas présent *window*)
- Dans la fenêtre s'affiche une page, c'est l'objet document
- Cette page peut contenir plusieurs objets, comme des formulaires, des images,…

Pour accéder à un objet il faut donc partir de l'objet le plus grand (l'objet *window*) pour descendre à l'objet que l'on veut atteindre.
Prenez par exemple le bouton (appelé *checkbox*) et le champ de texte d'un formulaire :

- Le formulaire (auquel on a donné le nom « form1 », pour le distinguer des autres formulaires de la page) est repéré par le code suivant:

```
window.document.forms["form1"]
```

- Le bouton checkbox (auquel on a donné le nom « checkbox ») est repéré par le code suivant:

```
window.document.forms["form1"].checkbox
```

- Le champ de texte (auquel on a donné le nom « champ_text ») est repéré par le code suivant:

```
window.document.forms["form1"].champ_text
```

Le bouton checkbox a entre autre une propriété checked, qui retourne la valeur 1 si le bouton est coché, 0 dans le cas contraire.

Essayez cet exemple:

```
<form name="form1">
<br><input type="checkbox" name="checkbox" onClick="ModifChamp();return true;">
<br><input type='TEXT' name='champ_text' value='Essai du javascript'
 size='24'>
</form>
```

Avec la fonction javascript associée au bouton checkbox :

```
<script language="Javascript">
<!--
function ModifChamp()
{
       if (document.forms["form1"].checkbox.checked)
       {document.forms["form1"].champ_text.value='Bouton coché'}
       else
       {document.forms["form1"].champ_text.value='bouton non coché'}
}
// -->
</script>
```

Le champ de texte a par exemple comme propriétés:

• name: le nom du champ de texte
• value: le texte contenu dans le champ
• size: la taille du champ de texte

Le concept de variable

Une variable est un objet repéré par son nom, pouvant contenir des données, qui pourront être modifiées lors de l'exécution du programme.
En Javascript, les noms de variables peuvent être aussi long que l'on désire, mais doivent répondre à certains critères:
• un nom de variable doit commencer par une lettre (majuscule ou minuscule) ou un "_".
• un nom de variables peut comporter des lettres, des chiffres et les caractères _ et & (les espaces ne sont pas autorisés!).
• Les noms de variables ne peuvent pas être des noms appartenant au langage Javascript.
Les noms de variables sont sensibles à la casse (le Javascript fait la différence entre un nom en majuscule et un nom en minuscules), il faut donc veiller à utiliser des noms comportant la même casse!

La déclaration de variables

Le Javascript étant très souple au niveau de l'écriture (à double-tranchant car il laisse passer des erreurs...), la déclaration des variables peut se faire de deux façons:
• soit de façon explicite, en faisant précéder la variable du mot clé var qui permet d'indiquer de façon rigoureuse qu'il s'agit d'une variable:

```
var chaine= "bonjour"
```

• soit de façon implicite, en écrivant directement le nom de la variable suivie du caractère = et de la valeur à affecter:

```
chaine= "bonjour"
```

Le navigateur détermine seul qu'il s'agit d'une déclaration de variable.
Même si une déclaration implicite est tout à fait reconnue par le navigateur, il est tout de même plus rigoureux de déclarer les variables de façon explicite avec le mot var...
Voici un exemple dans lequel deux variables sont déclarées :

```
<SCRIPT language="Javascript">
<!--
var MaVariable;
var MaVariable2 = 3;
MaVariable = 2;
document.write(MaVariable*MaVariable2);
// -->
</SCRIPT>
```

Portée (visibilité) des variables

Selon l'endroit où on déclare une variable, celle-ci pourra être accessible (visible) de n'importe où dans le script ou bien uniquement dans une portion confinée du code, on parle de "portée" d'une variable.

Lorsqu'une variable est déclarée sans le mot clé `var`, c'est-à-dire de façon implicite, elle est accessible de n'importe où dans le script (n'importe quelle fonction du script peut faire appel à cette variable). On parle alors de variable globale.

Lorsque l'on déclare de façon explicite une variable en javascript (en précédant sa déclaration avec le mot `var`), sa portée dépend de l'endroit où elle est déclarée.

• Une variable déclarée au début du script, c'est-à-dire avant toute fonction sera globale, on pourra alors l' utiliser à partir de n'importe quel endroit dans le script.

• Une variable déclarée par le mot clé `var` dans une fonction aura une portée limitée à cette seule fonction, c'est-à-dire qu'elle est inutilisable ailleurs, on parle alors de variable locale

Illustrons ceci par deux exemples :

```
<SCRIPT language="Javascript">
<!--
var a = 12;
var b = 4;
function MultipliePar2(b) {
   var a = b * 2;
   return a;
}
document.write("Le double de ",b," est ",MultipliePar2(b)) ;
document.write("La valeur de a est ",a);
// -->
</SCRIPT>
```

Dans l'exemple ci-dessus, la variable a est déclarée explicitement en début de script, ainsi que dans la fonction. Voici ce qu'affiche ce script :

Le double de 4 est 8

La valeur de a est 12

Voici un autre exemple dans lequel a est déclarée implicitement dans la fonction :

```
<SCRIPT language="Javascript">
<!--
var a = 12;
var b = 4;
function MultipliePar2(b) {
   a = b * 2;
   return a;
}
document.write("Le double de ",b," est ",MultipliePar2(b));
document.write("La valeur de a est ",a);
// -->
</SCRIPT>
```

Voici ce qu'affiche ce script :

Le double de 4 est 8

La valeur de a est 8

Ces exemples montrent la nécessité de déclarer systématiquement de nouvelles variables avec le mot clé `var`.

Les types de données dans les variables

En Javascript il n'y a pas besoin de déclarer le type de variables que l'on utilise, contrairement à des langages évolués tels que le langage C ou le Java pour lesquels il faut préciser s'il s'agit d'entier (int), de nombre à virgule flottante (float), de caractères (char), ...

En fait le Javascript n'autorise la manipulation que de 4 types de données:

• des nombres: entiers ou à virgules

• des chaînes de caractères (string): une suite de caractères

• des booléens: des variables à deux états permettant de vérifier une condition

`true`: si le résultat est vrai

`false`: lors d'un résultat faux

• des variables de type `null`: un mot caractéristique pour indiquer qu'il n'y a pas de données

Les événements

Les événements sont des actions de l'utilisateur, qui vont pouvoir donner lieu à une interactivité. L'événement par excellence est le clic de souris, car c'est le seul que le HTML gère. Grâce au Javascript il est possible d'associer des fonctions, des méthodes à des événements tels que le passage de la souris au-dessus d'une zone, le changement d'une valeur, ...

Ce sont les gestionnaires d'événements qui permettent d'associer une action à un événement. La syntaxe d'un gestionnaire d'événement est la suivante:

```
onEvenement="Action_Javascript_ou_Fonction();"
```

Lorsqu'il est utilisé dans un lien hypertexte, par exemple, la syntaxe sera la suivante :

```
<A href="onEvenement='Action_Javascript_ou_Fonction();'>Lien</A>
```

Les gestionnaires d'événements sont associés à des objets, et leur code s'insèrent dans la balise de ceux-ci...

Liste des événements

Evénement	Description
Abort (onAbort)	Cet événement a lieu lorsque l'utilisateur interrompt le chargement de l'image
Blur (onBlur)	Se produit lorsque l'élément perd le focus, c'est-à-dire que l'utilisateur clique hors de cet élément, celui-ci n'est alors plus sélectionné comme étant l'élément actif.
Change (onChange)	Se produit lorsque l'utilisateur modifie le contenu d'un champ de données.
Click (onClick)	Se produit lorsque l'utilisateur clique sur l'élément associé à l'événement
Dblclick	Se produit lorsque l'utilisateur double-clique sur l'élément associé

(onDblclick)	à l'événement (un lien hypertexte ou un élément de formulaire). Cet événement n'est supporté que par les versions de Javascript 1.2 et supérieures
Dragdrop (onDragdrop)	Se produit lorsque l'utilisateur effectue un glisser-déposer sur la fenêtre du navigateur. Cet événement n'est supporté que par les versions de Javascript 1.2 et supérieures
Error (onError)	Se déclenche lorsqu'une erreur apparaît durant le chargement de la page
Focus (onFocus)	Se produit lorsque l'utilisateur donne le focus à un élément, c'est-à-dire que cet élément est sélectionné comme étant l'élément actif
Keydown (onKeydown)	Se produit lorsque l'utilisateur appuie sur une touche de son clavier. Cet événement n'est supporté que par les versions de Javascript 1.2 et supérieures
Keypress (onKeypress)	Se produit lorsque l'utilisateur maintient une touche de son clavier enfoncée. Cet événement n'est supporté que par les versions de Javascript 1.2 et supérieures
Keyup (onKeypress)	Se produit lorsque l'utilisateur relâche une touche de son clavier préalablement enfoncée. Cet événement n'est supporté que par les versions de Javascript 1.2 et supérieures
Load (onLoad)	Se produit lorsque le navigateur de l'utilisateur charge la page en cours
MouseOver (onMouseOver)	Se produit lorsque l'utilisateur positionne le curseur de la souris au-dessus d'un élément
MouseOut (onMouseOut)	Se produit lorsque le curseur de la souris quitte un élément.
Reset (onReset)	Se produit lorsque l'utilisateur efface les données d'un formulaire à l'aide du bouton Reset.
Resize (onResize)	Se produit lorsque l'utilisateur redimensionne la fenêtre du navigateur
Submit (onSubmit)	Se produit lorsque l'utilisateur clique sur le bouton de soumission d'un formulaire (le bouton qui permet d'envoyer le formulaire)
Unload (onUnload)	Se produit lorsque le navigateur de l'utilisateur quitte la page en cours

Association des événements aux objets

Chaque événement ne peut pas être associé à n'importe quel objet. Il est évident par exemple qu'un événement OnChange ne pourra pas s'appliquer à un lien hypertexte. Voici un tableau récapitulant les objets auxquels peuvent être associés chaque événement :

Evénements	Objets concernés
abort	Image
blur	Button, Checkbox, FileUpload, Layer, Password, Radio, Reset, Select, Submit, Text, TextArea, window
change	FileUpload, Select, Submit, Text, TextArea
click	Button, document, Checkbox, Link, Radio, Reset, Select, Submit
dblclick	document, Link
dragdrop	window
error	Image, window
focus	Button, Checkbox, FileUpload, Layer, Password, Radio, Reset, Select, Submit, Text, TextArea, window
keydown	document, Image, Link, TextArea
keypress	document, Image, Link, TextArea
keyup	document, Image, Link, TextArea
load	Image, Layer, window
mousedown	Button, document, Link
mousemove	Aucun spécifiquement
mouseout	Layer, Link
mouseover	Area, Layer, Link
mouseup	Button, document, Link
move	window
reset	form
resize	window
select	text, Textarea
submit	Form
unload	window

Quelques exemples d'événements

Le mieux pour apprendre à se servir des événements est de s'entraîner à écrire de petits codes...

Pour vous inspirer, pensez à regarder les fichiers sources de certaines pages web, mais pensez toujours à respecter les auteurs des codes en ne faisant pas un copier-coller de leurs scripts sans leur accord (il est généralement de bon ton de citer la source du javascript que l'on récupère...).

Ouverture d'une boite de dialogue lors d'un click

Le code correspondant à une boîte de dialogue est très simple:

```
window.alert("Votre Texte");
```

Il s'agit donc de le mettre dans la balise d'un lien hypertexte:

```
<HTML>
<HEAD>
<TITLE>Ouverture d'une boite de dialogue lors d'un click</TITLE>
</HEAD>
<BODY>
<A href="javascript:;"onClick="window.alert('Message d\'alerte à utiliser avec
modération');"> Cliquez ici! </A>
</BODY>
</HTML>
```

Analyse du script:

Le gestionnaire d'événement onClick a été inséré dans la balise de lien hypertexte `<A href...`

Le seul but du script est de faire apparaître une boîte de dialogue, ainsi on ne désire pas que le lien nous entraîne sur une autre page, il faut alors insérer "javascript:;" dans l'attribut `href` pour signaler au navigateur que l'on désire rester sur la page en cours. Il ne faut pas mettre un attribut vide au risque de révéler le contenu de votre répertoire à vos visiteurs...

Remarquez l'emploi de \' dans la phrase "Message d'alerte a utiliser avec modération"

Le signe antislash (\) précédant le guillemet permet de signaler au navigateur qu'il ne faut pas l'interpréter comme un délimiteur de chaîne mais comme un simple caractère pour éviter qu'il génère une erreur!

Modification d'une image lors du survol d'un lien par le pointeur de la souris

Il peut être agréable de jouer avec le gestionnaire OnMouseOver pour créer un menu interactif qui se modifie au passage de la souris. On peut même ajouter le gestionnaire OnMouseOut pour rétablir l'image originale lorsque le curseur quitte l'image (Rappel: Son utilisation est limitée aux navigateurs supportant javascript 1.1 et supérieur!).

```
<HTML>
<HEAD>
<TITLE>Modification d'une image lors du passage de la souris</TITLE>
</HEAD>
<BODY>
<A href="javascript:;"
onMouseOver="document.img_1.src='image2.gif';"
onMouseOut="document.img_1.src='image1.gif';">
<IMG name="img_1" src="image1.gif"> </A>
</BODY>
</HTML>
```

Analyse du script:

Pour pouvoir associer un événement à une image il faut que celle-ci soit considérée comme un lien, ainsi on place la balise `<img ...>` entre les balises `<a ...>` et `</a>`

L'événement `onMouseOut` est limitée aux navigateurs supportant javascript 1.1 et supérieur.

La notion de fonction

On appelle *fonction* un sous-programme qui permet d'effectuer un ensemble d'instructions par simple appel de la fonction dans le **corps** du programme principal. Cette notion de sous-programme est généralement appelée fonction dans les langages autres que le Javascript (toutefois leur syntaxe est généralement différente...). Les fonctions permettent d'exécuter dans plusieurs parties du programme une série d'instructions, cela permet une simplicité du code et donc une taille de programme minimale. D'autre part, une fonction peut faire appel à elle-même, on parle alors de fonction récursive (il ne faut pas oublier dans ce cas de mettre une condition de sortie au risque sinon de ne pas pouvoir arrêter le programme...).
Javascript contient des fonctions prédéfinies qui peuvent s'appliquer pour un ou plusieurs types d'objets spécifiques, on appelle ces fonctions des **méthodes.**

La déclaration d'une fonction
Avant d'être utilisée, une fonction doit être définie car pour l'appeler dans le corps du programme il faut que le navigateur la connaisse, c'est-à-dire qu'il connaisse son nom, ses arguments et les instructions qu'elle contient. La définition d'une fonction s'appelle "*déclaration"*. La déclaration d'une fonction se fait grâce au mot clé `function` selon la syntaxe suivante:

```
function Nom_De_La_Fonction(argument1, argument2, ...)
{
liste d'instructions
}
```

Remarques:
• le mot clé `function` est suivi du nom que l'on donne à la fonction
• le nom de la fonction suit les mêmes règles que les noms de variables:
• le nom doit commencer par une lettre
• un nom de fonction peut comporter des lettres, des chiffres et les caractères _ et & (les espaces ne sont pas autorisés!)
• le nom de la fonction, comme celui des variables est sensible à la casse (différenciation entre les minuscules et majuscules)
• Les arguments sont facultatifs, mais s'il n'y a pas d'arguments, les parenthèses doivent rester présentes
•Il ne faut pas oublier de refermer les accolades
• Le nombre d'accolades ouvertes (fonction, boucles et autres structures) doit être égal au nombre d'accolades fermées!
• La même chose s'applique pour les parenthèses, les crochets ou les guillemets!
Une fois cette étape franchie, votre fonction ne s'exécutera pas tant que l'on ne fait pas appel à elle quelque part dans la page!

Appel de fonction

Pour exécuter une fonction, il suffit de faire appel à elle en écrivant son nom (une fois de plus en respectant la casse) suivi d'une parenthèse ouverte (éventuellement des arguments) puis d'une parenthèse fermée:

```
Nom_De_La_Fonction();
```

Remarques:

- le point virgule signifie la fin d'une instruction et permet au navigateur de distinguer les différents blocs d'instructions
- si jamais vous avez défini des arguments dans la déclaration de la fonction, il faudra veiller à les inclure lors de l'appel de la fonction (le même nombre d'arguments séparés par des virgules!)

Veillez toujours à ce qu'une fonction soit déclarée avant d'être appelée, sachant que le navigateur traite la page de haut en bas (Pour éviter des erreurs de ce type on déclare généralement les fonctions dans des balises `<SCRIPT>` situées dans l'en-tête de la page, c'est-à-dire entre les balises `<HEAD> et </HEAD>`)

Grâce au gestionnaire d'événement `onLoad` (à placer dans la balise `BODY`) il est possible d'exécuter une fonction au chargement de la page, comme par exemple l'initialisation des variables pour votre script, et/ou le test du navigateur pour savoir si celui-ci est apte à faire fonctionner le script.

Il s'utilise de la manière suivante:

```
<HTML>
<HEAD>
<SCRIPT language="Javascript">
<!--
  function Chargement() {
alert('Bienvenue sur le site');
}
 //-->
</SCRIPT>
</HEAD>
<BODY onLoad="Chargement();" >
  Javascript qui ne sert absolument à rien si ce n'est déranger vos visiteurs...
  </BODY>
</HTML>
```

Les paramètres d'une fonction

Il est possible de passer des paramètres à une fonction, c'est-à-dire lui fournir une valeur ou le nom d'une variable afin que la fonction puisse effectuer des opérations sur ces paramètres ou bien grâce à ces paramètres.

Lorsque vous passez plusieurs paramètres à une fonction il faut les séparer par des virgules, aussi bien dans la déclaration que dans l'appel et il faudra veiller à bien passer le bon nombre de paramètres lors de l'appel au risque sinon de créer une erreur dans votre Javascript...

Imaginons que l'on veuille créer une page Web qui affiche une boîte de dialogue (les boîtes de dialogues sont à éviter dans vos pages car elles sont énervantes, mais c'est toutefois une manière simple d'apprendre le Javascript) qui affiche un texte différent selon le lien sur lequel on appuie.

La méthode de base consiste à faire une fonction pour chaque texte à afficher:

```
<HTML>
<HEAD>
<SCRIPT language="Javascript">
<!--
function Affiche1() {
alert('Texte 1');
}
function Affiche2() {
alert('Texte2');
}
//-->
</SCRIPT>
</HEAD>
<BODY onLoad="Chargement();" >
<A href="javascript:;" onClick="Affiche1();">Texte1</A>
<A href="javascript:;" onClick="Affiche2();">Texte2</A>
</BODY>
</HTML>
```

Il existe toutefois une methode plus "classe" qui consiste à créer une fonction qui a comme paramètre le texte à afficher:

```
<HTML>
<HEAD>
<SCRIPT language="Javascript">
<!--
function Affiche(Texte) {
alert(Texte);
}
//-->
</SCRIPT>
</HEAD>
<BODY onLoad="Chargement();" >
<A href="javascript:;" onClick="Affiche('Texte1');">Texte1</A>
<A href="javascript:;" onClick="Affiche('Texte2');">Texte2</A>
</BODY>
</HTML>
```

Résultat: Aucune différence visuellement mais vous n'avez plus qu'une seule fonction qui peut vous afficher n'importe quel texte...

Travailler sur des variables dans les fonctions

Lorsque vous manipulerez des variables dans des fonctions, il vous arrivera de constater que la variable retrouve sa valeur d'origine dès que l'on sort de la fonction, malgré toutes les affectations faites au sein de celle-ci ...

Cela est dû à la portée des variables, c'est-à-dire selon qu'elles ont été définies comme **variables globales ou locales.**

• Une variable déclarée implicitement dans la fonction (non précédée du mot-clé `var`) sera globale , c'est-à-dire accessible après exécution de la fonction

• Une variable déclarée explicitement (précédée du mot-clé `var`) sera locale , c'est-à-dire accessible uniquement dans la fonction, toute référence à cette variable hors de la fonction provoquera une erreur (variable inconnue)...

***Le mot-clé* this**

Lorsque vous faîtes appel à une fonction à partir d'un objet, par exemple un formulaire, le mot clé *this* fait référence à l'objet en cours et vous évite d'avoir à définir l'objet en tapant `window.objet1.objet2...` ainsi lorsque l'on passe l'objet en cours en paramètre d'une fonction, il suffit de taper `nom_de_la_fonction(this)` pour pouvoir manipuler cet objet à partir de la fonction.
Pour manipuler les propriétés de l'objet il suffira de taper `this.propriete` (où `propriete` représente bien sûr le nom de la propriété).

La notion de méthode

Une méthode est une fonction associée à un objet, c'est-à-dire une action que l'on peut faire exécuter à un objet. Les méthodes des objets du navigateur sont des fonctions définies à l'avance par les normes HTML, on ne peut donc pas les modifier, il est toutefois possible de créer une méthode personnelle pour un objet que l'on a créé soi-même. Prenons par exemple une page HTML, elle est composée d'un objet appelé ***document***. L'objet ***document*** a par exemple la méthode `write()` qui lui est associée et qui permet de modifier le contenu de la page HTML en affichant du texte. Une méthode s'appelle un peu comme une propriété, c'est-à-dire de la manière suivante:

```
window.objet1.objet2.methode()
```

Dans le cas de la méthode `write()`, l'appel se fait comme suit:

```
window.document.write()
```

***La méthode* write**

La méthode `write()` de l'objet `document` permet de modifier de façon dynamique le contenu d'une page HTML. Voici la syntaxe de la méthode `write()` :

```
window.document.write(expression1, expression2, ...)
```

Cette méthode permet d'écrire le résultat des expressions passées en paramètre dans le document dans lequel elle est utilisée. Il est ainsi possible d'utiliser la méthode `write()` de différentes façons:

• soit en passant directement le texte en paramètres:

```
document.write("bonjour");
```

qui aura pour effet de concaténer la chaîne 'bonjour' à l'endroit où est placé le script

• soit en passant le texte par l'intermédiaire d'une variable:

```
Chaine='bonjour';
document.write(Chaine);
```

Ce qui aura pour effet de concaténer la chaîne 'bonjour' (contenue dans la variable *Chaine*)à l'endroit où est placé le script

• soit en utilisant les deux:

```
Chaine='bonjour';
document.write('je vous passe le ' + Chaine);
```

Ce qui aura pour effet de concaténer la chaîne 'bonjour' (contenue dans la variable *Chaine*) à la suite de la chaîne de caractère 'je vous passe le' dans la page HTML.

• soit en insérant directement une expression, qui sera évaluée dans un premier temps et dont le résultat sera ensuite affiché:

```
Chaine='La racine carrée de 2 vaut : ';
document.write(Chaine+sqrt(2));
```

Il est notamment possible d'utiliser des balises HTML à l'intérieur même de la méthode *write*:

```
document.write('<font color="#FF0000">Bonjour</font>');
```

Les objets du navigateur

Lorsque vous ouvrez une page Web, le navigateur crée des objets prédéfinis correspondant à la page Web, à l'état du navigateur, et peuvent donner beaucoup d'informations qui vous seront utiles.
Les objets de base du navigateur sont les suivants:

- **navigator** : qui contient des informations sur le navigateur de celui qui visite la page.
- **window** : c'est l'objet où s'affiche la page, il contient donc des propriétés concernant la fenêtre elle-même mais aussi tous les objets-enfants contenus dans celle-ci.
- **location** : contient des informations relatives à l'adresse de la page à l'écran.
- **history**: c'est l'historique, c'est-à-dire la liste de liens qui ont été visités précédemment
- **document** : il contient les propriétés sur le contenu du document (couleur d'arrière plan, titre,...)

Ces objets sont largement dépendant du contenu de la page. En effet, mis à part des objets tels que **navigator** qui sont figés pour un utilisateur donné, le contenu des autres objets variera suivant le contenu de la page, car suivant la page les objets présents dans celles-ci (sous-objets des objets décrits précédemment) sont différents. Voyons comment ceux-ci sont organisés.

Les objets du navigateur sont classés hiérarchiquement

Les objets du navigateur sont classés dans une hiérarchie qui décrit la page affichée à l'écran, et qui permet d'accéder à n'importe quel objet grâce à une désignation dépendant de la hiérarchie (on part du sommet puis on descend l'arborescence).
Dans cette hiérarchie, les descendants d'un objet sont des propriétés de ces objets mais peuvent aussi être des objets qui contiennent eux même des propriétés...
Voyons voir à quoi ressemble cette hiérarchie:

Niveau1	Niveau2	Niveau3	Commentaire
navigator			Informations sur le browser utilisé
window			Gestion de la fenêtre d'affichage
	parent, frames, self, top		Désignation de la sous-fenêtre
	location		Informations sur l'emplacement de la page
	history		Accès à l'historique (sites précédemment visités)
	document		Informations sur le contenu de la fenêtre (éléments qui composent la page)
		images	Référence des images présentes dans la page
		forms	Référence des formulaires présents dans la page
		links	Référence des liens présents dans la page
		anchors	Référence des ancrages présents dans la page

Comment accéder à un objet?

Pour accéder à un objet du navigateur, il faut parcourir la hiérarchie du navigateur, en partant du sommet (l'objet `window`), puis en parcourant tous les maillons jusqu'à atteindre l'objet désiré. La syntaxe est `window.objet1.objet2.objet3.objet_vise` (ici il y a trois objets intermédiaire `objet1 objet2 objet3` mais ce nombre peut varier de 0 à un très grand nombre d'objets, suivant l'imbrication de vos objets dans la page...).

Pour lire ou modifier le contenu d'une propriété de l'objet visé il suffit de rajouter un point, puis le nom de la propriété. Certaines propriétés sont modifiables, c'est-à-dire que dynamiquement il est possible de modifier un élément (du texte, une image, ...). Certaines propriétés sont par contre en lecture seule, c'est-à-dire qu'elles permettent uniquement de récupérer des informations mais qu'il est impossible de les modifier...

Les opérateurs

Les opérateurs sont des symboles qui permettent de manipuler des variables, c'est-à-dire effectuer des opérations, les évaluer, ...

On distingue plusieurs types d'opérateurs:

• les opérateurs de calcul
• les opérateurs d'assignation
• les opérateurs d'incrémentation
• les opérateurs de comparaison
• les opérateurs logiques

Les opérateurs de calcul

Les opérateurs de calcul permettent de modifier mathématiquement la valeur d'une variable

+	opérateur d'addition	Ajoute deux valeurs
-	opérateur de soustraction	Soustrait deux valeurs
*	opérateur de multiplication	Multiplie deux valeurs
/	plus: opérateur de division	Divise deux valeurs
=	opérateur d'affectation	Affecte une valeur à une variable

Les opérateurs d'incrémentation

Ce type d'opérateur permet de facilement augmenter ou diminuer d'une unité une variable. Ces opérateurs sont très utiles pour des structures telles que des boucles, qui ont besoin d'un compteur (variable qui augmente de un en un).

Un opérateur de type *x++* permet de remplacer des notations lourdes telles que *x=x+1* ou bien *x+=1*

++ Incrémentation	Augmente d'une unité la variable
-- Décrémentation	Diminue d'une unité la variable

Les opérateurs de comparaison

==	opérateur d'égalité	Compare deux valeurs et vérifie leur égalité
<	opérateur d'infériorité stricte	Vérifie qu'une variable est strictement inférieure à une valeur
<=	opérateur d'infériorité	Vérifie qu'une variable est inférieure ou égale à une valeur
>	opérateur de supériorité stricte	Vérifie qu'une variable est strictement supérieure à une valeur
>=	opérateur de supériorité	Vérifie qu'une variable est supérieure ou égale à une valeur
!=	opérateur de différence	Vérifie qu'une variable est différente d'une valeur

Les opérateurs logiques (booléens)

Ce type d'opérateur permet de vérifier si plusieurs conditions sont vraies:

||	OU logique	Vérifie qu'une des conditions est réalisée	((condition1)||(condition2))
&&	ET logique	Vérifie que toutes les conditions sont réalisées	((condition1)&&(condition2))
!	NON logique	Inverse l'état d'une variable booléenne (retourne la valeur 1 si la variable vaut 0, 0 si elle vaut 1)	(!condition)

Les priorités

Lorsque l'on associe plusieurs opérateurs, il faut que le navigateur sache dans quel ordre les traiter, voici donc dans **l'ordre décroissant** les priorités des principaux opérateurs:

()	[]			
--	++	!	~	-
*	/	%		
+	-			
<	<=	>=	>	
==	!=			
|				
&&	||			
?	:			
=				

Les structures conditionnelles

On appelle structure conditionnelle les instructions qui permettent de tester si une condition est vraie ou non, ce qui permet de donner de l'interactivité à vos scripts par exemple.

L'instruction if

L'instruction `if` est la structure de test la plus basique, on la retrouve dans tous les langages (avec une syntaxe différente...). Elle permet d'exécuter une série d'instructions si une condition est réalisée.

La syntaxe de cette expression est la suivante:

```
if (condition réalisé) {
liste d'instructions
}
```

Remarques:

• la condition doit être entre des parenthèses

• il est possible de définir plusieurs conditions à remplir avec les opérateurs ET et OU (&& et ||) par exemple:

`if ((condition1)&&(condition2))` teste si les deux conditions sont vraies

`if ((condition1)||(condition2))` exécutera les instructions si l'une ou l'autre des deux conditions est vraie

• s'il n'y a qu'une instruction, les accolades ne sont pas nécessaires...

L'instruction if ... else

L'instruction `if` dans sa forme basique ne permet de tester qu'une condition, or la plupart du temps on aimerait pouvoir choisir les instructions à exécuter en cas de non réalisation de la condition...

L'expression `if ... else` permet d'exécuter une autre série d'instruction en cas de non-réalisation de la condition.

La syntaxe de cette expression est la suivante:

```
if (condition réalisé) {
liste d'instructions
}
else {
autre série d'instructions
}
```

Une façon plus courte de faire un test

Il est possible de faire un test avec une structure beaucoup moins lourde grâce à la structure suivante:

```
(condition) ? instruction si vrai : instruction si faux
```

Remarques:

• La condition doit être entre des parenthèses

• Lorsque la condition est vraie, l'instruction de gauche est exécutée

• Lorsque la condition est fausse, l'instruction de droite est exécutée

Les boucles

Les boucles sont des structures qui permettent d'exécuter plusieurs fois la même série d'instructions jusqu'à ce qu'une condition ne soit plus réalisée...
La façon la plus commune de faire une boucle est de créer un compteur (une variable qui s'incrémente, c'est-à-dire qui augmente de 1 à chaque tour de boucle) et de faire arrêter la boucle lorsque le compteur dépasse une certaine valeur.

La boucle for

L'instruction `for` permet d'exécuter plusieurs fois la même série d'instructions: c'est une boucle!
Dans sa syntaxe, il suffit de préciser le nom de la variable qui sert de compteur (et éventuellement sa valeur de départ, la condition sur la variable pour laquelle la boucle s'arrête (basiquement une condition qui teste si la valeur du compteur dépasse une limite) et enfin une instruction qui incrémente (ou décrémente) le compteur.
La syntaxe de cette expression est la suivante:

```
for (compteur; condition; modification du compteur) {
liste d'instructions
}
```

Par exemple:

```
for (i=1; i<6; i++) {
Alert(i)
}
```

Cette boucle affiche 5 fois la valeur de i, c'est-à-dire 1,2,3,4,5
Elle commence à i=1, vérifie que i est bien inférieur à 6, etc... jusqu'à atteindre la valeur i=6, pour laquelle la condition ne sera plus réalisée, la boucle s'interrompra et le programme continuera son cours.

- il faudra toujours vérifier que la boucle a bien une condition de sortie (i.e le compteur s'incrémente correctement)
- une instruction Alert(i); dans votre boucle est un bon moyen pour vérifier la valeur du compteur pas à pas!
- il faut bien compter le nombre de fois que l'on veut faire exécuter la boucle:
    - `for(i=0;i<10;i++)` exécute 10 fois la boucle (i de 0 à 9)
    - `for(i=0;i<=10;i++)` exécute 11 fois la boucle (i de 0 à 10)
    - `for(i=1;i<10;i++)` exécute 9 fois la boucle (i de 1 à 9)
    - `for(i=1;i<=10;i++)` exécute 10 fois la boucle (i de 1 à 10)

L'instruction while

L'instruction `while` représente un autre moyen d'exécuter plusieurs fois la même série d'instructions.
La syntaxe de cette expression est la suivante:

```
while (condition réalisée) {
liste d'instructions
}
```

Cette instruction exécute la liste d'instructions tant que (while est un mot anglais qui signifie tant que) la condition est réalisée.

La condition de sortie pouvant être n'importe quelle structure conditionnelle, les risques de boucle infinie (boucle dont la condition est toujours vraie) sont grands, c'est-à-dire qu'elle risque de provoquer un plantage du navigateur!

Saut inconditionnel

Il peut être nécessaire de faire sauter à la boucle une ou plusieurs valeurs sans pour autant mettre fin à celle-ci.

La syntaxe de cette expression est "continue;" (cette instruction se place dans une boucle!), on l'associe généralement à une structure conditionnelle, sinon les lignes situées entre cette instruction et la fin de la boucle seraient obsolètes.

Exemple: Imaginons que l'on veuille imprimer pour x allant de 1 à 10 la valeur de 1/(x-7) ... il est évident que pour x=7 il y aura une erreur. Heureusement, grâce à l'instruction continue il est possible de traiter cette valeur à part puis de continuer la boucle!

```
x=1
while (x<=10) {
if (x == 7) {
Alert('division par 0');
continue;
}
a = 1/(x-7);
Alert(x);
x++
}
```

Il y avait une erreur dans ce script... peut-être ne l'avez-vous pas vue:

Lorsque x est égal à 7, le compteur ne s'incrémente plus, il reste constamment à la valeur 7, il aurait fallu écrire:

```
x=1
while (x<=10) {
if (x == 7) {
Alert('division par 0');
x++;
continue;
}
a = 1/(x-7);
Alert(x);
x++
}
```

Arrêt inconditionnel

A l'inverse, il peut être voulu d'arrêter prématurément la boucle, pour une autre conditon que celle précisé dans l'en-tète de la boucle. L'instruction `break` permet d'arrêter une boucle (`for` ou bien `while`). Il s'agit, tout comme continue, de l'associer à une structure conditionnelle, sans laquelle la boucle ne ferait jamais plus d'un tour!

Dans l'exemple de tout à l'heure, par exemple si l'on ne savait pas à quel moment le dénominateur (x-7) s'annule (bon...OK...pour des équations plus compliquées par exemple) il serait possible de faire arrêter la boucle en cas d'annulation du dénominateur, pour éviter une division par zéro!

```
for (x=1; x<=10; x++) {
a = x-7;
if (a == 0) {
Alert('division par 0');
break;
}
Alert(1/a);
}
```

Exercices

```
<HTML>
<HEAD>
<TITLE>Modification d'une image lors du passage de la souris</TITLE>
</HEAD>
<BODY>
<A HREF="javascript:;"
onMouseOver="document.img_1.src='image2.gif';"
onMouseOut="document.img_1.src='image1.gif';">
<IMG name="img_1" src="image1.gif">_</A>
</BODY>
</HTML>

<HTML><HEAD>
<TITLE>RollOver avec feuille de style</TITLE>
<SCRIPT LANGUAGE="JavaScript" type="text/javascript">

/* Afficher l'objet */
function affichobjet(objet) {
objet.visibility = VISIBLE;
}
/* Cacher l'objet */
function cachobjet(objet) {
objet.visibility = HIDDEN;
}
</SCRIPT>
<STYLE TYPE="text/css">
.texte {
font-family:         Futura Md BT;
font-size:           80px;
position:            absolute;
top:                 50px;
left:                5px;
visibility:          visible;
z-index:             2;
}
.texte2 {
font-family:         Futura Md BT;
font-size:           80px;
position:            absolute;
```

```
top:                150px;
left:               5px;
visibility:         visible;
z-index:            2;
}
.image {
position:           absolute;
top:                50px;
left:         50%;
visibility:         HIDDEN;
z-index:            1;
}
.image2 {
position:           absolute;
top:                50px;
left:         50%;
visibility:         HIDDEN;
z-index:            1;
}
BODY {
background-color:   "yellow";
}
A {
color:              "black";
text-decoration:    none
}
</STYLE>
</HEAD>
<BODY>
<DIV id ="sno" class="image"> <IMG SRC="snoopy.jpg" WIDTH="272" HEIGHT="350"></DIV>
<DIV class="texte"><A HREF="javascript:" onMouseOver = "affichobjet(snoopy)"
onMouseOut = "cachobjet(snoopy)" >Bonjour</A></DIV>
<DIV id ="sno2" class="image2"> <IMG SRC="snoopy2.jpg" WIDTH="272"
HEIGHT="350"></DIV>
<DIV class="texte2"><A HREF="javascipt:" onMouseOver = "affichobjet(snoopy2)"
onMouseOut = "cachobjet(snoopy2)" >Au revoir</A></DIV>

<SCRIPT LANGUAGE="JavaScript" type="text/javascript">
/* Détection du browser et attibution des variables */
var navi = (navigator.appName == "Netscape" && parseInt(navigator.appVersion) >= 4);
var HIDDEN = (navi) ? 'hide' : 'hidden';
var VISIBLE = (navi) ? 'show' : 'visible';
/* création des variables pour le positionnement absolu des titres */
var snoopy = (navi) ? document.sno : document.all.sno.style;
var snoopy2 = (navi) ? document.sno2 : document.all.sno2.style;
</SCRIPT>
</BODY>
</HTML>


<HTML><HEAD>
<TITLE>Popup</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var nouv_fenetre;
function popUp(page, name, details) {
nouv_fenetre=window.open(page, name, details);
nouv_fenetre.focus();
```

```
return false;}
</SCRIPT>
</HEAD><BR>
<BODY >
```
Introduction au `<a href="recitint.html" onClick="return popUp('recitint.html','nouv_fenetre','toolbar=no,location=no,directories=no,status=no,menubar=no,scrollbars=yes,resizable=yes,width=300,height=200')">` récit interactif`</a>`.
```
</BODY>
</HTML>

<HTML><HEAD>
<TITLE>L'objet formulaire</TITLE><SCRIPT LANGUAGE="JavaScript1.2"
type="text/javascript">
<!--
function commande()
{var message="Bonjour "
message=message+document.forms[0].prenom.value+" "+document.forms[0].nom.value+ ".
Vous avez commandé une "
if (document.forms[0].taille.options[0].selected){
message=message+"petite"
}
if (document.forms[0].taille.options[1].selected){
message=message+"moyenne"
}
if (document.forms[0].taille.options[2].selected){
message=message+"grande"
}
message=message+" pizza avec du fromage, des tomates"
if (document.forms[0].ingredient[0].checked){
message=message+" et du jambon"
}
if (document.forms[0].ingredient[1].checked){
message=message+" et du chorizo"
}
if (document.forms[0].ingredient[2].checked){
message=message+" et du lard"
}
message=message+". Elle vous sera livrée à l'adresse suivante:
"+document.forms[0].adresse.value
alert(message)
}
 -->
</SCRIPT>
</HEAD>
<BODY >
<P><B>Commandez votre pizza:</B></P>
<FORM Action="javascript:" METHOD="POST"  onSubmit="commande()" >
<TABLE BORDER="0" CELLSPACING="2" CELLPADDING="1">
<TR><TD>Nom:</TD> <TD><INPUT NAME="nom" TYPE="text" SIZE="25"
MAXLENGTH="25"></TD></TR>
<TR><TD>Prénom:</TD> <TD><INPUT NAME="prenom" TYPE="text" SIZE="25"
MAXLENGTH="25"></TD></TR>
<TR><TD>Adresse:</TD> <TD><TEXTAREA  NAME="adresse" ROWS=3
COLS=25></TEXTAREA></TD></TR>
```

```
<TR><TD>Taille:</TD> <TD><SELECT NAME="taille" SIZE="1">
<OPTION>Petite</OPTION>
<OPTION SELECTED>Moyenne</OPTION>
<OPTION>Grande
</OPTION>
</SELECT></TD></TR>
<TR><TD colspan=2><BR>
Choisissez vos ingrédients:<P></P>
<INPUT NAME="ingredient" TYPE="checkbox" VALUE="jambon" > Jambon<BR>
<INPUT NAME="ingredient" TYPE="checkbox" VALUE="chorizo" > Chorizo<BR>
<INPUT NAME="ingredient" TYPE="checkbox" VALUE="lardons" > Lardons</TD> </TR>
</P>
</TABLE>
<P>
<INPUT TYPE="submit" VALUE="Commandez"> <INPUT TYPE="reset" VALUE="Annuler"></P>
</FORM>
</BODY>
</HTML>
```

Commandez votre pizza:

Nom:

Prénom:

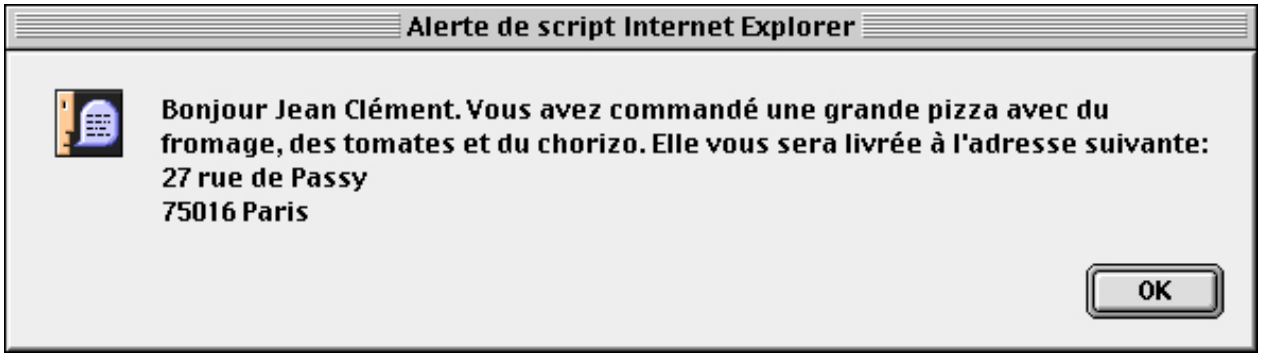
Adresse:

Taille: Moyenne

Choisissez vos ingrédients:

- ☐ Jambon
- ☐ Chorizo
- ☐ Lardons

Commandez Annuler


Alerte de script Internet Explorer
Bonjour Jean Clément. Vous avez commandé une grande pizza avec du fromage, des tomates et du chorizo. Elle vous sera livrée à l'adresse suivante:
27 rue de Passy
75016 Paris
OK

INTRODUCTION À PHP ET MySQL

PHP et MySQL

PHP est un langage de programmation qui s'écrit dans les page HTML, comme Javascript, mais qui est interprété par le serveur et non par le navigateur.
Un des très grand avantage de PHP est sans aucun doute l'accès facile à différentes bases de données telles que Oracle, Sybase, PostgreSQL ou encore MySQL. Nous traiterons ici plus particulièrement MySQL car c'est ce gestionnaire de bases de données que l'on retrouve chez la plupart des hébergeurs. Il suffit généralement à la plupart des sites Web et il est gratuite, comme PHP. MySQL fonctionne sous Linux et Windows. Il est très souvent utilisé avec PHP afin de créer un site entièrement dynamique et une mise à jour simplifiée.

Installer et lancer Easy PHP 1.5

EasyPHP est un pack logiciel gratuit qui permet d'installer sur une machine locale (non reliée à Internet) un serveur Apache, un interpréteur PHP et le gestionnaire de base de données MySQL. On peut ainsi simuler une situation de réseau.
On ne peut pas à proprement parler du lancement d'EasyPHP, il s'agit en fait de la mise en route du serveur Apache et de MySQL. A l'installation, un raccourci vers EasyPHP est créé dans le répertoire "Démarrer/Programmes/EasyPHP". Une fois EasyPHP lancé, une icone se place dans la barre des tâches à coté de l'horloge. Un clic droit permet d'accéder à différents menus :

- Fichier Log : renvoie aux erreurs générées par Apache et MySQL
- Configuration : donne accès aux différentes configurations d'EasyPHP
- Web local : ouvre la page "http://localhost/"
- Démarrer/Arrêter : démarre/arrête Apache et MySQL
- Quitter : ferme EasyPHP

Pour que vos pages PHP soient interprétées, il est impératif de placer vos fichiers dans le répertoire www. Le serveur Apache est configuré pour ouvrir automatiquement un fichier index lorsque vous saisissez l'adresse 'http://localhost/' (à condition évidemment que le serveur Apache soit en route). Cette page sert de page d'accueil au web local et permet de vérifier le bon fonctionnement d'EasyPHP. Il est conseillé de créer un répertoire par projet dans le répertoire www afin d'avoir une vision plus claire des développements.

Le langage MySQL

Une base de données peut contenir un nombre variable de tables (pour commencer une seule suffira). Chaque table est organisée en lignes et en colonnes. L'intersection entre une ligne et une colonne correspond à l'emplacement de l'élément de donnée que vous désirez stocker ou auquel vous désirez accéder. Chaque colonne accepte seulement un type de donéée prédéfini, `INT` pour

entier, par exemple, ou VARCHAR pour un nombre variable de caractères avec une limite supérieure définie.
Pour créer une nouvelle table dans une base de données que nous avons sélectionnée, nous utilisons la requête suivante :

```
CREATE TABLE matable (nom1 VARCHAR(20), nom2 VARCHAR(20), age INT) ;
```

Notre nouvelle table possède trois colonnes. Nom1 et nom2 peuvent contenir des chaînes composées de 30 caractères au maximum. Age peut recevoir un entier.
Pour ajouter des données à cette table, nous pouvons faire appel à une instruction INSERT :

```
INSERT INTO matable (nom1, nom2, age) VALUES ('Eugène', 'Dupond', 78) ;
```

Les noms de champs auxquels nous désirons ajouter des données sont définis dans la première paire de parenthèses. Les valeurs à insérer sont définies dans la seconde paire.
Pour recueillir toutes les données d'une table, nous pourrions faire appel à l'instruction SELECT :

```
SELECT * FROM matable ;
```

Le symbole « * » représente un caractère spécial signifiant « tous les champs ». Pour recueillir l'information d'un seul champ, nous pouvons remplacer le caractère spécial par le nom de la colonne :

```
SELECT age FROM matable ;
```

Pour modifier les valeurs déjà stockées dans une table, vous pouvez faire appel à une instruction UPDATE :

```
UPDATE matable SET nom1= « Jeanne » ;
```

Cela modifie le champ nom1 de chaque ligne en « Jeanne ». Nous pouvons réduire le champ d'action des instructions SELECT et UPDATE avec une clause WHERE.
Par exemple :

```
SELECT * FROM matable WHERE nom1= 'Jeanne' ;
```

Ne renvoie que les lignes dont les champs nom1 contiennent la chaîne « Jeanne ». L'exemple suivant

```
UPDATE matable SET nom1= « Jeanne » WHERE nom2= « Lagrange » ;
```

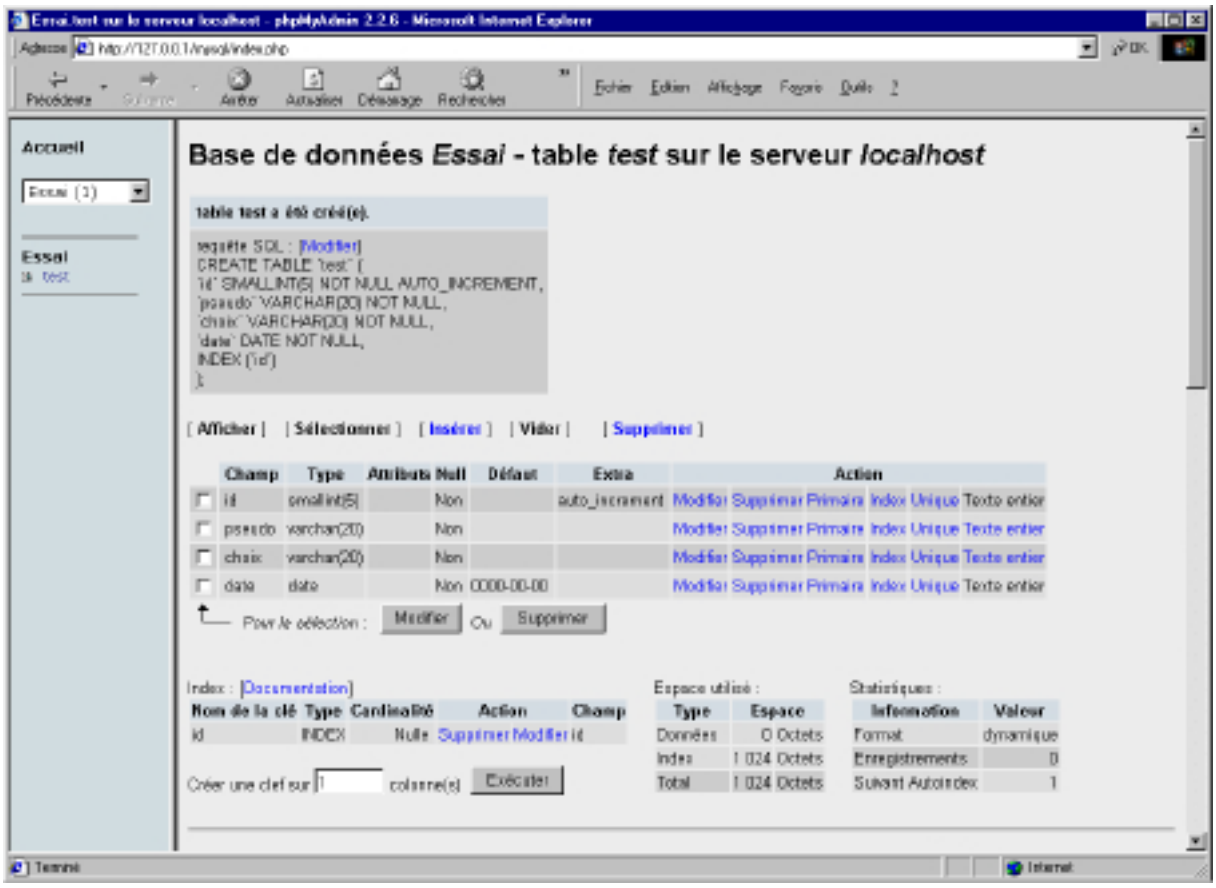
Modifie les champs nom1 de toutes les lignes dont le champ nom1 contient « Lagrange ».

Créer la base de données

Nous allons partir d'un exemple concret : recueillir les données d'un sondage à travers un formulaire, les stocker dans une base de données et les afficher sous forme de tableau. Pour créer la base de données, il faut lui donner un nom (« documentation », par exemple), puis créer une table qui contiendra les données (« sondage », par exemple). Cette table comportera quatre champs : « id » (le numéro de l'enregistrement), « pseudo » (le nom du sondé), « choix » (sa réponse), « date » (la date de sa réponse).

CREATE TABLE 'sondage (
'id' SMALLINT(5) NOT NULL AUTO_INCREMENT,
'pseudo' VARCHAR(20) NOT NULL,
'choix' VARCHAR(20) NOT NULL,

‘date’ DATE NOT NULL,
INDEX (‘id’)
) ;



La connexion

C’est le langage PHP qui assure la connection entre le navigateur et la base de donnée, grâce à des fonctions prédéfinies. Nous utiliserons les suivantes :

`mysql_create_db --` Crée une base de données MySQL.
`mysql_select_db --` Sélectionne une base de données MySQL.
`mysql_connect --` Ouvre une connexion à un serveur MySQL.
`mysql_query --` Envoie une requête SQL à un serveur MySQL.
`mysql_fetch_array --` Retourne une ligne de résultat sous la forme d'un tableau associatif.
`mysql_close --` Ferme la connexion MySQL
`mysql_pconnect --` Ouvre une connexion persistante à un serveur MySQL.

Avant de faire le moindre accès à une base de données, il faut impérativement établir une connexion et choisir sa base de données (en effet un serveur de bases de données peut donner accès à plusieurs bases même si les hébergeurs gratuits ou peu chers en proposent généralement une seule).

Cela se fait de la manière suivante:

```
<?php
$idConnexion = mysql_connect("Nom_Hote","Nom_Utilisateur","Password_utilisateur");
$connexionReussie = mysql_select_db("matable");
mysql_close();
?>
```

Remarques:

1. Nom_Hote est le nom de l'hôte où se trouve MySQL ("sql" ou "localhost" ou autre c'est une information qu'il faut rechercher chez votre hébergeur. La plupart du temps elle vous est transmise au moment de l'inscription. En local, c'est "localhost")
2. Nom_Utilisateur est le login de celui qui peut se connecter à la BD (c'est souvent celui du compte ftp) en local par défaut c'est "root".
3. Password_utilisateur est le mot de passe correspondant au login ci-dessus, en local c'est une chaîne vide par défaut, sinon c'est généralement votre mot de passe FTP.
4. matable est un nom de BD qui doit exister, en local, vous devrez en créer une en passant par exemple par PHPMyAdmin (il suffit de lui donner un nom), chez un hébergeur c'est souvent votre login.
5. mysql_connect() retourne un identifiant de connexion ou FALSE en cas d'échec.

Un exemple concret en local:

```
<?php
  $idConnexion = mysql_connect("localhost", "root", "");
  if ($idConnexion) echo "Connexion au serveur réussie";
  else echo "Connexion au serveur ratée";
  $connexionReussie = mysql_select_db("mabase");
  if ($connexionReussie) echo "Connexion à la base réussie";
  else echo "Connexion à la base ratée";
  mysql_close();
?>
```

Remarques:

1. mysql_select_db() renvoie un booléen utile pour savoir si la connexion a été réussie

On peut aussi utiliser une autre méthode pour stopper le script si la connexion ne peut pas se faire.

```
<?php
  mysql_connect("localhost", "root", "")
      or die("Impossible de se connecter au serveur de bases de données.");
  mysql_select_db("mabase")
      or die("Cette base de données n'existe pas");
  mysql_close();
?>
```

Remarques:

1. `or die("")` permet d'afficher un texte et de stopper le script si la fonction précédente renvoie une erreur.

Afin, d'éviter d'ouvrir une nouvelle connexion à chaque script, il est possible d'utiliser la fonction `mysql_pconnect()`.

Remarques:

1. dans ce cas, il n'y a plus lieu de faire appel à mysql_close()

Création d'un formulaire de sondage

L'idée c'est de créer un formulaire de sondage et de pouvoir consulter les résultats de celui-ci. L'interface sera de l'HTML "classique", voici le code de cette page que l'on nommera sondage.html

```
<html>
<head>
<title>Sondage PHP Documentation</title>
</head>
<body>
<h2>Comment avez-vous découvert le département documentation ?</h2>
<form method="post" action="sondage.php">
      <input type="radio" name="choix" value="presse">Par la presse spécialisée<br>
      <input type="radio" name="choix" value="annuaire">Par un annuaire<br>
      <input type="radio" name="choix" value="moteur">Par un moteur de recherche<br>
      <input type="radio" name="choix" value="forum">Par un forum<br>
      <input type="radio" name="choix" value="news">Par les news<br>
      <input type="radio" name="choix" value="ami">Par un ami<br>
      <input type="radio" name="choix" value="autre">Autre<br>
Merci de laisser votre pseudo:
      <input type="text" name="pseudo"><br>
      <input type="submit">
</form>
</body>
</html>
```

Remarques:

1. On choisit un formulaire avec une méthode POST, les éléments de ce formulaire sont de type radio, il n'y a qu'un choix possible.
2. On demande également le pseudo de l'utilisateur.

Création du programme sondage.php

Voilà maintenant le fichier qui nous intéresse le plus, c'est à dire celui qui va stocker dans la base de données ce résultat. Nous l'appelerons *sondage.php*

```
<?php
$serveur     = "localhost";
$utilisateur = "root";
$motDePasse  = "";
$base        = "documentation";

mysql_pconnect($serveur, $utilisateur , $motDePasse)
    or die("Impossible de se connecter au serveur de bases de données.");
mysql_select_db($base)
    or die("Base de données non trouvée.");
$pseudo = $_POST["pseudo"];
$choix  = $_POST["choix"];
mysql_query("INSERT INTO sondage (pseudo, choix)".

                    " VALUES ('$pseudo', '$choix') ")
    or die("Impossible d'insérer le résultat du sondage");
echo "Merci ".$pseudo;
?>
```

Remarques:

1. Après s'être correctement connecté, on insert dans la table sondage, $pseudo et $choix aux champs respectifs « pseudo » et « choix ». Comme nous ne renseignons pas le champ date de type TIMESTAMP, avec MySQL, celui-ci prend automatiquement la valeur de la date courante (également disponible par un appel à la fonction NOW()). Si l'insertion échoue un message d'erreur apparaît.

Visualiser les résultats du sondage

Maintenant que la base de données est alimentée, vous aimeriez pouvoir afficher les réponses à votre façon pour cela nous allons créer ce fichier:
(Voir exemple)

```
<html>
<head>
<title>Résultat du sondage PHP Facile !</title>
</head>
<body>
<center>A la question:<h2>Comment avez vous découvert PHP Facile!</h2>
   il a été repondu:<br><br>
<table border="1">
  <tr>
   <td><b>Pseudo</b></td>
```

```
   <td><b>Réponse</b></td>
  </tr>
<?php
function reponse($texte) {
    switch ($texte) {
       case "presse"  : return "Par la presse spécialisée"; break;
       case "annuaire": return "Par un annuaire"; break;
       case "moteur"  : return "Par un moteur de recherche"; break;
       case "forum"   : return "Par un forum"; break;
       case "news"    : return "Par les news"; break;
       case "ami"     : return "Par un ami"; break;
       case "autre"   : return "Par un autre moyen"; break;
    }
}

$serveur     = "localhost";
$utilisateur = "root";
$motDePasse  = "";
$base        = "mabase";

mysql_pconnect($serveur, $utilisateur, $motDePasse);
mysql_select_db($base) or die("Connexion ratée");

$requete = mysql_query("SELECT * FROM sondage");
while ($ligne = mysql_fetch_array($requete)) {
   print "<tr>
   <td>".$ligne["nom"]."</td>
   <td>".reponse($ligne["resultat"])."</td>
   </tr>";
}
?>
</table>
</body>
</html>
```

Remarques:

1. Le début et la fin du fichier sont de l'HTML "standard", les résultats seront affichés dans un tableau HTML.
2. Nous avons crée une fonction reponse() qui remplacera les termes presse, moteur... par du texte plus agréable.
3. Une fois le corps du tableau écrit, on fait une boucle while (tant que) profitant du fait que la fonction mysql_fetch_array() renvoie faux quand il n'y a plus de résultat à exploiter.
4. La requête sélectionne tous les éléments de la table. $ligne, correspond à une ligne de la table, on récupère d'abord le champ nom puis le champ resultat que l'on passe à la fonction reponse().

La fonction mysql_fetch_array()

Celle-ci retourne les divers champs d'une requête sous forme d'un tableau associatif (c'est cette propriété que nous avons utilisé dans l'exemple précédent). Il faut toutefois noter qu'elle retourne également (en même temps) les champs sous la forme d'un tableau indexé, le premier champ retourné par la requête est également disponible via $ligne[0]. Ceci peut s'avérer utile lorsque l'on n'a pas explicitement donné de nom au champ retourné par la requête (comme cela peut arriver avec une requête du type `"SELECT SUM(prix) FROM matable").`

Remarques:

1. Il est toujours possible de fixer le nom d'un champ retourné par une requête SQL (comme par `exemple "SELECT SUM(prix) AS 'total' FROM matable"` où la somme sera dans un champ appelé "total").

L'utilité de la base de données ne fait ici aucun doute, on aurait pu stocker les résultats dans un fichier texte mais il est plus simple d'utiliser une base de données si par la suite on veut faire des statistiques sur ces résultats.