KIT DE SURVIE JAVASCRIPT - TD #1

Pierre BIASUTTI
IUT Informatique, Licence Pro

1 Rappels de Javascript

Variables En JS, la déclaration de variable se fait à l'aide du préfixe ***var***.

Exemple

```
var a;
var b = 1.41;
var c = "texte";
```

Conditions Les conditions JS se font au travers des mots-clés ***if, else if, else***.

Exemple

```
if(a == 1){
        // Condition 1
}
else if(b < 12){
        // Condition 2
}
else{
        // Condition 3
}
```

Notez que l'usage des accolades n'est nécessaire que lorsque le bloc de la condition contient plus d'une instruction. Dans le cas où l'on souhaite créer beaucoup de conditions par rapport à la valeur d'une variable, on peut utiliser les mots-clés ***switch, case, break***.

Fonctions Les fonctions JS se déclarent à l'aide du mot-clé ***function***. Une valeur peut être retournée en utilisant le mot-clé ***return***

Exemple

```
function divideByX(n, x){
        return n / x;
}
```

Les fonctions peuvent aussi être déclarées comme des variables. Ainsi, une fonction peut être passé comme argument d'une autre fonction.

Exemple

```
var divideByX = function(n, x){ return n / x; };
```

Afficher la valeur d'un objet, débugger Le Javascript permet d'afficher le contenu d'une variable ou d'un objet grâce à la fonction ***console.log(var)***. Le résultat de cette fonction est lisible dans la console JS, disponible pour la plupart des navigateurs dans les Outils.

Exemple

```
var a;
console.log(a); // Affiche "undefined" dans la console
var b = "javascript";
console.log(b); // Affiche "javascript" dans la console
```

2 Objets en Javascript

2.1 Objet littéral et constructeur

Object littéral En Javascript, un objet peut se construire de manière littérale comme un tableau associatif.

Exemple

```
var personne = {
        prenom: "Elsa",
        nom:    "Dupont",
        age:    21,
        presenter: function(){
                console.log("Je suis " + this.prenom + " " + this.nom
                                + " et j'ai " + this.age + "ans");
        };
};

// On appelle la fonction ``presenter''
personne.presenter();
```

Ici l'objet est défini de manière unique, et tous ses attributs sont déjà pré-remplis.

Constructeur De manière à obtenir plus de modularité, on peut faire appel à un constructeur. Ce dernier permettra de construire l'objet en spécifiant son contenu lors de sa création.

Exemple

```
function Personne(prenom, nom, age){
        this.prenom = prenom;
        this.nom    = nom;
        this.age    = age;
        this.presenter = function(){
                console.log("Je suis " + this.prenom + " " + this.nom
                                + " et j'ai " + this.age + "ans");
        };
}

// Création d'un objet Personne
var pers = new Personne(``Elsa'', ``Dupont'', 21);
pers.presenter();
```

La souplesse du Javascript permet de modifier ou d'ajouter un attribut d'un objet de manière dynamique. On peut par exemple rajouter une nouvelle fonction membre a l'objet contenu dans la variable **pers**.

Exemple

```
pers.direPrenom = function(){ console.log(this.prenom); };
```

Exercice 01. Ecrire un objet littéral "Etudiant" contenant les informations suivantes: Nom, Prénom, Numéro étudiant, Date de naissance (au format représenté sous la forme d'un tableau de 3 entiers Jour-Mois-Année).

Exercice 02. Ecrire le constructeur d'un objet Etudiant possédant les mêmes attributs que dans l'exercice 01. Le constructeur doit prendre en argument Nom, Prénom, Num. étudiant et Date de Naissance. Rajouter a ce constructeur les fonctions suivantes:

- presenter: affiche une courte phrase résumant les différentes informations de l'étudiant dans la console
- age: calcul l'age de l'étudiant en fonction de sa date de naissance

Enfin, créer deux objet Etudiant correspondant à deux Etudiants différents.

2.2 Prototypes

Prototype Le Javascript n'a pas de concept de classe. Cependant, les prototypes peuvent être utiles pour définir des fonctions de base à des objets.

Exemple

```
var Personne = {
        nom: "",
        prenom: "",
        dateNaissance: [1, 1, 1977],

        init: function(nom, prenom, dateNaissance){
                this.nom           = nom;
                this.prenom        = prenom;
                this.dateNaissance = dateNaissance;
        },

        decrire: function(){ console.log( ... ); }
};

var pers = Object.create(Personne);
pers.nom = "Dupont";
pers.prenom = "Michel";
pers.dateNaissance = [12, 5, 1971];

// Ou

pers.init(''Dupont'', ''Michel'', [12, 5, 1971]);
```

Ici, la fonction membre create de l'objet générique **Object** permet de créer une référence a l'objet **Personne** dans la variable **pers**. De ce fait, si l'on souhaite accéder à un attribut qui n'existe pas dans **pers**, le Javascript ira automatiquement chercher cet attribut dans l'objet **Personne**. Ce concept est très utilisé pour dériver des objets depuis d'autres objets. Par exemple, en partant de notre objet **Personne**, on peut définir un objet **Etudiant** possédant tous les attributs d'une **Personne** ainsi que des attributs supplémentaires.

Exemple

```
var Etudiant = Object.create(Personne); // On crée la référence à l'objet Personne
Etudiant.numEtudiant = 0;
Etudiant.initEtudiant = function(nom, prenom, dateNaissance, numEtudiant){
        // On appelle la fonction de l'objet Personne
        this.init(nom, prenom, dateNaissance);

        this.numEtudiant = numEtudiant;
}

var etu = Objet.create(Etudiant);
etu.initEtudiant(''Durand'', ''Kevin'', [26, 7, 1991], 574812);
```

Exercice 03. En partant des exemples au dessus, créer un objet **Professeur** dérivé de l'objet **Personne** qui contient en plus l'adresse mail du professeur comme attribut. Créer une fonction ***initProfesseur*** permettant d'initialiser une nouvelle instance de cet objet. Enfin, modifier la fonction decrire pour qu'elle affiche l'intégralité des informations contenues dans l'objet **Professeur**. Testez votre code en créant deux professeurs.

Exercice 04. Modifier maintenant le prototype ***Personne*** en lui ajoutant un attribut **taille** et une fonction **direAge** qui calcul l'age de la personne en fonction de sa date de naissance. Ces attributs sont-ils définis pour les professeurs crées dans l'exercice 03? Pourquoi?

3 jQuery

jQuery est une bibliothèque Javascript dont l'usage est extrêmement répendu sur dans le développement web. Cette bilbliothèque intègre de nombreuse fonctions permettant la manipulation du DOM en toute simplicité. De plus, elle permet de rendre les pages web plus intéractive grâce à la gestion de nombreuses intéractions entre l'utilisateur et la page. Enfin, jQuery permet d'utiliser un même code a travers tous les navigateurs, sans se soucier des spécificités liées à chacun d'entre eux.

3.1 Mise en place de jQuery

La bibliohtèque jQuery consiste en un unique fichier JS contenant l'intégralité des fonctions. Ce fichier est téléchargeable depuis le site de jQuery (*http://jquery.com*) dans sa version compressée (jquery-X.X.X.min.js). Une fois le fichier téléchargé, on l'inclue dans une page HTML de la manière suivante:

Exemple

```
<!DOCTYPE html>
<head>
        <meta charset="utf-8" />
        <title>Titre de la page</title>
    <meta name="description" content="Description de la page">

        <script src=''jquery-X.X.X.min.js''></script>
        <script type=''text/javascript''>
                // Code JS
        </script>
</head>
<body>
        ...
</body>
</html>
```

De manière générale, on execute le code de jQuery une fois tout le DOM chargé. Pour ce faire, on défini une fonction qui s'executera une fois la page completement chargée:

Exemple

```
<!DOCTYPE html>
<head>
        <meta charset="utf-8" />
        <title>Titre de la page</title>
    <meta name="description" content="Description de la page">

        <script src=‘‘jquery-X.X.X.min.js’’></script>
        <script type=‘‘text/javascript’’>
                $(document).ready(function(){
                        // S'execute une fois tout le DOM chargé
                });
        </script>
</head>
<body>
        ...
</body>
</html>
```

3.2 Selecteurs

jQuery fonctionne sur la base d'une unique fonction **jQuery(...)** plus souvent utilisée dans sa version abrégée **$(...)**. Cette fonction prend en paramètre un selecteur **CSS** (Cascading Style Sheets) qui permet de définir le ou les éléments du DOM sur lesquels on souhaite intervenir.

Exemple

```
$('a')          // Toutes les balises <a>
$('.classe')    // Tous les éléments dont la class est ‘‘classe’’
$('#id')        // Tous les éléments dont l'id est ‘‘id’’
$('div p')      // Toutes les balises <p> contenu dans un <div>
$('input:text') // Tous les <input> donc le type est ‘‘text’’
```

On peut aussi sélectionner des éléments du DOM en fonction de leur attributs. L'attribut d'un élément du DOM est défini après le nom de la balise entre les chevrons.

Exemple

```
<a href=‘‘/index.html’’>Lien</a>            // ‘‘href’’ est un attribut
<input type=‘‘submit’’ value=‘‘Envoyer’’ /> // ‘‘type’’ et ‘‘value’’ sont des attributs
```

Avec jQuery, on peut sélectionner les attributs d'un élément du DOM avec la syntaxe suivante:

Exemple

```
$('[attribut=‘‘valeur’’]')  // L'attribut vaut ‘‘valeur’’
$('[attribut*=‘‘valeur’’]') // L'attribut contient ‘‘valeur’’
$('[attribut!=‘‘valeur’’]') // L'attribut ne vaut pas ‘‘valeur’’
$('[attribut^=‘‘valeur’’]') // L'attribut commence par ‘‘valeur’’
$('[attribut$=‘‘valeur’’]') // L'attribut termine par ‘‘valeur’’
$('[attribut~=‘‘valeur’’]') // L'attribut contient ‘‘valeur’’ délimité par des espaces
```

Enfin, on peut aussi se servir de la position de l'élement dans l'arborescence du DOM pour sélectionner un élément:

Exemple

```
$(':empty')        // Element qui n'a pas d'enfant
$(':first-child')  // Premier enfant
$(':last-child')   // Dernier enfant
```

```
$(':nth-child(n)') // Enieme enfant
$(':only-child')   // Seul enfant
$('a > b')         // Element b dont le parent direct est a
```

Notez qu'il est aussi possible d'utliser les fonctions ***.parent, .children*** pour se déplacer dans le DOM.

Exemple

```
$('div').children('p') // Les paragraphes contenus dans des divs
$('p').parent()        // Le parent de chaque paragraphe
```

La fonction jQuery renvoie systématiquement un objet jQuery. Celui-ci fonctionne d'une manière assez similaire à celle d'un tableau. Il possède un attribut "length" qui permet de connaitre le nombre d'éléments concernés par la requête.

Pour la suite des exercices, nous travaillerons sur les fichiers HTML disponibles à l'adresse suivante: ***http://labri.fr/perso/pbiasutt/KDJS/td01.zip***

Exercice 05. a) Ecrire un script Javascript qui affiche dans la console le nombre de lien (a) dans le fichier exo5.html.
b) Même question pour tous les paragraphe (p) dont la classe est ".paragraphe".
c) Même question pour tous les input (input) dont le type n'est pas "text".
d) Même question pour toutes les listes (ul) contenant plus de 2 éléments.

Exercice 06. La fonction **.css(propriete, valeur)** permet de modifier une propriété CSS de l'élément préalablement sélectionné. A l'aide de cette fonction, et de la prioriété "color" de CSS, mettez en rouge tous les premiers item de liste (li) du fichier exo6.html. Mettez en bleu le premier et le dernier paragraphe (p) de chaque bloc (div). A l'aide de la propriété "border" de CSS, mettez un bord rouge de 1px à toutes les images contenues dans une liste.

3.3 Modification d'un élément

jQuery implémente un large ensemble de fonction permettant de modifier les éléments du DOM.

Attributs La méthode ***.attr(attribut, valeur)*** permet de modifier ou d'ajouter un attribut à un élément. Notez que la méthode ***.removeAttr(attribut)*** permet de supprimer un attribut d'un élément.

Exemple

```
$('a').attr('about', '_blank'); // Rajoute l'attribut about=_blank à tous les <a>
$('a').removeAttr('href');      // Supprime le lien de tous les <a>
```

Exercice 07. Dans le fichier *exo7.html*, écrire un script Javascript qui transforme tous les champs de texte en bouton et qui remplace toutes les images par une image que de votre choix.

Jouer avec les classes Les classes permettent de rajouter des propriétés CSS à plusieurs éléments à la fois. jQuery permet de manipuler les classes des éléments du DOM très simplement

Exemple

```
$('p').addClass('classe');    // Ajoute la classe ``.classe'' à tous les paragraphes
$('p').removeClass('classe'); // Retire la classe ``.classe'' à tous les paragraphes
$('p').toggleClass('classe'); // Ajoute la classe si elle n'y est pas, la supprime sinon
$('p').hasClass('classe')     // Retourne vrai si l'élément possède la classe
```

Exercice 08. La balise **style** permet de définir du CSS directement dans un fichier HTML. Elle se place avant l'ouverture de la balise **body**.
a) A l'aide de cette balise, définissez les classe *.orange, .violet, .gras, .italique* ayant respectivement pour effet de mettre le texte en orange, en violet, en gras et en italique.
Dans le fichier *exo8.html*, écrire un script qui met en orange tous les premiers éléments d'une liste et en violet

tous les derniers éléments d'une liste. Rajouter un script qui met tous les titres (h1, h2, etc.) en italique, et tous les paragraphes en gras.
b) Ajoutez maintenant la classe *violet* à tous les premiers éléments de liste. Comment remettre les tous les premiers éléments de liste en orange?
c) Ajoutez la classe *violet* aux balises **h1**. Ajoutez maintenant la classe *orange*. Qu'observez-vous? Expliquez pourquoi.

3.4 Ajouter / Supprimer des éléments

En plus de modifier les propriétés CSS des éléments, jQuery possède plusieurs fonction permettant la modification du DOM. Ainsi, on peut ajouter, modifier ou supprimer des éléments de l'arborescence de la page.

Exercice 09. A l'aide de la console Javascript du navigateur, observer les effets des instructions suivantes sur un des fichiers utilisé au dessus:

Exemple

```
1 console.log($('ul').text());
2 console.log($('ul').html());
3 $('p').text('<a href=''#''>lien</a>');
4 $('p').html('<a href=''#''>lien</a>');
5 console.log($('input:text[name=nom]').text());
6 console.log($('input:text[name=nom]').val());
```

A quoi servent chacune de ces fonctions?

Exercice 10. A l'aide de la console Javascript du navigateur, observer les effets des instructions suivantes:

Exemple

```
1 $('p').append('Javascript<br />');
2 $('p').prepend('<br />Javascript');
3 $('p').remove();
4 $('ul').before('Before');
5 $('ul').after('After');
```

A quoi servent chacune de ces fonctions?

Exercice 11. A l'aide de la console Javascript du navigateur, observer les effets des instructions suivantes:

Exemple

```
1 $('<strong>SUPER!</strong>').appendTo('li');
2 $('<strong>OK!</strong>').prependTo('li');
3 $('<strong>HEY!</strong>').insertBefore('li');
4 $('<strong>OH!</strong>').insertAfter('li');
```

A quoi servent chacune de ces fonctions?

Vérifier vos réponses avec le chargé de TD avant de passer à la suite.

3.5 Les évenements

L'utilisation du Javascript est très utilisée dans le but de rendre les pages web plus dynamiques. Il permet de faciliter l'interaction entre l'utilisateur et la page web. jQuery implémente de nombreuses fonctions pour simplifier la gestion de ces intéractions, qu'elles soient faites depuis la souris, le clavier, ou encore en redimensionnant la page. La structure générale d'une interaction avec jQuery se définit comme suit:

Exemple

```
1 $('element').action(fonctionAExecuter);
```

Ici, lorsque l'interaction "action" est effectuée sur un élement "element" du DOM, la fonction "fonctionAExecuter" est appelée. En général, pour éviter de devoir définir une nouvelle fonction pour chaque intéraction, on fait appel à un ***callback***. Un ***callback*** est une fonction sans nom que l'on passe en argument et qui sera executée automatiquement par une autre fonction. Dans le cas d'une interaction jQuery, on peut définir le schéma général suivant:

Exemple

```
$('element').action(function(){
        // Ne s'execute que lorsque ``action'' est effectué
});
```

Notez que ce schéma s'applique par exemple à l'évemenement ".ready" défini sur le document. Voici une liste des événements couramment utilisés:

- **.click**: on clique sur un élément
- **.hover**: on survole un élément
- **.focus**: on séléctionne un élément (ex. un champ de texte)
- **.blur**: on désélectionne un élément
- **.keypress**: on tape sur une touche du clavier
- **.submit**: on valide/soumet un formulaire (form)
- **.change**: détecte un changement sur un élément (particulièrement utilisé pour les listes "select")
- **.off**: désactive un événement sur un élément

La liste complête des événements jQuery est accessible à l'adresse suivante: *https://api.jquery.com/category/events/*. Dans un ***callback***, on peut parfois chercher à accéder à l'élément déclancheur ou à un de ses enfants. De manière analogue à un objet, on peut appeler la méthode ***$(this)*** qui correspond directement à l'élément qui a déclanché l'événement.

Exemple

```
$('a').click(function(){
        console.log($(this).attr('href')); // Affiche l'URL du lien dans la console
});
```

Exercice 12. La fonction ***.each*** permet d'executer un code pour chaque élément de l'objet jQuery qui la précède.

Exemple

```
$('div').hover(function(){
        $(this).children('p').each(function(){
                console.log($(this).text());
        });
});
```

D'après vous, que produit ce code? Testez-le pour vérifier.

Exercice 13. a) Ecrire un script qui met en gras un lien lorsqu'il est survolé, et qui le remet comme d'origine lorsqu'il n'est plus survolé. b) Ecrire un script qui affiche les dimensions de la page dans la console lorsque celle-ci est redimmensionnée. Des fonctions jQuery permettent d'accéder directement à la largeur et la hauteur d'un élément, à vous de les trouver. c) Ecrire un code qui, dès qu'une touche est pressée, affiche sa valeur dans la console.

Exercice 14. En utilisant les événements présentés ci-dessus ainsi que vos connaissances Javascript et HTML:
a) Créez un formulaire contenant:

- Un champ de texte “nom”
- Un champ de texte “prenom”
- Un champ de texte numérique “age”
- Un “select” sexe avec deux options: M et F
- Une zone de texte (textarea) description
- Un bouton “submit”

b) Comment exectuer une fonction lorsque le formulaire est validé/soumis?
c) Ecrire un script qui, lorsque le formulaire est validé/soumis, récupère les valeurs du formulaire et les stock dans un objet “Utilisateur” contenant les attributs appropriés. L’objet devra ensuite être affiché dans la console.

Exercice 15. On cherche maintenant à faire un générateur de page web en Javascript. L’idée est qu’avec l’aide d’un formulaire très simple, on puisse créer de nouveaux éléments dans une page vierge. On peut ensuite choisir d’afficher le code HTML de la page ou de vider le contenu de la page. Le ficher *exo15.html* contient le squelette de notre générateur.

Selon le type d’élément sélectionné, la valeur du champs “contenu” représentera:

- **h1**: le contenu du titre
- **h2**: le contenu du titre
- **h3**: le contenu du titre
- **p**: le contenu du titre
- **img**: l’url de l’image
- **a**: l’url du lien
- **input:text**: le placeholder
- **input:submit**: le titre du bouton
- **label**: le titre du label

Lorsque l’option “Insérér avant” est cochée, le nouvel élément s’insère avant tous les autres dans la page. Sinon, il s’insère à la suite. Quand on clique sur le bouton “Ajouter”, le nouvel élément est créé dans le div “conteneur”. Lorsque l’on clique sur le bouton “Générer”, une popup s’affiche avec le code HTML de la page générée. Et lorsque l’on clique sur “Réinitialiser”, la page est vidée de son contenu.