Javascript aux Ateliers Nomades - Session Mars 2012

Table des matières

Programmation

Un programme est un ensemble d'**instruction**.

Une instruction est une commande «comprise» (interprétée/exécutée) par la machine cible .

Analogie

Pour un réveil on peut imaginer les instructions:

- réveilleMoi()
- ajusterHeure()

Ces instructions nécessitent en général des informations supplémentaires: A quelle heure faut-il te réveiller? Quelle est l'heure juste actuelle?. Mais ce n'est pas obligatoire! Par exemple l'instruction arrêteDeSonner() n'a besoin d'aucune information supplémentaire...

Ces informations sont appelées des **paramètres**. Exemples de nos instructions avec les paramètres associés:

- réveilleMoi(8:00, sonnerie-musicale, répéter-trois-fois)
- ajusterHeure(12, 30, 07)

Le type (texte? nombre? date?...) et le nombre des paramètres dépendent de la spécification de chaque instruction. Certaines instructions peuvent proposer des paramètres optionnels : s'il ne sont pas fournis, l'instruction utilise des valeurs par défaut (sonnerie X, et ne-pas-répéter) par exemple.

Pour réaliser un programme, il suffit donc d'utiliser les bonnes instructions dans le bon ordre avec les bons paramètres...

Les formants de base

Il existe trois formants de base suffisants pour construire n'importe quel programme.

- La séquence
- L'alternative
- La boucle

La séquence: une séquence d'instruction sera toujours interprétée dans le même ordre lors de chaque déroulement du programme.

L'alternative permet de réaliser des chemins différents selon le contexte. L'alternative commence donc par un test, ce test est soit positif, soit négatif:

```
Si (test à effectuer)
        Instruction(s) si le test est vrai
Sinon
        Instruction(s) si le test est faux
Fin
```

La boucle permet de répéter des instructions, soit un nombre déterminé de fois, soit tant qu'une condition est vraie:

Structure avec un nombre déterminé:
Faire de 1 à 10 fois
    Instructions à répéter
Fin, revenir à Faire.

Structure tant qu'une condition est vraie
Si (condition est vraie) alors
    Instruction(s) à répéter
Fin et revenir au Si

Exemple de boucle : afficher les puissances de 2 (entre 2 et 1024 inclus)

```
// Définition d'une variable et affectation de la valeur 2
var nombre = 2;
// Tant que nombre <= 1024
while (nombre<=1024){
      document.write(nombre);
      document.write("<br/>");
      /* Calcul avec une variable et stocker le
      résultat dans la variable: */
      nombre = nombre * 2;
}
```

Programmation orienté objet

L'objet et la programmation orientée objet

Un objet peut être vu comme un ensemble d'informations (propriétés) et de responsabilités (méthodes).
Un réveil par exemple aura comme propriété l'heure courante, l'heure de réveil, le choix de la sonnerie, le nombre de répétitions et comme méthodes réveilleMoi(heure), arrêteToi (), changerHeure(heure), changerChoixSonnerie(choix), etc.

La modélisation objet est moins implicite et accessible que la programmation procédurale * ou événementielle**. Il est en effet plus difficile de modéliser un système en le découpant en objets dotés de propriétés et méthodes que par fonctionnalités ou événements. Le succès de la programmation orientée objet vient de la robustesse des applications réalisées en objets, faciles à maintenir, à réutiliser.

*Programmation procédurale : la modélisation se fait en découpant le système en procédures ou fonctions principales en sous-procédures ou sous-fonctions. (une «procédure» est une «fonction» qui ne retourne pas de résultat mais réalise un travail)

**Programmation événementielle : la modélisation se fait en découpant le système par ses événements et interactions.

Remarque : ces trois types de programmation sont utilisés pour nommer le paradigme de programmation de la méthode ou de l'outil ou du développement utilisé.
Dans tout développement il y aura des procédures/fonctions et sous-procédures/sous-fonctions, des événements, et en général des objets!

Exemple: un développement en JavascriptJavascript avec des sous-procédures. L'application utilisera nécessairement les objets fournis par l'environnement (window par exemple) et des événements (onMouseOver par exemple).

Conclusion - Recommandation

Une réalisation en Javascript peut être faite en utilisant les objets existant ou fournis par une librairie, sans devoir en modéliser de nouveau.

L'approche orientée objet est souvent remplacée par une approche événementielle dans les projets de sites web. Cela permet aux développeurs débutants de réaliser rapidement des applications peu complexes et ne pose pas de problème particulier.

Il est par contre **fortement recommandé** d'utiliser la programmation orientée objet lors du développement d'application plus complexe! En effet les efforts de l'apprentissage objet seront vite récompensé par la valeur ajoutée en terme de réutilisation, de maintenance, etc.

Outils

Navigateur

Le navigateur interprète le Javascript. Les erreurs ne sont pas systématiquement affichées! Pour Mozilla, menu Outils -> Console d'erreurs
Pour Safari, menu Développement -> Afficher la console des erreurs
Pour IE, menu Outils -> Options Internet, dans le groupe Navigation, cocher/décocher l'option correspondante (Arrêter le débogage de script ou Afficher les erreurs dans la page, selon les versions)

Editeur de texte

Si un éditeur de texte est suffisant, il est évident que la coloration de la syntaxe ou l'auto-complétion de code permettent de programmer plus rapidement et sûrement! Dreameweaver est recommandé s'il est déjà connu par l'utilisateur! D'autres éditeurs de texte, Smultron ou Editplus restent efficaces, car simples et légers, mais ils ne proposent pas d'auto-complétion de code.

Débogueur

Lorsque un programme est développé, il faut disposer d'outil permettant rapidement de trouver les éventuels erreurs. Chaque navigateur indique (cf. Navigateur ci-dessus) la ligne de l'erreur et le message d'erreur. Pour déboguer plus finement, il faut utiliser des outils plus complets, tels que Firebug pour Mozilla (Firebug lite pour IE, Safari, ..), qui permettent d'interpréter ligne par ligne un programme, de suivre les valeurs d'une variable, etc.

Références

Un site ou livre de référence est nécessaire lorsque l'on développe! Ils permettent de retrouver une syntaxe, une fonction ou méthode. Les sites de référence fournissent également des exemples de mise en oeuvre.

Il faut distinguer :
- la référence du langage (les IF, les opérateurs, ...)
- le DOM (Domain Object Model) qui fournit le nom des objets disponibles avec leurs propriétés et méthodes
- des tutoriaux qui expliquent et proposent des exemples complets

Le tableau de bord indique quelques site de références. Un bon site de référence pour un développeur est un site dans lequel il trouve rapidement son information! Certain sites sont très sobres, d'autres très interactifs, à chacun de trouver celui qui lui convient!

Syntaxe

L'objectif de cet partie est de relever quelques éléments particuliers, et non de remplacer les nombreux supports disponibles!

Les instructions sont séparées par des point-virgules ;

Les accolades { et } permettent de définir un bloc d'instruction, par exemple dans un if ou dans une boucle:

```
if (condition) {
      instructions
} else {
      instructions
}

for (i=0; i<10; i++){
      instructions
}
```

Les chaines de caractères doivent toujours être délimitées par des guillemets ou des apostrophes: `"une chaine"` ou `'une chaine'`.

Une fonction ou une méthode est toujours utilisée avec ses parenthèses, même si aucun paramètre n'est fourni ou nécessaire:

```
reveilleMoi();
document.write("message à afficher");
```

Variables

Une variable est emplacement mémoire dans lequel une valeur (3, une chaine, un objet, ...) peut être stockée. Le développeur utilise le nom de la variable, le navigateur utilisera la valeur courante stockée dans la variable.

Définition d'une variable:

```
var nombre;
```

Affectation d'une variable:

```
nombre = 4;
```

Utilisation d'une variable:

```
document.write(nombre);
```

Modification d'une variable (= nouvelle affectation..):

```
nombre = 5;
```

Utilisation et modification (double la valeur dans nombre):

```
nombre = nombre * 2;
```

Nom des variables: il est fortement recommander de nommer les variables de manière à ce que le nom reflète le contenu: `nombre`, `taux`, `salaireNet`, `prixTotal`, etc... Le nom

d'une variable ne peut contenir de caractères spéciaux ou espace! Le nom d'une variable ne peut commencer par un chiffre.

Opérateurs
Il existe des opérateurs pour chaque type de données (chaine de caractère, booléen, nombres, ...). Voici quelques exemples :

Opérateur d'affectation
```
=
```

Opérateurs de comparaison
```
==
!=
<
>
<=
>=
```

Opérateur de concaténation (permet de regrouper deux chaines en une seule chaine)
```
+
```

Exemple:
```
document.write(nombre);
document.write('<br/>');
```
peut être écrit:
```
document.write(nombre + '<br/>');
```

Opérateurs particuliers (+=, *=, /=, ++, --, etc..):
La ligne suivante:
```
nombre = nombre + 2; //ajoute 2 au contenu de nombre
```
Peut être écrite ainsi:
```
nombre +=2; // Ajoute 2 au contenu de nombre
```
Pour incrémenter une variable de 1 :
```
i++;
```

Fonctions et procédures
De nouvelles fonctions ou procédure peuvent être définies dans un programme. Elles permettent de structurer le code, de décomposer le programme pour faciliter la conception, la lecture, la réutilisation et la maintenance.

Exemple d'une fonction (sans paramètre)
```
function afficherNombre(){
	for (var i=0; i<10; i++)
	{
		response.write(nombre + '<br/>');
	}
}
```

Exemple d'une fonction (avec paramètres)
```
function afficherMessageErreur(message) {
	document.write('Erreur:');
	document.write(message);
```

```
	document.write('<hr/>');
}
```

Les deux exemples ci-dessus montrent des fonctions qui effectuent une suite d'instructions chaque fois qu'elles seront appelées.

Exemple:

```
afficherNombre();
afficherNombre();
```

Affichera deux fois la suite des nombres de 0 à 9:

```
0 (Début de la première suite)
1
...
8
9
0 (Début de la deuxième suite)
1
...
8
9
```

Les fonctions peuvent également calculer et retourner un résultat:

```
function tauxActuel() {
	var taux = 0;
	// Instructions, calculs pour le taux...
	return taux
}
```

Dans ce cas le résultat de la valeur peut être directement utilisé ou stocké dans une variable:

```
	montantNet = montantBrut * tauxActuel();
```

ou

```
	var taux = tauxActuel();
```

ou

```
	document.write(tauxActuel());
```

Remarque : différence entre une procédure et une fonction?
Une procédure réalise un traitement, une fonction retourne un résultat. C'est pour cela qu'il est judicieux et recommandé de nommer les procédures par des verbes (afficherNombre(), envoyerMail()) et les fonctions par des substantifs (donnéeValides(), tauxActuel(),...)

Fonctions existantes dans le langage Javascript
Javascript est fortement orienté objet, il n'existe qu'une douzaine de fonctions natives, telles que isNaN(), parseInt(), etc. (liste complète dans le tableau de bord). Toutes les autres «fonctions» sont en fait des méthodes fournies par des objets (cf. Programmation orientée objet)
Lorsqu'on utilise la «fonction» suivante:

```
alert('Vous devez saisir votre adresse');
```

On utilise en faite la méthode alert de l'objet window:

```
window.alert('Vous devez saisir votre adresse');
```

Le `window` est l'objet principal par défaut, il n'est donc pas nécessaire de l'écrire. C'est pour cela que les méthodes de `window` sont souvent confondues avec des fonctions...

Structure de donnée

Les tableaux

Une variable permet de stocker une valeur ou un objet, pour stocker un ensemble de valeurs ou d'objets, il faut définir une liste (un tableau à une dimension).

Soit `nombres`, un tableau permettant de stocker 12 nombres:

nombre 1	nombre 2	..			nombre 12

Pour modifier le 1er nombre:

```
nombres[0] = 7;
```

Le 2ième:

```
nombres[1] = 4;
```

Le dernier:

```
nombres[11] = 8;
```

Les crochets [et] permettent donc d'indiquer l'indice (adresse dans le tableau) qui doit être mise à jour ou utilisée.

Exemple: doubler la 6ième valeur:

```
nombres[5] *= 2;
```

Attention: l'indice commence à 0 et finit à nombre-d'éléments-moins-un! (comme pour les immeubles...)

L'indice est pratique pour réaliser une boucle sur l'ensemble du tableau:

```
for (i=0; i<12; i++) {
      document.write(nombres[i]);
}
```

Les tableaux associatifs

Il est également possible de définir des clés au lieu d'indices. Très utiles lorsque sont stockées un ensemble d'information plutôt que un ensemble de valeur de même type:

```
client['nom'] = 'Dupont';
client['prenom'] = 'Paul';
client['annee'] = 1999;
client['email'] = 'Paul.Dupont@dupont.com'';
```

On accède ainsi facilement aux informations:

```
document.write(client['nom']);
```

L'objet Array

Pour définir un tableau, on passe par l'objet Array:

```
var nombres = new Array();
```

```
var  client = new Array();
```

Cette objet fournit la propriété length et un ensemble de méthodes.

Mise en oeuvre

Intégration du Javascript dans le HTML

Avec la balise <script>

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

En référençant un fichier annexe

```
<script type="text/javascript" src="fonctions.js"></script>
```

Le fichier «fonctions.js» ne doit alors uniquement contenir du code javascript (pas de balises HTML).

Directement dans une balise HTML:

```
<img src=... onMouseOver="agrandir()">
```

Interactions avec le HTML

Pour accéder à un élément : document.getElementById("idElement")

Pour modifier la source de l'image ayant l'id logo:
document.getElementById("logo").src="logoNB.png";

Pour agrandir une div de 100px:

```
var taille = parseInt(document.getElementById("id_div").width);
taille = taille + 100;
document.getElementById("id_div").width=taille + 'px';
```

Une connaissance du DOM est essentielle pour utiliser les propriétés, méthodes et événements accessibles!