

Secteur Tertiaire Informatique
Filière « Etude et développement »

Séquence « Développer des pages Web »

Programmation objet en JavaScript

Apprentissage

Mise en pratique

Evaluation

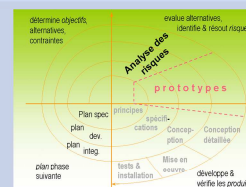
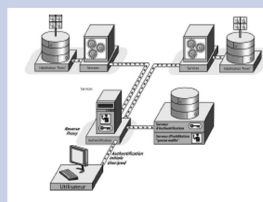


TABLE DES MATIERES

Table des matières	3
1. Formulaire de saisie	5
1.1 Design de la page	5
1.2 Programmation des contrôles de saisie façon 'historique'	6
1.3 Programmation des contrôles de saisie façon 'moderne'	6
2. Fonctions	8
2.1 Tableaux et objets	8
2.2 Classe, Objet et Prototype	8
2.2.1 Pour commencer	8
2.2.2 Manipulation d'objets instanciés	9
2.2.3 Manipulation d'objet littéral	9
2.2.4 Manipulation du prototype d'une classe	9
3. Gestion Contacts	10
3.1 L'objectif	10
3.2 Première version à base de fonctions classiques	10
3.2.1 Page HTML	10
3.2.2 Script à base de fonctions	11
3.3 Deuxième version orientée objet avec une classe	12
3.4 Troisième version orientée objet avec un objet littéral	15
3.5 Et pour aller plus loin	16

Préambule

Ces exercices permettant de mettre en pratique la programmation objet en JavaScript.

Objectifs

Les deux premiers entraînements concernent la mise en œuvre de la programmation moderne JavaScript/JQuery. Le dernier est un exercice de synthèse à réaliser de 3 manières différentes pour bien illustrer le procédural et l'orienté objet en JavaScript.

Méthodologie

Réalisez les exercices au fur et à mesure de l'étude du support d'apprentissage [A-programmation-objet-javascript.pdf](#) et reportez-vous à ce document pour trouver les syntaxes et comprendre les principes mis en œuvre.

1. FORMULAIRE DE SAISIE

A partir des extraits de code donnés dans le chapitre 1 du support d'apprentissage `A-programmation-objet-javascript.pdf`, réalisez la mini-application de 2 manières, en suivant les consignes ci-dessous.

1.1 DESIGN DE LA PAGE

- Réalisez tout d'abord la présentation HTML du formulaire sans vous préoccuper du code JavaScript (phase de maquettage) ;
- Le formulaire n'a pas forcément d'attribut `action` renseigné (il se rappellera alors lui-même automatiquement) ; prévoyez les habituels boutons `submit` et `reset` ;
- La saisie se fera systématiquement dans des zones de texte ;
- Il peut être judicieux de prévoir des styles CSS (ou même une feuille de styles séparée) pour pouvoir gérer l'affichage ou le masquage de la zone de message (`<div>` identifiée 'erreur' dans l'exemple) ; faites simple pour le rendu graphique : le but n'est pas ici de s'entraîner sur les styles CSS et la présentation standard HTML peut suffire ;

Exemple de design :

Petit formulaire

Pseudonyme

Mot de passe

Confirmation

E-mail

Exemple d'affichage en cas d'erreur :

Petit formulaire

Vous n'avez pas rempli correctement les champs du formulaire !


Pseudonyme

Mot de passe

Confirmation

E-mail

Message de la page Web

 Mots de passe différents
Email obligatoire

Mise en pratique : Programmation objet en JavaScript

Afpa © 2016 – Section Tertiaire Informatique – Filière « Etude et développement »

1.2 PROGRAMMATION DES CONTROLES DE SAISIE FAÇON 'HISTORIQUE'

Réalisez une première version de contrôle de saisie du formulaire par JavaScript (sans utiliser JQuery) de façon à vérifier que chaque zone est bien renseignée et que les 2 mots de passe sont identiques.

- Ajoutez l'appel de la fonction dans la balise HTML du bouton d'envoi ; comme ce bouton a un comportement HTML par défaut, il sera nécessaire de contrôler son action en JavaScript par le biais de l'instruction `return` et la fonction de contrôle devra retourner un booléen ;
- Placez le code de la/des fonctions dans la partie `<head>` de la page HTML (intégrées dans la balise `<script>` ou dans un fichier `.js` externe) ;
- Accédez par JavaScript aux éléments HTML selon leur propriété `id` (façon DOM-2 et non en parcourant l'arbre du document façon DOM-0) ;
- Effectuez les contrôles en série de chacune des zones de saisie au sein de la même fonction principale ; comme le script peut détecter plusieurs erreurs, il est important, à chaque erreur détectée, de préparer un message d'erreur pour pouvoir l'afficher au besoin en fin de traitement de contrôle ;
- Toute zone de saisie faisant l'objet d'une erreur sera repérée graphiquement pour l'utilisateur (bordure rouge, par exemple) ;
- Au final, en cas d'erreur détectée, le message d'erreur personnalisé sera affiché dans une boîte `alert()` et la zone d'erreur sera rendue visible par JavaScript ; de plus, l'action du bouton `submit` sera neutralisée de manière à ce que l'utilisateur reste sur le formulaire ;
- Au final, en cas de succès, l'action associée au formulaire sera déclenchée normalement ;
- Effectuez tests et mise au point en usant et abusant du débogueur du navigateur.

1.3 PROGRAMMATION DES CONTROLES DE SAISIE FAÇON 'MODERNE'

Réalisez une deuxième version de contrôle de saisie du formulaire par JavaScript en utilisant JQuery de façon à effectuer les mêmes contrôles mais de manière plus dynamique : testez le contenu à chaque frappe au clavier pour afficher la bordure en vert quand tout va bien et en rouge en cas d'erreur. Bien entendu, un contrôle global sera aussi réalisé lors de la soumission du formulaire.

- Retirez les appels de fonctions JavaScript qui sont faits directement dans les balises HTML ;
- Retirez la balise `<script>` placée dans la partie `<head>` du document ;
- Insérez une balise `<script>` en fin de partie `body` de manière à appeler 2 scripts JavaScript externes, l'un pour le framework JQuery, l'autre pour votre propre script ;
- Implémentez une fonction anonyme par la méthode JQuery `.ready()` de l'objet `document` de manière à récupérer par la fonction JQuery `$()` les références des différentes zones de saisie dans des variables afin de leur implémenter les fonctions anonymes de contrôle dynamique en cours de frappe pour changer la couleur de bordure (vert = OK, rouge = erreur - utilisez la méthode JQuery `.css()`) :
 - Testez à la volée que le deuxième mot de passe est bien identique au premier ;

Mise en pratique : Programmation objet en JavaScript

Afpa © 2016 – Section Tertiaire Informatique – Filière « Etude et développement »

- Ajoutez une même vérification à la volée pour toutes les zones de saisie : 5 caractères au minimum ;
- Implémentez par JQuery une fonction anonyme, associée à l'événement `click` du bouton `submit`, qui réalisera tous les tests en série comme pour la première version ; en cas d'erreur, il faudra neutraliser l'action HTML par défaut de ce bouton ;
- Si ce n'était pas déjà fait dans votre première version, modularisez le code JavaScript en réalisant une sous-fonction, nommée, assurant le contrôle élémentaire d'une zone de saisie passée en paramètre de manière à changer sa couleur de bordure et à faire apparaître la zone d'erreur si nécessaire ; cette sous-fonction sera appelée en série par la fonction de contrôle général associée au bouton d'envoi ;
- Terminez en écrivant une nouvelle fonction associée à l'événement `click` du bouton `reset` de manière à rétablir les couleurs de bordure des zones de saisie et faire disparaître la zone d'erreur (l'effacement des zones de saisie sera laissé aux bons soins de HTML grâce à l'action par défaut de ce bouton `reset`) ;
- Effectuez tests et mise au point en usant et abusant du débogueur du navigateur.

Exemples de rendus graphiques :

Petit formulaire

Pseudonyme

Mot de passe

Confirmation

E-mail

en cours de frappe

Petit formulaire

Vous n'avez pas rempli correctement les champs du formulaire !

Pseudonyme

Mot de passe

Confirmation

E-mail

après validation

2. FONCTIONS

Il s'agit simplement ici de mettre en œuvre quelques techniques JavaScript/JQuery au sein de scripts qui n'afficheront rien en page HTML mais produiront des messages sur la console de débogage du navigateur. Il sera donc nécessaire d'activer le débogueur pour voir le résultat des traitements.

2.1 TABLEAUX ET OBJETS

On veut tout d'abord représenter en JavaScript notre héros préféré, 'Bob Morane', de 3 manières différentes :

- Ecrivez une page HTML vide, support du script, appelant en fin de `body` le script JQuery et celui de l'exercice ;
- Implémentez par JQuery une fonction anonyme quand le document est prêt pour créer les 3 structures et en afficher le résultat en console de débogage :
 - Créez un tableau ordinaire indexé par instantiation d'un objet JavaScript `Array`, et contenant simplement 2 valeurs, le nom et le prénom ; afficher le nom dans la console ;
 - Créez un tableau associatif contenant 2 postes 'nom' et 'prenom' avec les valeurs correspondantes ; afficher le nom dans la console ;
 - Créez un objet littéral doté de 2 propriétés 'nom' et 'prenom' affectées des valeurs correspondantes ; afficher le nom dans la console ;
 - Testez en essayant les différentes variantes de notation permettant d'accéder à une valeur d'un tableau/objet JavaScript ; placez des points d'arrêt sur chaque affichage de manière à consulter finement les variables dans le débogueur ;
- Recopiez le code de cette première version et conservez l'original en commentaire ; transformez la construction globale de manière à oublier l'exécution automatique JQuery en réalisant une fonction anonyme auto-exécutante ; testez ;
- Ajoutez en fin de cette fonction auto-exécutante l'activation d'un mécanisme permettant d'afficher dans la console une horloge perpétuelle en vous basant sur la version avec *closure*.

2.2 CLASSE, OBJET ET PROTOTYPE

2.2.1 Pour commencer

Ecrivez un script permettant de définir une classe JavaScript 'Employe' comportant :

- Une propriété 'nom'
- Une propriété 'prenom'
- Une propriété 'branche'
- Un attribut privé 'salaire'
- Un constructeur d'initialisation qui affecte ces 4 informations
- Un getter sur le 'salaire' (`getSalaire()`)
- Une méthode calculant le salaire annuel (`salaireAnnuel()`)

Mise en pratique : Programmation objet en JavaScript

- Un setter sur le salaire (`setSalaire()`) qui contrôle que le nouveau salaire est bien supérieur au salaire actuel ; en cas d'erreur, il s'agit de lever une exception JavaScript à l'aide de l'instruction `throw` ;

Ecrivez un script exploitant cette Classe ; ce script sera enrichi au fur et à mesure de l'évolution de cet exercice. Dans un premier temps, écrivez simplement les instructions JavaScript permettant d'instancier deux objets `Employe`, par exemple `Pierre` et `Paul` en vous inspirant des exemples données dans le support d'apprentissage. Les messages de trace seront simplement affichés en console du débogueur.

Ecrivez une page HTML de support, qui n'affiche rien de particulier mais qui charge les 2 scripts.

Testez en navigateur, débogueur activé pour vérifier votre construction et vos syntaxes ; placez des points d'arrêt pour explorer vos objets.

2.2.2 Manipulation d'objets instanciés

Enrichissez le script permettant d'exploiter votre classe `Employe`.

Les deux objets '`Pierre`' et '`Paul`' sont déjà instanciés. Affichez par de simples `console.log()` leurs nom, prenom, branche et salaire ; pour cette dernière information, appelez aussi bien l'attribut que le *getter*. Testez

Ajoutez à l'objet `Pierre` une propriété '`chef`' dont la valeur sera la référence de l'objet `Paul` ; affichez le nom du chef de `Pierre` et le nom du chef de `Paul`. Testez.

Ajoutez une modification du salaire de `Pierre`, en proposant d'abord un salaire inférieur au salaire initial puis un salaire supérieur. Protégez le code en captant les exceptions éventuellement levées. Testez

2.2.3 Manipulation d'objet littéral

Enrichissez encore le script permettant d'exploiter votre classe `Employe`.

Ajoutez la création d'un objet littéral '`Jacques`' comportant 4 propriétés similaires à celles de la classe `Employe`. Affichez les valeurs des propriétés de `Jacques`. Affectez son salaire par un salaire inférieur au salaire initial. Ajoutez à l'objet `Paul` une propriété `chef` dont la valeur est la référence de cet objet `Jacques`. Affichez le nom du chef de `Pierre` et le nom du chef du chef de `Paul`. Testez.

2.2.4 Manipulation du prototype d'une classe

Enrichissez à nouveau le script permettant d'exploiter votre classe `Employe`.

Ajoutez au prototype de la classe `Employe` une nouvelle propriété `sonChef` initialisée à la valeur `null` par défaut. Affectez à `Pierre` la valeur de cette propriété de manière à définir `Paul` comme 'son chef'. De même, affectez le chef de `Paul` comme étant `Jacques`. Affichez les noms de ces nouveaux chefs pour les objets `Pierre`, `Paul` et `Jacques`. Placez un point d'arrêt au premier affichage de manière à pouvoir bien observer les données dans le débogueur du navigateur. Testez à fond et explorez les structures de données dans le débogueur.

Mise en pratique : Programmation objet en JavaScript

Afpa © 2016 – Section Tertiaire Informatique – Filière « Etude et développement »

3. GESTION CONTACTS

3.1 L'OBJECTIF

Il s'agit maintenant de réaliser une mini-application de **gestion de contacts stockés dans une base de données Web SQL** embarquée dans le navigateur.

Pour simplifier, chaque contact n'a que 2 informations, nom et adresse email. Un petit formulaire permet de saisir de nouveaux contacts. Il est suivi d'une liste des contacts existants dans laquelle chaque personne peut être supprimée grâce à un bouton de commande.

Tout est piloté par JavaScript et se passe sur la même page HTML (une seule URL) ; le tableau listant les contacts existants aura donc un contenu dynamique.

Exemple de présentation :



Mes contacts		
Nom :	<input type="text"/>	
Mail :	<input type="text"/>	
<input type="button" value="Ajouter"/>		
Benoit	benoit.hezard@afpa.fr	<input type="button" value="Suppr."/>
Huguette	huguette.fluzin@afpa.fr	<input type="button" value="Suppr."/>
Stéphane	stephane.thomy@afpa.fr	<input type="button" value="Suppr."/>

3.2 PREMIERE VERSION A BASE DE FONCTIONS CLASSIQUES

3.2.1 Page HTML

Le code HTML définit simplement un formulaire pour saisir les 2 informations des nouveaux contacts (à l'aide de champs textes identifiés par `nom` et `mail`) ; le bouton 'Ajouter' (type `button`) permettra d'insérer par JavaScript les coordonnées de la personne dans la base de données.

Ce formulaire est suivi d'un tableau HTML à 3 colonnes, vide par défaut car c'est le script qui le peuplera ; la partie contenu du tableau (élément `<tbody>`) devra donc posséder une valeur d'attribut `id` de manière à pouvoir être manipulée par JavaScript.

Enfin, l'appel du script sera fera en toute fin de page HTML.

Testez la présentation.

3.2.2 Script à base de fonctions

Pour cette première version, on programmera les fonctionnalités à base de fonctions JavaScript indépendantes regroupées dans un script externe (`contactsClassique.js`) :

- Une variable globale (`var labase ;`) permettra à chaque fonction d'accéder à la base de données ;
- Une fonction exécutée dès le chargement de la page appelle successivement les fonctions d'ouverture de la base de données et d'implémentation des événements :

```
// démarrage :  
// -----  
// initialisations  
$(document).ready(function(){  
    initDbase(); // création/ouverture BDD  
    initEvents(); // implémentation events  
});
```

- La fonction `initEvents()` implémente (par JQuery !) l'événement *click* sur le bouton 'Ajouter': la fonction anonyme associée à l'événement appellera une fonction `insertContact()` ;
- Ecrivez la structure vide pour le moment de cette fonction `insertContact()` ;
- La fonction `initDbase()` appelle la fonction paramétrée `openContacts()` en lui passant toutes les informations utiles pour l'ouverture d'une base Web SQL ;
- La fonction `openContacts(nom, version, libelle, taille)` initialise la variable globale par le résultat de l'ouverture de la base de données selon les paramètres reçus ; comme la fonction Web SQL `openDatabase()` n'a pas de fonction callback, il faut ensuite appeler la première fonction d'exploitation de la base de données, `testContacts()` :

```
// ouvrir/créer la BDD  
function openContacts(nom, version, libelle, taille){  
    // dbase = base WebSQL contacts  
    labase = window.openDatabase(nom, version, libelle, taille);  
    // pas de fonction callback automatique...  
    testContacts(); // ... donc, appel explicite  
};
```

- La fonction `testContacts()` crée au besoin la table `contacts` à l'aide d'une transaction Web SQL et appelle automatiquement une fonction de callback en cas de succès (`lireContacts`) ou une autre en cas d'échec (`onError`) ; la table est constituée des colonnes `name` et `email`, toutes deux de type texte :

```
// créer au besoin la table contacts  
function testContacts(){  
    // libelle requete sql  
    var sql;  
    labase.transaction(function(tx){  
        sql = "create table if not exists contacts
```

Mise en pratique : Programmation objet en JavaScript

```

        (name text, email text)";

        tx.executeSql(sql,[], lireContacts, onError);
        // callback succès ==> rechercher et lister
    })
};

```

- La fonction `lireContacts(tx, rs)` lance sur le même modèle une autre requête SQL à travers une transaction Web SQL de manière à récupérer tous les contacts (préciser l'ordre de tri) et appelle automatiquement une fonction callback en cas de succès (`listContacts`) et une autre en cas d'échec (`onError`) ;
- La fonction `listContacts(tx, rs)` explore les lignes du `SQLResultSet` reçu en paramètre de manière à générer la ligne de code HTML assurant, pour chaque enregistrement récupéré, l'insertion d'une ligne de tableau HTML constituée de 3 cellules, une pour le nom, une pour l'email et une dernière pour un bouton type `button` sans action pour le moment ;
- Ecrivez la fonction `onError(tx, e)` qui affiche dans la console JavaScript le message de l'erreur reçue en paramètre ;
- Testez (en suivant l'exécution pas à pas dans le débogueur du navigateur) et mettez au point cette fonctionnalité de listage des contacts existants ; à ce niveau, le tableau doit rester vide mais tout doit se dérouler sans erreur ;
- Ecrivez le contenu de la fonction `insertContact()` : il s'agit de lancer une requête SQL `insert` en recherchant (par JQuery !) les valeurs des champs de saisie `nom` et `mail` ; cette fonction appellera automatiquement les fonctions de callback (par exemple `onSuccessInsertDelete` et `onError`) ;
- Ajoutez la fonction `onSuccessInsertDelete(tx, rs)` qui affiche un message témoin dans la console JavaScript, vide (par JQuery !) le contenu du tableau HTML et appelle la fonction de listage des contacts `lireContacts(tx)` ;
- Testez et mettez au point la saisie et l'affichage ;
- Affinez dans la fonction `listContacts(tx, rs)` le code HTML personnalisé de chaque bouton de manière à ce qu'un *click* de l'utilisateur déclenche par HTML la fonction de suppression `supprContact(unName)` en lui passant en paramètre la valeur du nom du contact à supprimer ;
- Ecrivez cette fonction `supprContact(unName)` qui lance une requête SQL `delete` sur la base à travers une transaction Web SQL et appelle automatiquement les fonctions de callback `onSuccessInsertDelete` et `onError` ;
- Testez bien ; c'est fini !

3.3 DEUXIEME VERSION ORIENTEE OBJET AVEC UNE CLASSE

Il s'agit maintenant de **réaliser une copie du script** de manière assurer exactement les mêmes fonctionnalités à l'aide d'un objet, instance d'une classe, qui encapsule toutes ces fonctionnalités.

- Effectuez une copie du premier script sous un autre nom (`contactsObjetClasse.js`) ; adaptez tout de suite l'appel du script dans la page HTML (conservez l'ancienne balise `<script>` en commentaire HTML de manière à pouvoir aisément passer d'une version à l'autre) ;

Mise en pratique : Programmation objet en JavaScript

- Encapsulez les fonctions `testContacts()`, `lireContacts()`, `listerContacts()`, `insertContact()`, `supprContact()`, `onSuccessInsertDelete()` et `onError()` comme membres d'une classe BDD, selon le modèle :

```
// classe gestion BDD
// -----
function BDD(nom, version, libelle, taille){
... ..
    // méthode créer au besoin la table contacts
    this.testContacts = function(laBDD){...} ;
    // méthode rechercher les contacts en BDD
    this.lireContacts = function(tx, rs){...}
... ..
}
```

- Ajoutez 2 membres de données à la classe BDD, l'un pour la requête SQL et l'autre pour la référence à la base de données :

```
// classe gestion BDD
// -----
function BDD(nom, version, libelle, taille){
    // prop libelle requete sql
    this.sql="";
    // prop db = base WebSQL
    // ouvrir la BDD
    this.db = window.openDatabase(nom, version, libelle, taille);
}
```

NB : en JavaScript, il s'agit en fait de la définition du *constructeur* de la classe

- Adaptez la fonction `initDbase()` de manière à ce qu'elle instancie un objet BDD en lui passant les paramètres nécessaires, avant d'appeler sa méthode `testContacts()` :

```
// instantiation et ouverture BDD WebSQL
function initDbase(){
    labase = new BDD("Webcontact", "1.0", "mes contacts", 5*1024*1024);
    labase.testContacts(labase);
}
```

- Adaptez la fonction `testContacts(laBDD)` qui reçoit maintenant en paramètre la référence de l'objet instancié par la fonction `initDbase()` ; l'objet `laBDD` étant une instance de la classe, il faut préciser maintenant sa propriété `db` dans l'appel de la méthode `.transaction()` ; de plus, la variable de requête SQL étant maintenant un membre de la classe, tout comme les fonctions de callback, elles doivent être elles-aussi être préfixées :

```
// méthode créer au besoin la table contacts
this.testContacts = function(laBDD){
    laBDD.db.transaction(function(tx){
        laBDD.sql = "create table if not exists contacts(name text, email text)";
        tx.executeSql(laBDD.sql,[], laBDD.lireContacts, laBDD.onError);
    });
}
```

Mise en pratique : Programmation objet en JavaScript

```

        // callback succès ==> rechercher et lister
    })
};

```

- Adaptez la méthode `lireContacts(tx, rs)` sur le même modèle ; attention ici, on est dans le cas d'un appel callback automatique et la signature de la méthode est donc imposée ; il sera impossible de lui passer la référence à l'objet de base de données et, comme il s'agit d'un traitement asynchrone, `this` référencerait l'objet JavaScript window et non cet objet de base de données (essayez donc !) ; on est bien dans un cas de besoin de *closure* pour mémoriser cet objet de base de données tant qu'on le connaît et le passer en paramètre à la fonction asynchrone :

```

// méthode rechercher les contacts en BDD
this.lireContacts = function(tx, rs){
    // mémo ref à la BDD car la transaction sera asynchrone et this
    // référencera l'objet window

    // ici on est sur une fonction callback à la structure imposée ==>
    // reprendre la var globale du script

    var self = labase;

    self.db.transaction(function(tx){
        self.sql = "select * from contacts order by name";
        tx.executeSql(self.sql,[],self.listerContacts,self.onError);

        // callback succès ==> lister
    })
};

```

- Cette construction sera à reproduire pour toutes les fonctions asynchrones ; on peut même généraliser ce principe dans toute la classe en adaptant à nouveau le code de la fonction précédente de manière à conserver une certaine cohérence dans le code :

```

// méthode créer au besoin la table contacts
this.testContacts = function(laBDD){
    // mémo ref à la BDD car la transaction sera asynchrone et this
    // référencera l'objet window

    // ici on est dans un appel direct depuis la méthode précédente qui a
    // passé en paramètre la ref à cet objet

    var self = laBDD;

    self.db.transaction(function(tx){
        self.sql = "create table if not exists contacts(name text, email
text)";
        tx.executeSql(self.sql,[],self.lireContacts,self.onError);

        // callback succès ==> rechercher et lister
    })
};

```

- Adaptez les autres méthodes de la classe en suivant ces principes ; adaptez aussi l'appel de la fonction de suppression dans le code HTML généré par la méthode `listerContacts` ;

Mise en pratique : Programmation objet en JavaScript

Afpa © 2016 – Section Tertiaire Informatique – Filière « Etude et développement »

- Testez à fond en abusant du débogueur du navigateur ; l'application doit fonctionner à l'identique pour l'utilisateur ; seule la construction du code a changé.

Cette version orientée objet classique permettrait d'instancier plusieurs objets BDD dans la même application, ce qui n'est pas le besoin ici. Le seul avantage de cette version dans notre cas est que cette classe est maintenant quasi indépendante du script qui l'utilise et elle pourrait être externalisée dans un script séparé pour une réutilisation dans un autre contexte (essayez donc !).

3.4 TROISIEME VERSION ORIENTEE OBJET AVEC UN OBJET LITTERAL

Encore un effort ! Vous allez maintenant réaliser une autre version à fonctionnalités identiques mais construite avec un objet littéral JavaScript. Vous avez déjà bien préparé le travail avec la version objet précédente.

- Effectuez une nouvelle copie de votre script (contactsObjetJSON.js) et adaptez le code de la page HTML en conséquence ;
- Transformez la définition de la classe BDD classe en une **définition d'objet littéral** (attention à la syntaxe !) :

```
// objet littéral gestion BDD
// -----
// (c'est un objet non paramétrable)
```

```
var BDD = {
```

```
    // prop libelle requete sql
```

```
    sql : "",
```

```
    // prop db = base WebSQL
```

```
    db : null,
```

```
    // méthode ouvrir la BDD
```

```
    openBDD : function(){
```

- Ecrivez la fonction **openBDD()** qui ne peut plus être paramétrable ni réutilisable (ce qui n'est pas gênant car le but d'un objet littéral est bien d'être spécialisé pour servir dans un contexte particulier) :

```
    // méthode ouvrir la BDD
```

```
    openBDD : function(){
```

```
        this.db = window.openDatabase("Webcontact","1.0","mes contacts",5*1024*1024);
```

```
        // pas de callback...
```

```
        this.testContacts(this);
```

```
        //...donc appel explicite avec passation ref à cet objet
```

```
    }
```

- Adaptez la méthode initDbase() de manière à appeler explicitement cette méthode d'ouverture de la base de données :

```
// création/ouverture BDD WebSQL
```

```
function initDbase(){
```

Mise en pratique : Programmation objet en JavaScript

```

labase = BDD; // conservé pour cohérence avec les autres scripts
labase.openBDD();
}

```

Et voilà, l'essentiel y est ! Vérifiez bien vos syntaxes et testez à fond en abusant toujours du débogueur et en comparant les structures de données entre un objet littéral et un objet instance de classe.

Cette version à base d'objet littéral est assez typique de ce qu'on retrouve couramment dans le monde JavaScript quand on souhaite encapsuler des données et méthodes sans pour autant nécessiter de multiples instances.

3.5 ET POUR ALLER PLUS LOIN

On peut encore s'entraîner aux syntaxes JavaScript en réalisant par copies quelques autres variantes :

- Script à base de classe JavaScript déclarée comme un 'pointeur de fonction'

```

// fonction gestion BDD
// -----
// (c'est une closure)
var dbaseContacts = function(nom, version, libelle, taille){
    // prop libelle requete sql
    this.sql="";
    // prop dbase = base WebSQL
    this.db = null;

    // méthode ouvrir la BDD
    this.openBDD = function(){...} ... .. }

```

- Remplacement de la fonction de démarrage JQuery par une fonction en autoexécution placée en fin de script ;
- Définition des membres dans le prototype de la classe...

CREDITS

ŒUVRE COLLECTIVE DE L'AFPA

Sous le pilotage de la DIIP et du centre d'ingénierie sectoriel Tertiaire-Services

Equipe de conception (IF, formateur, mediatiseur)

B. Hézard – formateur

Stéphane Thomy - formateur

Ch. Perrachon – Ingénieure de formation>

Date de mise à jour : 27/4/16

Reproduction interdite

Article L 122-4 du code de la propriété intellectuelle.

« Toute représentation ou reproduction intégrale ou partielle faite sans le consentement de l'auteur ou de ses ayants droits ou ayants cause est illicite. Il en est de même pour la traduction, l'adaptation ou la reproduction par un art ou un procédé quelconque. »

Mise en pratique : Programmation objet en JavaScript

Afpa © 2016 – Section Tertiaire Informatique – Filière « Etude et développement »