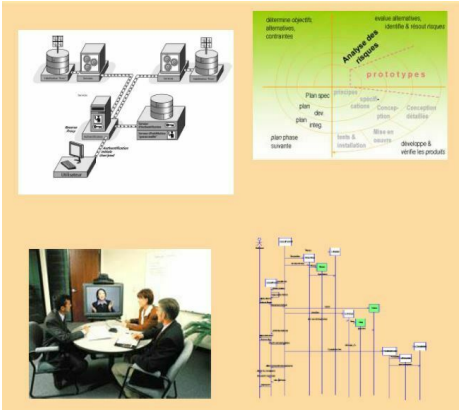
DÉCOUVERTE DU LANGAGE JAVASCRIPT P1 - SÉQUENCE « DÉVELOPPER DES PAGES WEB »

Appui technique

Introduction

Bonjour et bienvenue dans ce module consacré au langage Javascript.

Découverte du langage JavaScript P1 - Apprentissage - Séquence « Développer des pages Web »



Introduction à Javascript

1.1 Javascript, c'est quoi ?

Le langage de programmation JavaScript **côté client** est une extension au langage de description Html qui définit toute page Web. Le langage JavaScript peut rendre dynamique les pages Web notamment en accédant directement aux éléments de la page Html, en les manipulant et en interagissant avec l'utilisateur. Les scripts JavaScript, qui s'ajoutent ici aux balises Html, peuvent être comparés aux macros d'un traitement de texte en ce sens qu'ils permettent d'automatiser un certain nombre de tâches et de réaliser des contrôles de saisie.

JavaScript côté client peut faire beaucoup de choses :

- Ajouter, supprimer et modifier le contenu Html dynamiquement ;
- Interagir sur le code CSS ;
- Animer et ajouter des effets sur les textes, les images...
- Intercepter les événements (souris, clavier, timer...) ;
- Contrôler les saisies effectuées par l'utilisateur dans les formulaires Html ;
- Générer des menus dynamiques ;
- Détecter le type de navigateur de manière à adapter le rendu graphique ou l'ergonomie aux fonctionnalités spécifiques.

Les scripts JavaScript sont téléchargés avec la page ; ils peuvent être inclus directement dans le code source Html ou stockés dans des fichiers distincts reliés à la page Web. Après téléchargement depuis le serveur Web, ils vont être gérés et exécutés par un interpréteur JavaScript intégré dans le navigateur. Les instructions JavaScript seront donc traitées en direct et surtout sans retard par le navigateur (pas besoin de recharger la page ni d'allers-retours entre navigateur et serveur Web pour récupérer des ressources).

JavaScript côté client, utilisé seul, possède certaines limites qui font parties de ses avantages :

- Il ne peut pas accéder à des fichiers stockés sur le disque dur du poste utilisateur ;
- Il est limité à la manipulation du navigateur, du contenu de la page Web en cours ainsi que de la barre d'adresse du navigateur et des éventuelles informations passées en paramètres dans l'url de la page ;
- Il ne peut pas accéder directement aux bases de données du serveur Web ;
- Il n'est pas multitâche.

Mais l'arrivée du Html5 et ses API corrigent certains « manques » (balise Html <*canvas*>, base de données locale...).

La bibliothèque JavaScript **Ajax** permet d'accéder aux bases de données du serveur Web via des requêtes HTTP secondaires sans quitter la page Web en cours.

JavaScript côté serveur se développe beaucoup ces dernières années. L'outil **Node.JS** en est un très bon exemple et devient une alternative aux autres technologies côté serveur comme les Servlet en JAVA ou les technoligies JSP (JavaServer Pages), l'ASP (Active Server Pages (C#)), PHP (Hypertext Preprocessor Language), Python, Ruby ...

Aujourd'hui, JavaScript est un langage utilisé également pour créer des applications mobiles (JQuery Mobile, PhoneGAP ...), des applications Windows avec **WinJS**, manipuler des fichiers PDF et en modifiant leurs apparences et ajouter des validations de formulaires...

1.2 Historique

JavaScript a été initialement développé par Netscape et s'appelait alors LiveScript. Adopté à la fin de l'année 1995 par la firme Sun (qui a aussi développé Java), il prit alors son nom de JavaScript.

Microsoft a implémenté le langage JScript qui est semblable à JavaScript dans son navigateur Web Internet Explorer. Microsoft encourageait plutôt l'utilisation de VBScript côté client ; son navigateur possède les deux interpréteurs mais le standard universellement reconnu reste le langage JavaScript, aujourd'hui relativement homogène d'un navigateur à l'autre.

Nous en sommes à la version 1.8, la version 2.0 ayant été abandonnée. Ce qui n'est pas sans poser certains problèmes de compatibilité des pages comportant du code JavaScript, selon le navigateur utilisé.

Le noyau de JavaScript est défini par le standard ECMA*-262, approuvé par l'ISO**-16262. Depuis 2009, JavaScript est basée sur le standard ECMA-262 version 5.

(* European Computer Manufacturer's Association (www.ecma-international.org))

(**International Organization for Standardization)

Tout d'abord un peu oublié pendant la bataille entre le HTML et le XHTML, avec l'arrivée du HTML 5, l'avenir de JavaScript est bien relancé.

Aujourd'hui la majorité des navigateurs sont compatibles ECMA Edition 5.

Cependant, les navigateurs continuent à ajouter des fonctionnalités non standard.

Html 5 est bien parti pour normaliser le tout...

1.3 Javascript dans le développement moderne

Dans le développement Web moderne, on attend à ce qu'un site fonctionne correctement quel que soit l'équipement utilisé (PC, Tablette, Téléphone, système d'exploitation, navigateur...) Même sur des anciens navigateurs et même si JavaScript est désactivé.

Il est fortement recommandé d'externaliser le code JavaScript dans des fichiers source distincts des pages Web ; le code JavaScript devient alors non intrusif car il est séparé du code Html.

On utilise l'enrichissement progressif, en séparant les couches :

1. La sémantique ou structure (Html)
2. La présentation (CSS)
3. Le comportement (JavaScript)

Ainsi, la page est toujours fonctionnelle même si JavaScript et CSS sont désactivés par l'utilisateur.

2. Les bases du langage Javascript

2.1 La balise <script>

Dans la logique du langage Html, il faut signaler au navigateur par une balise, que ce qui suit est du code JavaScript (et non du code Html ou VBScript par exemple). C'est le rôle de la balise double <*script*>.

Depuis la version HTML 4.01 et le XHTML (Extensible HyperText Markup Language), cette balise prend un attribut, le type MIME (Multipurpose Internet Mail Extension. Il permet de définir un format de données), pour indiquer le format du code qu'elle contient :

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

Auparavant, les spécifications utilisaient l'attribut "*language*" :

```
<script language="JavaScript">
```

En utilisant du code HTML 5, on n'est plus obligé de spécifier le type MIME.

```
<script> …code JavaScript … </script>
```

2.2 La syntaxe de Javascript

JavaScript est sensible à la casse. Ainsi pour afficher une boîte de dialogue d'alerte, il faudra écrire *alert*() et non *Alert*().

Pour l'écriture des instructions JavaScript, on utilisera l'alphabet ASCII classique (à 128 caractères) comme en Html. Les caractères accentués comme é ou à ne peuvent être employés que dans les chaînes de caractères.

Pour déclarer une chaîne de caractères, les guillemets " et l'apostrophe ' peuvent être utilisés à condition de ne pas les mélanger. Si vous souhaitez utiliser des guillemets dans vos chaînes de caractères, tapez \" ou \' pour les différencier.

JavaScript ignore les espaces, les tabulations et les sauts de lignes.

Les commentaires en JavaScript suivent les conventions utilisées en C et C++ :

```
// Commentaire sur une seule ligne

/* Commentaire
   sur
   plusieurs
   lignes */
```

Les points-virgules terminent les instructions. Cependant, les interpréteurs JavaScript acceptent les instructions isolées sans terminaison en point-virgule.

Pensez toujours à finir vos instructions par un point-virgule afin d'éviter les erreurs d'exécution !

Réglementation

Si vous utilisez des pages en HTML 4 ou XHTML et que vous voudriez valider votre page dans le validateur du W3C (World Wide Web Consortium), vous devrez encapsuler votre

code JavaScript entre des balises de commentaires Html. Autrement, si votre code comprend des marqueurs <, >, &, le validateur les prendra pour des balises et ne validera pas votre page :

```
<script type="text/JavaScript" language="JavaScript">
   <!-- Masquer le script pour les anciens navigateurs
        code JavaScript
   // Cesser de masquer le script -->
</script>
```

2.3 Où inclure le code en Javascript ?

Il existe plusieurs moyens d'ajouter du code JavaScript à votre page :

- Entre les balises *<script> ... </script>* ; elles-mêmes positionnées n'importe où entre les balises *<head>...</head>* et *<body>...</body>*.
- Directement dans les balises d'éléments Html via les attributs de gestion des événements :

```
<input type="button" onClick="alert('Hello');" /> <!-- Intrusif -->
```

- En appelant un fichier JavaScript externe :

```
<script src="maBibli.js"> // Pas de code JavaScript ici ! </script>
```

Une bonne pratique aujourd'hui consiste à ce que le comportement de JavaScript soit « non intrusif ».

Pour la réutilisabilité du code, il est conseillé d'implémenter votre code JavaScript dans des fichiers externes et de les appeler ensuite dans les pages Web grâce à la balise Html *<script src=...>*.

De plus, il est préférable d'appeler votre code JavaScript à la toute fin de votre code Html, juste avant la balise fermante *</body>*, pour 2 raisons :

- Le navigateur traite votre page Html de haut en bas (y compris vos ajouts en JavaScript). Si le code JavaScript est lourd à charger, votre page risque d'être longue à charger également.
- JavaScript est très utilisé pour modifier les éléments du code Html (le DOM). Votre code JavaScript ne pourra atteindre les éléments Html de la page qu'une fois ces derniers chargés. Par conséquent, il ne doit être interprété qu'à la fin du chargement de la page.

2.4 Les variables

Les variables peuvent se déclarer n'importe où dans le code et de deux façons :

- Soit de façon explicite avec le mot clé var (pour variant).

  Par exemple :

```
var numAdherent = 1; // Déclaration et initialisation d'une variable
// Déclaration de plusieurs variables et initialisation des 2 premières
var nomAdherent = "Darme",
    prenomAdherent = "Jean",
    age;
```

- Soit de façon implicite. On écrit directement le nom de la variable suivi de la valeur qu'on lui attribue et JavaScript s'en accommode.

  Par exemple :

```
// Déclaration et initialisation de 2 variables
numAdherent = 2;
prenomAdherent = "Luc";
age; // va provoquer une erreur de déclaration de variable
```

Attention ! Malgré cette apparente facilité, la façon dont on déclare la variable aura une grande importance pour la "visibilité" (la "portée") de la variable dans le programme JavaScript.

Les variables sont typées dynamiquement. Selon la valeur qu'on lui affecte, la variable prendra le type correspondant.

JavaScript utilise 5 types de données : les nombres, les chaînes de caractères, les booléens, les objets et le mot clé undefined pour les variables non initialisée.

Exemple :

```
var maVariable;            // son type est undefined
maVariable = 324;          // son type devient number (base 10)
maVariable = 0324;         // son type reste number (base 8)
maVariable = 0x324;        // son type reste number (base 16)
maVariable = "Bonjour";    // son type devient string
maVariable = true;         // son type devient boolean
maVariable = new Array();  // son type devient Object ou Array
```

Les tableaux de variables *Array* peuvent posséder une ou plusieurs dimensions et leur taille peut s'auto adapter à leur contenu courant (voir les compléments en fin de document).

Exemple :

```
// crée un tableau à 1 dimension pouvant contenir 10 valeurs
tabScore = new Array(10) ;
// affecte le 2° poste de la valeur 15
tabScore[1] = 15 ;
```

À noter qu'une donnée de type *string* est automatiquement dotée de 'méthodes' simplifiant ses manipulations :

Méthodes	Descriptions
`.charAt()`	Retourne le caractère selon son indice (base zéro)
`.indexOf() , lastIndexOf()`	Retourne l'indice d'un caractère à partir du début ou de la fin de la chaîne
`.trim()`	Supprime les espaces inutiles en début et fin de chaîne
`.toUpperCase(), .toLowerCase()`	Convertit en MAJUSCULES, en minuscules
`.substr(), .substring()`	Extrait une sous-chaîne de caractères
`.big(), .italics()`	Transforme en plus grand, en italique (comme le fait HTML)

Le nom d'une variable (ou d'une fonction) se nomme identificateur.

Voici la liste des mots clés réservés à ne pas utiliser pour nommer vos variables JavaScript :

« *break, case, catch, class, const, continue, debugger, default, delete, do, else, enum, export, extends, false, finally, for, function, of, import, in, instanceof, new, null, return, switch, super, this, throw, true, try, typeof, var, void, while, with* ».

On peut vérifier le type en cours d'une variable avec la fonction *typeof()* ou l'attribut *constructor* (voir exercice 3.5).

Conseil

Pour la clarté de votre script, on ne peut que vous conseiller :

- de déclarer toutes les variables d'un bloc au début de bloc,
- d'utiliser à chaque fois le mot-clé varvarvar pour déclarer une variable,
- de lui assigner un nom correspondant à son contenu,
- et de ne pas faire varier son type dans un même script.

2.5 Les opérateurs

Voici la liste des opérateurs les plus courants mis à disposition par JavaScript :

Opérateurs	Description
+, -, *, /, %, =	Opérateurs arithmétiques de base
==, ===, <, >, <=, >=, !=, !==	Opérateurs de comparaison
+=, -=, *=, /=, %=	Opérateurs associatifs
&&, ||, !	Opérateurs logiques (AND, OR et NOT)
X++, x--	Opérateurs d'incrémentation et de décrémentation
+	Concaténation de chaînes de caractères

L'opérateur d'identité === contrôle également le même typage des 2 valeurs.

Exemple :

```
console.log(42 == "42");   // retourne true
console.log(0 == false);   // retourne true
console.log(42 === "42");  // retourne false
console.log(0 === false);  // retourne false
```

Pour être plus précis dans vos comparaisons, préférez l'opérateur d'identité ===.

2.6 Les conditions

À un moment ou à un autre de la programmation, on aura besoin de tester une condition. Ce qui permettra d'exécuter ou non une série d'instructions.

« Si maman si » ou l'expression IF :

```
if (condition vraie) une seule instruction;

if (condition vraie)  {
      une;
      ou plusieurs instructions;
}

if (condition vraie) {
      instructions1;
} else if {
      instructions2;
} else {
      Instructions3;
}
```

Moins lisibles, les expressions ternaires retournent une valeur.

```
(test condition) ? valeur si vrai : valeur si faux;
```

Exemple :

```
var genre = "f";
alert((genre == "h")? "Monsieur" : "Madame");
```

Et si ma condition de test propose plusieurs sorties... je **switch** :

```
var animal = "oiseau";
switch(animal) {
      case "chien": …
      case "oiseau" : …
      case "poisson": …
      case "vache" :
            console.log("C'est un vertébré");
            break;
      case "mouche" :…
      default :
            console.log("C'est un invertébré");
}
```

Pour tous les tests conditionnels, ordonnez les tests du plus probable au moins probable.

2.7 Les répétitions

« Je t'ai répété 100 fois ... », la boucle FOR :

```
for (var i = 0 ; i < 100 ; i++ ) {
      console.log("Préfère la boucle FOR si tu connais le nombre !");
}
```

« Tant que tu ne comprends pas, je recommencerai... », les boucles WHILE :

```
var i;
while (!(i)) {
      i = confirm("As-tu compris ?");
}

do {    // l'instruction suivante sera exécutée au moins 1 fois !
      i = prompt("laisse vide ou annule");
} while (i);
```

L'instruction *break* permet d'interrompre prématurément une boucle *for* ou *while* (mais elle ne devrait jamais être utilisée en bonne programmation structurée).

L'instruction *continue* permet de sauter une instruction dans une boucle *for* ou *while* et de passer à l'itération suivante de la boucle (sans sortir de celle-ci comme le fait *break*, mais elle ne devrait jamais être utilisée elle aussi).

Exemple :

```
var compt=0;
while (compt<10) {
      compt++;
      if (compt == 3) continue;
      if (compt == 6) break;
      console.log("ligne : " + compt);
}
```

3. Les outils de développement

Pour apprendre et exploiter le langage JavaScript, il vous faut au minimum :

- un navigateur qui reconnaît et interprète le langage JavaScript,
- et un éditeur de texte (NotePad++, SublimeText ...).

Pour faciliter le débogage, vous pouvez aussi utiliser des outils plus spécialisés, comme :

- une interface de développement intégrée (Eclipse, NetBeans ...),
- ou des outils de mise au point intégrés au(x) navigateur(s).

3.1 Le débogage

Au chargement du script par le navigateur, JavaScript passe en revue les différentes erreurs de syntaxe qui pourraient empêcher le bon déroulement du script. En cas d'anomalie, l'interpréteur ne va pas plus loin et le navigateur se comporte comme si la page Html ne contenait pas de script JavaScript.

Si JavaScript décèle une erreur de syntaxe, c'est tout le chargement du bloc qui est annulé !

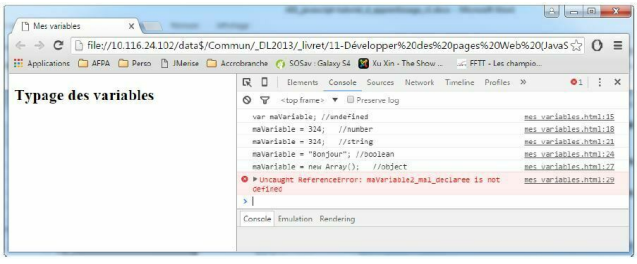
L'utilisation de la fonction *alert()* aide très souvent le développeur à déceler une erreur en affichant des boites de dialogue supplémentaires permettant de tracer le déroulement d'un script.

Un autre bon moyen de contrôler un ensemble de valeurs est de les afficher dans la console du navigateur au moyen de la fonction *console.log()*.

La console du navigateur est accessible en lançant l'outil de développement du navigateur.

C'est également dans cette console que l'on pourra récupérer les informations sur les erreurs de syntaxe.

Exemple sous Chrome :



Tous les navigateurs modernes possèdent des outils permettant d'explorer la structure de la page, les styles CSS et d'afficher des informations sur le code JavaScript, les valeurs courantes des variables, ainsi que les éventuelles erreurs de syntaxe.

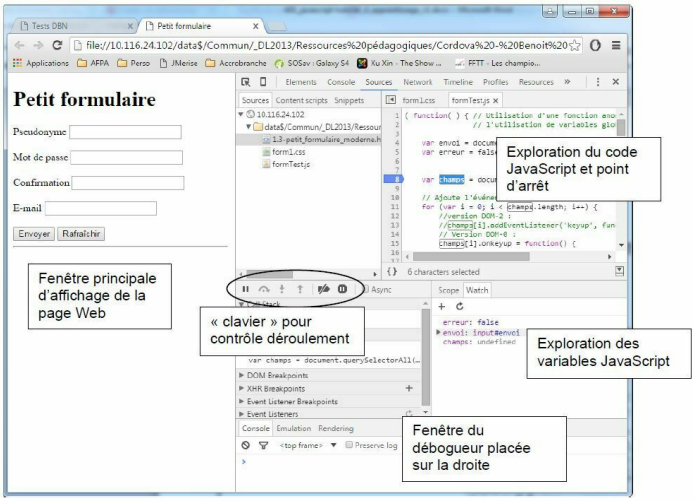
Chaque navigateur utilise des outils propriétaires. Leurs utilisations et les informations retournées sont différentes d'un navigateur à l'autre.

Ces outils permettent notamment :

- d'inspecter les éléments HTML et CSS et de modifier « à la volée » leurs valeurs ;
- de déboguer le code JavaScript avec des points d'arrêt et des exécutions pas à pas ;
- de contrôler les requêtes HTTP, notamment pour AJAX.

3.2 Les outils de Google Chrome

On peut appeler la barre de développement de Chrome avec le raccourci **F12** ou le raccourci **CTRL+MAJ+I** ou via le menu « **paramètres/plus d'outils/outils de développement** »

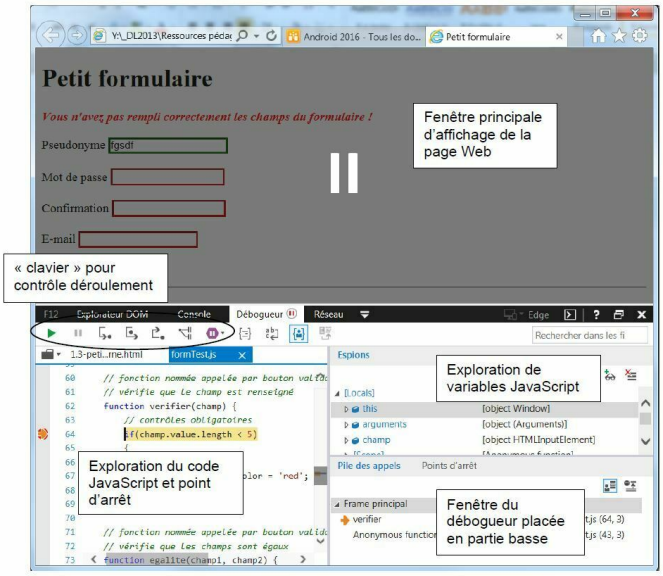


Pour avancer pas à pas après une pause sur un point d'arrêt, utiliser les touches de fonction F10 et F11.

3.3 Les outils de Microsoft Internet Explorer et Edge

Les anciennes versions d'Internet Explorer affichaient une barre d'état dans laquelle apparaissait un icône d'alerte si la page comportait des erreurs. 

On peut appeler la barre de développement d'Internet Explorer avec le raccourci **F12** ou via le menu « **paramètres/outils de développement** ».

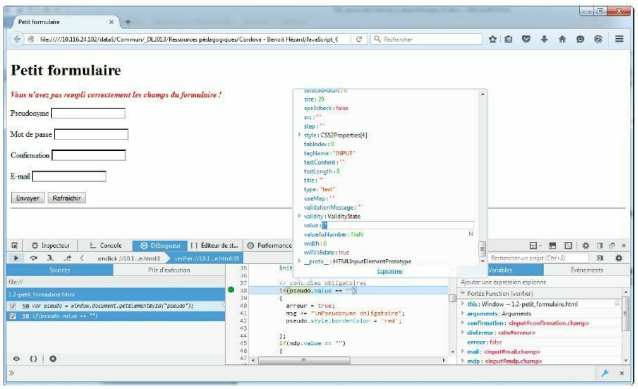


Pour avancer pas à pas après une pause sur un point d'arrêt, utiliser les touches de fonction **F10** et **F11** aussi bien avec Internet Explorer, Edge ou Chrome.

Référence : https://msdn.microsoft.com/fr-fr/library/bg182326(v=vs.85).aspx

3.4 Les outils de Mozilla FireFox

On peut appeler la barre de développement de Mozilla FireFox avec le raccourci **F12** ou via le menu « **paramètres/Développement/outils de développement** »



Les outils de développement intégrés dans les dernières versions du navigateur Mozilla FireFox sont très performants.

Le navigateur Mozilla FireFox propose aussi des plug-ins très intéressants comme :

- **Web Developer** : il ajoute une barre d'outils qui affiche en direct la validité du code HTML/CSS et JavaScript. Il peut désactiver « à la volée » le code CSS, le JavaScript, les images, gérer les cookies, les formulaires, lancer la console d'erreur, et bien plus...
- **Firebug** : extension très utilisée avant l'arrivée du dernier module intégré de développement de Mozilla.

4. Les fonctions

Une fonction a pour but principal de définir un bloc d'instructions à un seul endroit du script, réutilisable et exécutable autant que nécessaire par simples appels depuis le script principal (ou depuis une autre fonction). Les notions sous-jacentes sont celles de 'sous-programme' et de 'modularisation du code'.

Les procédures n'existent pas en tant que telles en JavaScript ; une fonction qui ne retourne rien est donc une procédure. JavaScript utilise l'instruction return pour retourner une valeur et redonner la main au programme appelant.

Si l'instruction *return* n'est pas spécifiée, la fonction retournera *undefined.*

Le langage JavaScript permet de définir ses propres fonctions (mot-clé *function*) et possède des fonctions natives très utiles.

Exemple :

Instruction	Description
eval(string)	Méthode qui évalue la chaîne passée en argument comme un script JavaScript. Exemple : *eval("x=10;y=20;console.log(x*y);");*
isNaN(x)	Méthode qui retourne true si le paramètre x n'est pas un nombre.
parseFloat(string)	Méthode qui convertit la chaîne en nombre à virgule flottante.
parseInt(string)	Méthode qui convertit la chaîne en entier.

Attention

Les fonctions *parseFloat()* et *parseInt()* acceptent une chaîne de caractères. Si la chaîne commence par un nombre, la fonction le renverra. Si le premier caractère n'est pas un nombre, la fonction renverra "*NaN*" (Not A Number).

Orientation objet oblige, de nombreuses fonctions standards sont livrées sous formes de 'méthodes d'instances' ou 'méthodes statiques'.

Exemple :

- *Math.sqrt()* est une méthode statique de l'objet prédéfini *Math* ; elle calcule la racine carrée du nombre fourni en paramètre.
- *"Centre Afpa".indexOf("C")* retourne la valeur *0*, soit l'indice du caractère '*C*' dans la chaîne qui est implicitement un objet *string* doté de nombreuses méthodes.

Il existe plusieurs sortent de fonctions :

- les instructions de fonctions (les plus courantes),
- les expressions de fonctions,
- les fonctions anonymes (qui servent à isoler une partie du code),
- les fonctions « Callback »,
- les fonctions auto-exécutables,
- les fonctions issues d'un objet (les méthodes et les constructeurs).

4.1 Les instructions de fonctions

À l'origine, toutes les fonctions JavaScript devaient être déclarées et définies dans la partie *<head>* de la page HTML selon la syntaxe suivante :

```
function nomFonction(liste_paramètres_reçus){instructions_à_exécuter};
```

Exemples :

```
// calcul la surface et la retourne
function calculeSurface (largeur, hauteur) {
     return largeur * hauteur;
};

// affiche un message constant
function coucou() { alert("coucou !"); }
```

L'appel d'une fonction se fait le plus simplement du monde par le nom de la fonction suivi des parenthèses qui incluent les éventuels paramètres à passer à la fonction :

```
Var s = calculeSurface(8, 4); // s est affecté par la valeur 32
coucou(); // affiche "coucou !" dans une boîte de dialogue
```

En JavaScript les instructions de fonctions sont automatiquement remontées en haut du bloc de script. Il est donc possible d'appeler ces fonctions avant de les avoir déclarées !

4.2 La porte des variables

Avec les fonctions, le bon usage des variables locales et globales prend toute son importance.

Une variable déclarée dans une fonction par le mot clé var aura une portée limitée à cette seule fonction. C'est une variable locale accessible uniquement par cette fonction.

Les paramètres éventuels de la fonction constituent aussi des variables locales.

En revanche, toute variable déclarée sans le mot clé **var** aura une portée globale. Elle sera une propriété de l'objet prédéfini *window*.

Les variables déclarées à l'extérieur de la fonction ou globales sont bien entendu visibles elles aussi dans la fonction.

Exemple :

```
var nomExterne = "Hein "; // var 'locale' pour le script, donc globale

function portee(nom) {
     var prenom = "Terieur "; // var locale
     nomGlobale = "Halle "; // var globale
     console.log(window.nomGlobale + nom + prenom);
     console.log(nomGlobale + nomExterne + prenom);
}

portee("Ex ");
console.log(prenom);// provoque une erreur
```

4.3 Les paramètres des fonctions

Contrairement aux langages fortement typés comme Java ou C#, la **signature** ou **liste des paramètres** d'une fonction JavaScript est assez 'souple' et n'impose pas le respect strict des paramètres attendus (ce qui permet de reproduire la notion de 'surcharge de méthode' courante dans les langages orientés objet comme Java ou C#).

La fonction constructeur de l'objet standard Date en est un parfait exemple : *Date()* ; retourne la date et heure du jour alors que *Date("December 17, 2015 03:24:00")* ; ou encore *Date(2015,11,17)* ; retournent une date (et une heure) spécifiées. Dans ces différents cas, le nombre et le type des arguments sont très variables.

À chaque appel d'une fonction, l'objet *arguments* stocke tous les paramètres envoyés lors de l'appel de la fonction. Ainsi, en JavaScript, le développeur n'écrit qu'une seule définition de fonction en cas de variantes/surcharges mais il se doit de tester les paramètres reçus.

Exemple : la fonction ci-dessous accepte de 0 à n paramètres en fonction de la forme géométrique dont on veut calculer le périmètre.

```
function perimetre(largeur, longueur) {
     var resultat = 0;
     // test si au moins un paramètre reçu
     if (!largeur) resultat = 0;
     else if (!longueur) resultat = 4*largeur; // 1 param recu : carré
     else if (arguments.length == 2)
          resultat = (largeur + longueur)*2;   // 2 param : rectangle
     else {
          for (i in arguments) resultat += arguments[i]; // polygone
     }

     console.log(resultat);
}

perimetre();          // affiche  0
perimetre(5);         // affiche 20
perimetre(9,6);       // affiche 30
perimetre(3,7,20,8); // affiche 38
```

4.4 Les expressions de fonctions

Les expressions de fonctions passent par la création d'une variable affectée par la définition d'une fonction :

```
var getCalculeSurface = function calculeSurface(largeur, hauteur) {
     return largeur * hauteur;
};  // Attention à ne pas oublier le point-virgule de l'instruction !

var s = getCalculSurface(8, 4);
```

Dans ce cas, la fonction elle-même n'a plus besoin de nom. On dit qu'elle est anonyme :

```
var getCalculeSurface = function (largeur, hauteur) {
     return largeur * hauteur;
};

var s = getCalculSurface(8, 4);
```

On peut aussi affecter une fonction déjà créée à une variable :

```
var getCoucou = coucou;
function coucou() { alert("coucou !"); }

getCoucou();  // affiche "coucou !" dans une boîte de dialogue
```

Remarque

Attention qu'il s'agit bien de **demander l'exécution immédiate** de la fonction ; c'est pourquoi, le nom de la variable doit être suivi des habituelles parenthèses !

Les expressions de fonctions diffèrent des instructions de fonctions :

- Elles peuvent être déclarées n'importe où dans le code (dans un *if()* par exemple),

- Elles ne sont pas remontées automatiquement en haut du bloc de script (elles ne peuvent donc pas être appelées avant d'avoir été déclarées).
- Si l'expression de fonctions utilise une fonction anonyme, elle ne pourra pas être récursive.

4.5 Les fonctions anonymes

Une fonction **anonyme** ne porte pas de nom ; elle est définie 'à la volée', ce qui surcharge considérablement le code au détriment de sa lisibilité.

Les fonctions anonymes sont très utilisées dans le langage JavaScript, notamment dans la gestion des évènements, les objets, les closures, et les callback ...

En voici déjà un aperçu. JavaScript propose 4 fonctions dîtes temporelles :

- `var id = setTimeout(fct1, temps)`: créer un timer qui appelle la fonction *fct1()*après le *temps* écoulé (en milliseconde).
- `var id = setInterval(fct2, temps)`: crée un timer qui répète l'appel de la fonction *fct2()* à toutes les intervalles de *temps* (en milliseconde).
- clearTimeout (*id*) qui arrête le timer avant l'expiration du délai fixé.
- clearInterva l(id) qui arrête le timer avant le prochain appel de la fonction.

Remarque

Attention qu'il s'agit bien de **désigner** la fonction à exécuter, **sans demander son exécution** immédiate ; c'est pourquoi, le nom de la fonction ne doit pas être suivi des habituelles parenthèses !

Les fonctions appelées *fct1()* et *fct2()* peuvent même être des fonctions anonymes déclarées à l'intérieur de l'appel de la fonction de timer.

Exemple d'un minuteur :

```
var i = 9;
var decompte = setInterval(function() {
                    console.log(i--); // décompte de 10 à 1
                }, 1000); // se lance toutes les secondes

var minuteur = setTimeout(function() {
                    var d = new Date();
                    var date = d.getHours() + ":" + d.getMinutes();
                    alert("Après 10 secondes, il est " + date);
                    clearInterval(decompte); // stoppe le décompte
                }, 10000); // se lance après 10 secondes
```

4.6 Les fonctions "Callback"

Définition

Un **callback** est une fonction de retour, nommée ou anonyme, placée en paramètre d'une autre fonction qui n'est pas exécutée aussitôt mais à un moment donné, à la suite d'un lapse de temps défini, ou d'un événement précis en cas de fonctionnement asynchrone.

Attention

Pour passer des paramètres à une fonction **callback**, il va falloir utiliser une technique qui consiste à englober l'appel de la fonction de rappel dans une **fonction anonyme**. Cette fonction anonyme correspond bien à une référence à une fonction et non à une demande d'exécution (Sainte Axe, priez pour nos neurones !).

Reprenons l'exemple du minuteur précédent en ajoutant un paramètre pour définir la durée de départ :

```
var temps = 10;
setInterval(function() {
      (function(duree) {
            console.log(duree);}(temps--));
      }, 1000);
```

Ici, à chaque seconde la fonction anonyme est déclenchée et elle appelle une fonction anonyme en lui passant en paramètre `temps--`.

La forme peut sembler déroutante mais elle s'explique par les points techniques abordés précédemment...

Remarque

Attention aux pièges de syntaxe que constituent ces imbrications de fonctions au niveau des accolades, parenthèses et autres virgules ! Un faux-pas, et plus rien ne fonctionne...

Tous les traitements asynchrones comme Ajax ou l'accès aux bases de données embarquées reposent sur l'usage de fonctions callback.

4.7 Les fonctions auto exécutables

Une nouvelle notation permet de déclarer et exécuter immédiatement une fonction anonyme. La syntaxe impose simplement de faire suivre la définition de la fonction d'une paire de parenthèses afin de provoquer son exécution.

Exemple :

```
var test = function() {
      console.log('hello world');
}();
```

Attention

Pour provoquer l'exécution immédiate d'une fonction, on peut encore **l'englober dans une autre paire de parenthèses** sans oublier de **la faire suivre par sa paire de parenthèses** (Sainte Axe, priez pour nous !)

Exemple :

```
(function() {
      console.log('hello world');
}());
```

```
(function() {
      console.log('hello world');
})();
```

Cette notation a de plus l'avantage de créer un espace de travail 'privé', isolé de l'environnement du script, dans lequel on peut déclarer et utiliser des variables et fonctions invisibles de l'extérieur, ce qui peut être très utile au démarrage d'une application JavaScript.

Tout le code spécifique peut maintenant être intégré dans l'espace privé défini par ces parenthèses et il sera isolé des 'effets de bord' potentiellement générés par les nombreux autres scripts composant l'application.

```
(function() {
      console.log('hello world');

      function fct1() { … };

      function fct2() { … };

      …
})();
```

Toutes ces notions avancées sur les fonctions JavaScript font l'objet d'une étude plus approfondie lors d'une autre séance.

Fin du module

Crédits

OEUVRE COLLECTIVE DE L'AFPA

Sous le pilotage de la Direction de l'ingénierie

DATE DE MISE À JOUR

31/08/2021