III Les notions de langages objets

Pour l'instant, le javascript ne paraît pas très différent d'autres langages et ne donne pas l'impression d'offrir de fonctionnalités plus intéressantes.
La gestion des événements et des objets en est une principale.

III.1 Les événements

La gestion des événements est une particularité de ce langage, il va permettre d'ajouter du dynamisme à vos pages.

a. Intérêt

Dans des langages comme le PHP, la « seule » interaction possible avec l'utilisateur est le clic de souris qui permet de soumettre un formulaire. J'omets volontairement des interactions à l'environnement de l'utilisateur (langue, navigateur …).
Grâce aux événements, vous allez pouvoir activer des programmes, effectuer des actions et ceci en fonction d'une gamme d'interactions engendrée par le comportement de l'utilisateur sur votre site beaucoup plus large.

b. Définition et syntaxes

Selon le dictionnaire « Petit Robert », un événement est :
« un fait auquel vient d'aboutir une situation ».
L'utilisateur effectue une action qui est interceptée par le navigateur client et qui engendre le déclenchement d'une partie de code d'un programme.
Le plus souvent, les événements sont des actions engendrées par la manipulation de la souris (survol, clic …). Ils peuvent aussi être liés au clavier, au chargement d'une page …
Les événements sont rattachés aux objets de votre page HTML et lorsque ces objets sont manipulés, l'interpréteur javascript intercepte cet événement pour effectuer l'action définie par le programme.

La syntaxe des événements en javascript est précise.

Syntaxe

```
onEvenement="fonction() ou code javascript"
```

Les guillemets peuvent être remplacés par les simples cotes (‘ sous le chiffre 4 du clavier).
Les événements s’insèrent dans le code html de vos pages web.
La gestion des événements est quelque chose d’assez complexe. Le navigateur possède un gestionnaire d’événements qui va travailler sur un objet nommé event. Cet objet propose différents types d’événements qui ont chacun des propriétés particulières. Il existe des « raccourcis » qui évitent d’utiliser cet objet et ces méthodes en les nommant explicitement, ce qui est assez lourd. Par exemple onClick est plus pratique à utiliser.

Attention
Il existe une liste normalisée des événements gérés par les navigateurs. Cette normalisation n’a pas été utilisée par Microsoft sur son navigateur Internet Explorer. Certains événements normalisés sont gérés d’autres non, d’autres sont propres à Microsoft … Ceci n’est pas fait pour faciliter les choses. Il est malgré tout conseillé de n’utiliser que ce qui est général et normalisé sauf si vous décidez de faire des codes différents pour IE et les autres navigateurs.

Il existe 4 grands types d’événements. Ils vont êtres étudiés dans les 4 paragraphes suivants.

c. Les événements liés à la souris

La liste de ces événements est présentée dans le tableau qui suit.

Evénement	Comportement
onClick	Un clic sur un des boutons de la souris
onDblClick	Un double clic sur un des boutons de la souris
onMouseDown	Un bouton de la souris reste enfoncé
onMouseUp	Un bouton de la souris est relaché
onMouseOver	Le curseur de la souris survole une zone
onMouseOut	Le curseur de la souris quitte la zone survolée
onMouseMove	Le curseur de la souris est en mouvement

Ces événements sont associés à un ou plusieurs objets qui sont représentés par une balise HTML (bouton, lien hypertexte …). Le tableau suivant indique avec quel type d’objet l’événement peut être associé. Tout cela sera présenté de manière pratique plus loin avec les multiples exemples de codes.

Evénement	Association objet	Correspondance html
onClick	Link, document, formulaire	a, body, input
onDblClick	Link, document, area	a, body, input
onMouseDown	Link, document, formulaire	a, body, input
onMouseUp	Link, document, formulaire	a, body, input
onMouseOver	Link, area	a, area
onMouseOut	Link, area, formulaire	a, area, input
onMouseMove	Pas d'objet particulier	

Attention

Une fois de plus Microsoft ne fait pas comme tout le monde. La numérotation des boutons de la souris ne suit pas la norme ce qui est très ennuyeux lors des tests pour déterminer sur quel bouton l'utilisateur a appuyé.

Evénement	Norme (netscape, mozilla, opera ...)	Microsoft (Internet Explorer)
Clic gauche	1	1
Clic milieu	2	3
Clic droit	3	2

d. Les événements liés au clavier

La liste de ces événements est présentée dans le tableau qui suit.

Evénement	Comportement
onKeyDown	Une touche du clavier est enfoncée
onKeyUp	Une touche du clavier est relachée
onKeyPress	Une touche du clavier est enfoncée puis relachée

Ces événements sont associés à un ou plusieurs objets qui sont représentés par une balise HTML (bouton, lien hypertexte …). Le tableau suivant indique avec quel type d'objet l'événement peut être associé.

Evénement	Association objet	Correspondance html
onKeyDown	Lien, image, document, form	a, img, body, text, textarea
onKeyUp	Lien, image, document, form	a, img, body, text, textarea
onKeyPress	Lien, image, document, form	a, img, body, text, textarea

e. Les événements liés aux sélections

Il existe des éléments dans vos pages HTML (boutons, frame, …) qui peuvent être sélectionnés. Lorsqu'ils le sont, on parle de **focus**. La zone active est alors la cible des frappes, clics de souris …
A contrario, la zone qui perd le focus se trouve dans un état « désélectionné » on parle de **blur**.

La liste de ces événements est présentée dans le tableau qui suit.

Evénement	Comportement
onFocus	La zone rattachée à l'événement est sélectionnée
onBlur	La zone rattachée à l'événement est désélectionnée
onSelect	La zone rattachée à l'événement a été sélectionnée
onDragDrop	Un objet est sélectionné et glissé vers la fenêtre
onMove	La fenêtre active est déplacée
onResize	La fenêtre active est redimensionnée
onSubmit	Le bouton de formulaire de type submit a été pressé
onReset	Le bouton de formulaire de type reset a été pressé

Ces événements sont associés à un ou plusieurs objets qui sont représentés par une balise HTML (bouton, lien hypertexte …). Le tableau suivant indique avec quel type d'objet l'événement peut être associé.

Evénement	Association objet	Correspondance html
onfocus	fenêtre, frame, formulaire	body, frame, input
onBlur	fenêtre, frame, formulaire	body, frame, input
onSelect	formulaire	Input, textarea
onDragDrop	fenêtre, frame	body, frame, frameset
onMove	fenêtre, frame	body, frame, frameset
onResize	fenêtre, frame	body, frame, frameset
onSubmit	formulaire	form
onReset	formulaire	form

L'utilisation de onBlur permet, par exemple, d'effectuer un test après une saisie dans une zone de formulaire.

Remarque
Evitez d'utiliser des boîtes de dialogue pour afficher les erreurs car souvent il est difficile de les fermer. Privilégier un affichage dans la zone de saisie par exemple.

f. Les autres événements

Il existe d'autres événements liés à un certain nombre d'objet HTML. La liste de ces événements est présentée dans le tableau qui suit.

Evénement	Comportement
onAbort	Le changement d'une image a été arrêtée
onChange	Un élément du formulaire a été modifié
onError	Erreur lors du chargement d'une image ou d'une fenêtre
onLoad	Le chargement d'une image ou d'une page est terminé
onUnLoad	La page est remplacée par une autre

Ces événements sont associés à un ou plusieurs objets qui sont représentés par une balise HTML (bouton, lien hypertexte …). Le tableau suivant indique avec quel type d'objet l'événement peut être associé.

Evénement	Association objet	Correspondance html
onAbort	image	img
onChange	formulaire	Input, textarea, select
onError	image, fenêtre, frame	Img, body, frame, frameset
onLoad	image, fenêtre, frame	Img, body, frame, frameset
onUnLoad	fenêtre, frame	body, frame, frameset

g. Exemple

La page HTML suivante permet de tester la validité d'un login saisi dans un formulaire. Ce login ne doit pas dépasser la taille de 8 caractères. Il est évidemment possible de limiter la zone de saisie en HTML avec l'option maxlenght mais le javascript va permettre de dynamiser graphiquement cette partie de code HTML.

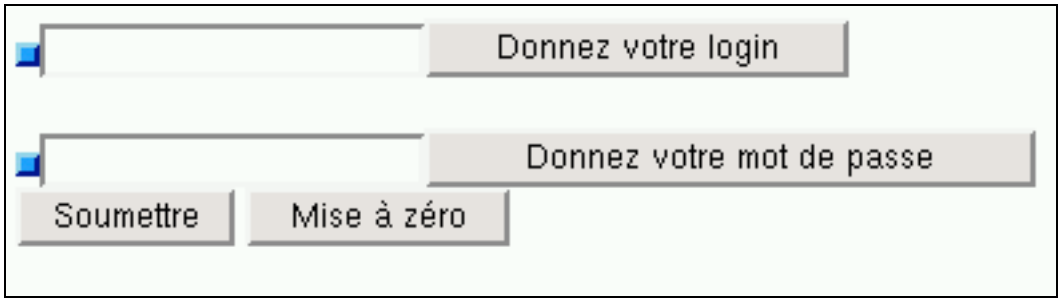
	Code
1	<HTML>
2	<HEAD>
3	<script language="javascript">
4	function verifie()
5	{
6	if(window.document.formu.login.value.length > 8)
7	{
8	window.document.formu.login.value="Login limité à 8 caracteres";

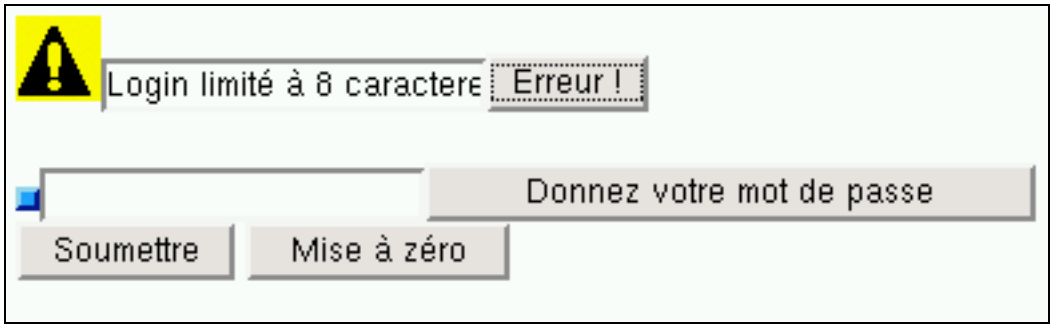
```
9    window.document.formu.but1.value="Erreur !";
10   window.document.images[0].src="IMAGE/attention_warning.gif";
11  }
12 }
13 </script>
14 </HEAD>
15 <BODY>
16 <B>
17 <FORM action="login.php" method=post name=formu>
18 <IMG SRC="IMAGE/bluere.gif">
19 <INPUT type=text name=login onBlur="verifie()">
20 <INPUT TYPE=BUTTON NAME=but1 VALUE="Donnez votre login">
21 <BR>
22 <BR>
23 <IMG SRC="IMAGE/bluere.gif">
24 <INPUT type=text name=password>
25 <INPUT TYPE=BUTTON NAME=but2 VALUE="Donnez votre mot de passe">
26 <BR>
27 <INPUT type=submit value="Soumettre">
28 <INPUT type=reset value="Mise à zéro">
29 </FORM>
30 </BODY>
31 </HTML>
```

Explications

Création d'une fonction qui va être associée à l'événement onBlur (désélection) à la ligne 19.
Cette fonction permet lorsque l'utilisateur a sélectionné (focus) puis désélectionné (blur) le champ du formulaire de saisi du login de vérifier si ce login ne possède pas plus de 8 caractères.
Ce test s'effectue ligne 6.
Les objets sont le document qui possède un formulaire du nom de formu (voir l'attribut name ligne 17). Ce formulaire possède lui même un champ texte du nom de login (voir l'attribut name ligne 19). Cet objet possède une propriété value d'où ligne 8 la ligne qui permet de « pointer » sur la valeur du champ du login.
Les autres objets sont créés suivant le même système.
Ceci est vu plus loin.
Si le login possède plus de 8 caractères, un message d'erreur est inséré dans la zone de saisie du login (ligne 8), le bouton (ligne 20) prend pour valeur « Erreur ! » (ligne 9) et l'image devant la zone de saisie (ligne 18) change et prend la valeur de celle indiquée ligne 10.

Captures écran





III.2 Le monde objet

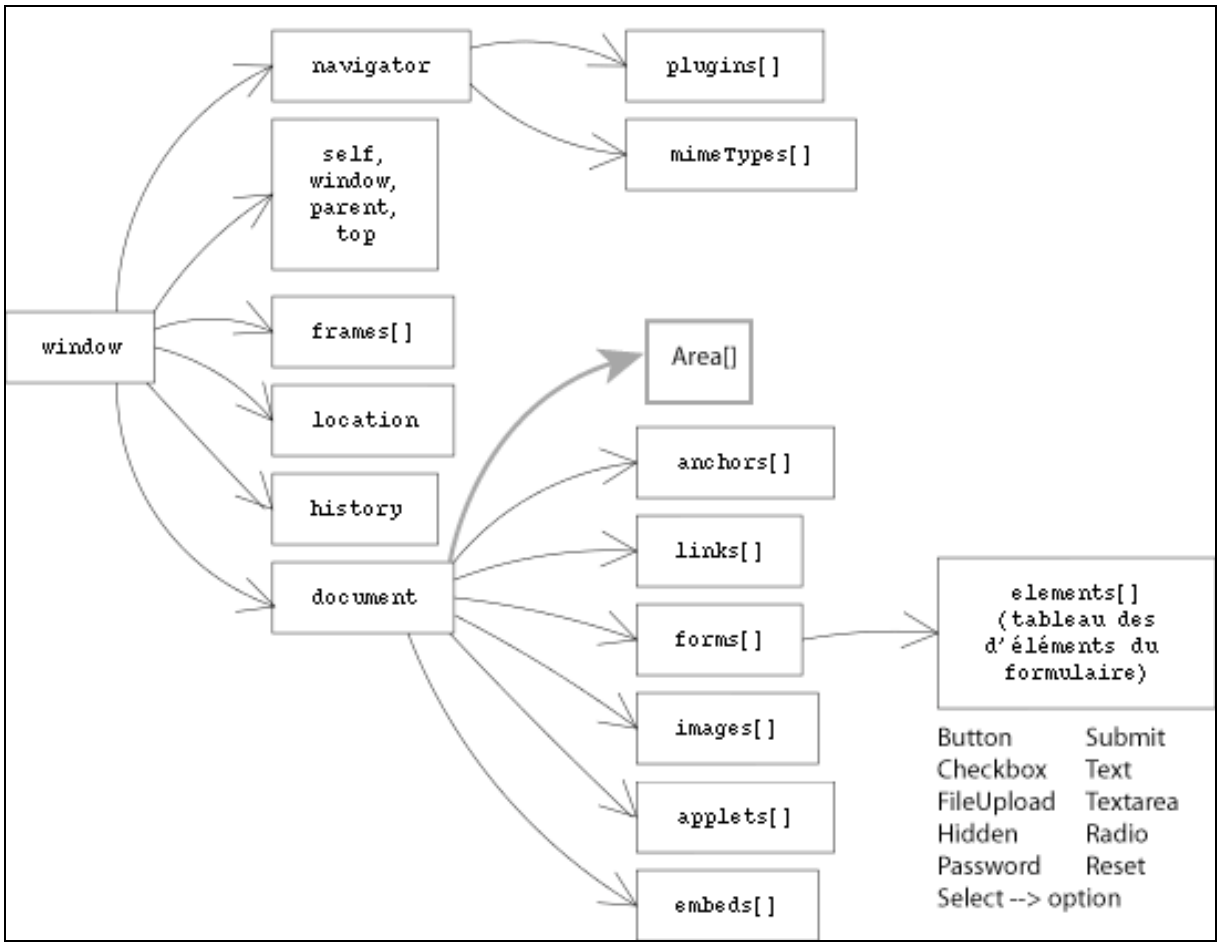
L'intérêt du langage javascript est de pouvoir travailler sur les objets contenus dans les pages HTML. Ces objets sont associés à des propriétés et à des méthodes qui permettent de les manipuler.
A chaque objet est associé un certains nombres d'éléments qui permettent d'atteindre et de modifier ces objets.

a. Les objets

Définition
Un objet est une enveloppe, un tiroir dans lequel on va pouvoir ranger un grand nombre de choses qui lui seront liés (variables, structures …).
Il existe des objets prédéfinis par les normes javascript, mais il est possible d'en créer soit même.

A ces objets sont associés des propriétés et des méthodes nécessaires pour leur manipulation. L'organisation des objets qui constituent une page Web est appelée DOM (Document Object Model).
Ces objets ont une hiérarchie. Certains sont les pères d'autres, certains sont des fils …

L'objet qui se situe au sommet de la hiérarchie est l'objet **window**. Tous les autres objets sont des fils de cet objet.
Pour bien comprendre la façon dont ce langage est construit et comment vous pouvez atteindre les objets inclus dans vos pages HTML, il est nécessaire de garder en mémoire l'organisation des objets prédéfinis de Javascript (DOM javascript).
Cette organisation est présentée dans le schéma suivant :



b. Les propriétés

Définition

Une propriété est une information ou attribut qui caractérise un objet. Un objet peut posséder plusieurs propriétés.

Les valeurs des propriétés associées à un objet peuvent être modifiées. Par exemple, un document HTML est caractérisé par l'objet **document** auquel un certain nombre de propriétés sont associées comme la couleur du texte, la couleur du fond d'écran …
A chaque propriété une valeur est associée. Cette valeur peut être définie (elle existe) ou non définie (elle n'existe pas valeur undefined).

La valeur existe si, dans votre document, la propriété de l'objet est utilisée et est renseignée. Dans le cas contraire, elle est non définie.

Afin de lister les propriétés de ces objets, la boucle for peut être utilisée. Voir l'exemple ci-après.

Attention

Toutes les propriétés ne sont pas forcément listable. En effet, un certain nombre d'entre elles peuvent être en mode « DontEnum », « internal » …

La norme ECMA262 le montre. Voir le tableau ci-dessous.

Attribute	*Description*
ReadOnly	The property is a read-only property. Attempts by ECMAScript code to write to the property will be ignored. (Note, however, that in some cases the value of a property with the ReadOnly attribute may change over time because of actions taken by the host environment; therefore "ReadOnly" does not mean "constant and unchanging"!)
DontEnum	The property is not to be enumerated by a `for-in` enumeration (section 12.6.4).
DontDelete	Attempts to delete the property will be ignored. See the description of the `delete` operator in section 11.4.1.
Internal	Internal properties have no name and are not directly accessible via the property accessor operators. How these properties are accessed is implementation specific. How and when some of these properties are used is specified by the language specification.

Exemple

Le programme suivant permet de lister les propriétés visibles de l'objet **navigator**.

Code

```
1  <HTML>
2  <HEAD>
3  <TITLE>Les proprietes de l'objet document</TITLE>
4  <script language="javascript">
5  for (var i in window.navigator)
6  {
7   document.write("navigator" + "." + i + "=" + window.navigator[i] + "<BR>");
8  }
9  </script>
10 </HEAD>
11 <BODY>
12
13 </BODY>
14 </HTML>
```

Explications

La ligne 4 à 9 permet d'insérer un code javascript qui est interprété par le client au moment du chargement du programme HTML.

De la ligne 5 à la ligne 8, la boucle « for » permet de récupérer les propriétés (contenu à chaque passage dans la variable i ligne 5).
La ligne 7 permet d'afficher le mot navigator suivi du nom de la propriété récupérée (i) ainsi que la valeur qui lui est associée (window.navigator[i]).
La valeur peut être une chaîne de caractère mais aussi une fonction voir un autre objet.

Captures écran

```
navigator.appCodeName=Mozilla
navigator.appName=Netscape
navigator.appVersion=5.0 (X11; en-US)
navigator.language=en-US
navigator.mimeTypes=[object MimeTypeArray]
navigator.platform=Linux i686
navigator.oscpu=Linux i686
navigator.vendor=
navigator.vendorSub=
navigator.product=Gecko
navigator.productSub=20020722
navigator.plugins=[object PluginArray]
navigator.securityPolicy=
navigator.userAgent=Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.1b) Gecko/20020722
navigator.cookieEnabled=true
navigator.javaEnabled= function javaEnabled() { [native code] }
navigator.taintEnabled= function taintEnabled() { [native code] }
navigator.preference= function preference() { [native code] }
```

Attention

Le script fonctionne bien avec mozilla et netscape par contre avec opera ou konqueror, l'objet navigator ne retourne rien.

- Autre notation

Vous pouvez aussi atteindre les valeurs associées aux propriétés d'un objet directement. Dans l'exemple portant sur le login (ci-dessus), la valeur du login sais est récupérée grâce à la ligne *window.document.formu.login.value*.

L'objet **window** possède un objet fils **document** (voir le DOM) qui possède lui même un objet **formu** (nom du formulaire). L'objet **formu** contient un élément représenté par une balise input qui porte le nom de login. Une des propriétés de cet élément est value qui permet de retourner ou d'atteindre la valeur associée à cet élément.
La notation *window.document.forms[0].elemenst[0].value* est aussi valide. Cette notation est plus générale car, quelque soit le nom du formulaire et de la balise input elle représente le premier formulaire et la première balise de ce formulaire (Voir DOM).
L'objet **forms** est une liste. A l'indice 0 se situe le premier formulaire du document HTML (représenté par la balise form), à l'indice 1 se situe le deuxième formulaire de ce même document …
A l'intérieur de ces formulaires il existe des éléments représentés par la liste elements. A l'indice 0 se situe le premier élément et ainsi de suite.

Remarque

Si vous devez traiter plusieurs propriétés d'un même objet, vous pouvez utiliser l'instruction with qui permet de raccourcir la saisie en précisant que ce qui suit concerne l'objet indiqué dans le bloc with.

Exemple

Reprenons l'exemple portant sur le login.

Code

```
1  <HTML>
2  <HEAD>
3  <script language="javascript">
4  function verifie()
5  {
6  with(window.document.forms[0].elements[0])
7   if(value.length > 8)
8   {
9   value="Login limité à 8 caracteres";
10  window.document.formu.but1.value="Erreur !";
11   window.document.images[0].src="IMAGE/attention_warning.gif";
12  }
13 }
14 </script>
15 </HEAD>
16 <BODY>
17 <B>
18 <FORM action="login.php" method=post name=formu>
19 <IMG SRC="IMAGE/bluere.gif">
20 <INPUT type=text name=login onBlur="verifie()">
21 <INPUT TYPE=BUTTON NAME=but1 VALUE="Donnez votre login">
22 <BR>
```

```
23  <BR>
24  <IMG SRC="IMAGE/bluere.gif">
25  <INPUT type=text name=password>
26  <INPUT TYPE=BUTTON NAME=but2 VALUE="Donnez votre mot de passe">
27  <BR>
28  <INPUT type=submit value="Soumettre">
29  <INPUT type=reset value="Mise à zéro">
30  </FORM>
31  </BODY>
    </HTML>
```

c. Les méthodes

Définition

Une méthode est une fonction propre à un objet. La plupart du temps, la méthode est suivie de parenthèses. document.write() permet d'afficher du contenu dans votre document. La méthode associée à l'objet document est write().

d. Création d'objets

Javascript utilise les fonctionnalités des langages objets, il permet donc de ne pas se contenter des objets, méthodes et autres éléments prédéfinis dans le langage. En effet vous pouvez très bien en définir de nouveaux.
Lors des paragraphes précédents, cette partie a été évoquée. Deux possibilités sont envisagées :

- La méthode énumérative
- L'utilisation de l'opérateur this

➢ La méthode énumérative

Il suffit de définir l'objet et de lui associer des propriétés, des valeurs et des méthodes.
L'inconvénient de ce système est de ne pouvoir ensuite, récupérer la définition pour un autre objet du même type, il est alors nécessaire de tout redéfinir pour ce nouvel objet.

Exemple

Code

```
1  <HTML>
2  <HEAD>
```

```
3  </script>
4  </HEAD>
5  <BODY>
6  <script language="javascript">
7  aliment= {
8  nom:"cassoulet",
9  prix:"3",
10 composition:{ingredient1:"sauce",ingredient2:"saucisses",ingredient3:"haricots"},
11 origine:"sud-ouest"
12 };
13 for(var i in aliment)
14 {
15  if(aliment[i] == "[object Object]")
16  {
17   for(var j in aliment[i])
18   {
19    document.write("aliment." + i + "." + j + " = " + aliment[i][j] + "<BR>");
20   }
21  }
22  else
23  {
24   document.write("aliment." + i + " = " + aliment[i] + "<BR>");
25  }
26 }
27
28 </BODY>
29 </HTML>
30
```

Explications

Les lignes 7 à 13 permettent d'associer à l'objet aliment des propriétés et leurs valeurs.
Ligne 10 la propriété composition contient elle aussi des propriétés (ingredient?).
Les lignes 14 à 27 permettent d'afficher le contenu de cet objet. Du fait de le présence de sous-propriétés, il est nécessaire de tester le contenu récupérer pour n'afficher que les valeurs (ligne 16).

Captures écran

```
aliment.nom = cassoulet
aliment.prix = 3
aliment.composition.ingredient1 = sauce
aliment.composition.ingredient2 = saucisses
aliment.composition.ingredient3 = haricots
aliment.origine = sud-ouest
```

Si nous voulions utiliser à nouveau cette structure d'objet, il serait nécessaire de redéfinir un nouvel objet et toutes les propriétés. Ce qui n'est pas très souple.

- L'opérateur this

Cet opérateur permet de créer un objet « souche » qui peut ensuite être réutilisé pour la définition d'un objet de même contenu.
Le nouvel objet est une instanciation de celle de l'objet souche.

Exemple

L'exemple précédent devient alors :

Code

```
1  <HTML>
2  <HEAD>
3  </script>
4  </HEAD>
5  <BODY>
6  <script language="javascript">
7
8  function aliment(n,p,c,o)
9  {
10  this.nom=n;
11  this.prix=p;
12  this.composition=c;
13  this.origine=o;
14 }
15
16 function contenu(i1,i2,i3)
17 {
18  this.ingredient1=i1;
19  this.ingredient2=i2;
20  this.ingredient3=i3;
21 }
```

```
22
23
24  cassoulet = new aliment("cassoulet",3,recette,"sud-ouest");
25  recette = new contenu("sauce","saucisses","haricots");
26
27  for(var i in cassoulet)
28  {
29   if(cassoulet[i] == "[object Object]")
30   {
31    for(var j in cassoulet[i])
32    {
33     document.write("cassoulet." + i + "." + j + " = " + cassoulet[i][j] + "<BR>");
34    }
35   }
36   else
37   {
38    document.write("cassoulet." + i + " = " + cassoulet[i] + "<BR>");
39   }
40  }
41
42  </script>
    </BODY>
    </HTML>
```

Explications

Les lignes 8 à 14 permettent de créer une fonction **aliment** qui gère les propriétés de « niveau 1 ». La propriété composition contient à nouveau d'autres propriétés qui sont définies dans la fonction **contenu** (ligne 16 à 21).

Notez bien qu'à aucun moment dans ces déclarations nous n'avons indiqué de valeur. Ces déclarations fournissent uniquement la structure des objets.

Les lignes 24 à 25 permettent elles, de déclarer et d'associer aux propriétés de ces objets des valeurs.

La ligne 24 permet de définir l'objet **cassoulet** qui est constitué d'un ensemble de propriétés dont les valeurs sont indiqués entre parenthèses. Ces valeurs doivent être dans le même ordre qu'indiqué dans la définition de l'objet. L'objet **cassoulet** est une instanciation de l'objet **aliment**.

La ligne 25 permet d'associer aux propriétés de l'objet **recette** leur valeur. L'objet **recette** est une instanciation de l'objet **contenu**.

Captures écran

```
cassoulet.nom = cassoulet
cassoulet.prix = 3
cassoulet.composition.ingredient1 = sauce
cassoulet.composition.ingredient2 = saucisses
cassoulet.composition.ingredient3 = haricots
cassoulet.origine = sud-ouest
```

Si vous désirez créer un nouveau « plat » dont la structure est équivalente à celle créée dans l'objet « souche », vous n'êtes pas obligé de tout redéfinir. Il suffit de reprendre les deux lignes 24 et 25.

- Les différentes notations possibles

Il est possible d'atteindre les valeurs des objets de plusieurs manières. Les deux syntaxes suivantes renvoient la valeur sauce :

Syntaxe

```
document.write(cassoulet.nom);
document.write(cassoulet. ["nom"]);
```

Si l'objet principal est composé d'un autre objet qui possède des propriétés, les syntaxes suivantes sont possibles et renvoeint le même résultat :

Syntaxe

```
document.write(cassoulet.composition.ingredient1);
document.write(cassoulet.composition.["ingredient1"]);
document.write(cassoulet.["composition"].ingredient1);
document.write(cassoulet.["composition"].["ingredient1"]);
```

En combinant deux notations cela propose bien quatre possibilités ($2^e2=4$).

- Destruction des valeurs

Il est possible de supprimer la valeur d'une propriété en lui associant le mot clé **undefined** ou en utilisant l'opérateur **delete**.

Syntaxe

```
cassoulet.nom=undefined;
delete cassoulet.nom;
```

Attention

Il y a une différence entre les deux méthodes de destruction.
L'utilisation de delete supprime la propriété, elle disparaît de l'affichage (elle n'est alors plus associée à l'objet) tandis que l'affectation de la valeur undefined conserve la propriété, l'affichage renvoie alors la valeur undefined.