IAM OUAGA

L'Institut Africain de Management Ouagadougou

Enseignant : SANOU Stéphane S.

Présentation

- Nom et Prénoms
- Vos Connaissances sur le JavaScript
- Vos attentes sur le cours

IAM
OUAGA

Objectifs

- Connaitre l'environnement de JavaScript;
- Explorer les fondements du JavaScript;
- Connaitre les concepts de base de JavaScript;
- Connaitre la mise en œuvre des principes objet avec le JavaScript.

IAM
OUAGA

Le plan

I. Introduction

II. Les notions de base

III. Les tableaux

III. Les chaines de caractères

IV. Les fonctions

V. L'orienté objets

IV. JavaScript & HTML

IAM OUAGA

Introduction

- Le langage JavaScript a été créé par Brench Eich pour le compte de Netscape Communications Corporation.
- En novembre 1996, il a servi de fondation à la première version du standard ECMA-262 décrivant le langage ECMAScript.
- JavaScript correspond donc à un dialecte de ce standard et a évolué indépendamment par la suite.
- Face au succès de JavaScript en tant que langage de script pour les pages Web, Microsoft a sorti, en août 1996, le langage JScript, similaire à JavaScript et correspondant également à un dialecte du langage ECMAScript.
- Ainsi, chaque navigateur supporte ECMAScript par l'intermédiaire de son propre dialecte.
- Par abus de langage, tous ces dialectes sont désignés par le terme JavaScript.

IAM
OUAGA

Introduction

- JavaScript est un langage de script dont le noyau est standardisé par la spécification ECMA-262. Cette dernière décrit le langage ECMAScript, dont JavaScript peut être considéré comme un dialecte, au même titre que JScript. Par abus de langage, JavaScript désigne en fait tous ces dialectes.
- En tant que langage interprété, JavaScript permet notamment de mettre en œuvre des scripts dans des pages Web. Il offre ainsi la possibilité d'interagir avec les différents langages et technologies utilisés dans les pages Web. En recourant à la technologie DOM, il permet en outre de manipuler la structure en mémoire des pages Web.
- Ce langage correspond véritablement à la clé de voûte des applications Web 2.0, aux interfaces graphiques riches et interactives.
- Longtemps freiné par les spécificités des navigateurs, JavaScript peut désormais recourir à différentes bibliothèques afin d'adresser ces spécificités. Les pages Web n'ont alors plus à les gérer directement.

IAM
OUAGA

Introduction

- Le JavaScript ou le JS est le langage de script par excellence des navigateurs Web;
- C'est un composant majeur de nos sites web;
- Les pages web sont composées de trois éléments essentiels :
  - HTML pour la structure et le squelette de la pages
  - Le CSS qui va décorer le HTML
  - Le JavaScript ou le JS qui assure l'interactivité et le la logique des pages web.
- Il existe une certaine confusion autour du terme JavaScript. De plus, contrairement à ce que son nom pourrait laisser penser, JavaScript et Java n'ont que peu de similitudes, hormis celles issues de leur parent commun, le langage C, et qui se limitent à la syntaxe de base. JavaScript est un vrai langage de programmation

II . La console

- La console est un outil pour développeur intégré dans les navigateurs web
  - Elle permet d'écrire du code JavaScript et de l'exécuté.
  - Exécutons le code suivant dans la console:

```
alert(''hello world'');
```

- Un autre outil très utilisé avec la console est log. Il permet suivre les étapes d'exécution d'un code à plusieurs étapes. Il s'écrit dans le code JavaScript et s'affiche dans la console. Il est très utilisé pour faire du débogage de code JavaScript.
- Exemple : `console.log('je suis à étape 3');`

II. Exécution de scripts JavaScript

- JavaScript n'étant pas un langage compilé, il doit être exécuté par le biais d'un interpréteur.
- beaucoup de ces interpréteurs sont disponibles gratuitement tandis que d'autres sont intégrés aux navigateurs Web.
- Nous nous concentrons sur l'utilisation de JavaScript dans les navigateurs dans le cadre de ce cours.

IAM
OUAGA

II. Utilisation de JavaScript dans le navigateur

- La façon d'utiliser un script JavaScript est identique pour tous les navigateurs.
- Elle consiste à utiliser la balise HTML générique script, avec l'attribut type spécifiant le type de langage de script. Dans notre cas, la valeur de cet attribut doit être text/javascript.
- Il est fortement conseillé de placer la balise script à l'intérieur de la balise head. Cette dernière étant évaluée avant le corps du document HTML matérialisé par la balise body, le code JavaScript est disponible avant la construction graphique de la page, celle-ci pouvant l'utiliser à cet effet si nécessaire.
- Néanmoins, le développeur peut insérer un bloc script n'importe où dans la page HTML.

II. Utilisation de JavaScript dans le navigateur

- Le code suivant illustre un squelette de page HTML intégrant un script JavaScript :

```
<html>
    <head>
        (...)
        <script type="text/javascript" (...)>
                (...)
        </script>
    </head>
    <body>
        (...)
    </body>
</html>
```

II. Utilisation de JavaScript dans le navigateur

- Il existe deux manières d'utiliser JavaScript dans une page Web :
  - Implémenter directement des traitements JavaScript dans la page, comme dans le code suivant :

```
<script type="text/javascript">
    var test = "Ceci est un test";
    alert(test);
</script>
```

II. Utilisation de JavaScript dans le navigateur

- Faire référence à une adresse contenant le code par
  - l'intermédiaire de l'attribut src de la balise script. Ainsi, aucun code n'est contenu dans cette dernière balise. Cette technique permet de partager entre plusieurs pages HTML des traitements ou des fonctions JavaScript communes et de les mettre en cache au niveau du navigateur. Elle permet également d'appeler une ressource qui génère dynamiquement du code JavaScript côté serveur.

II. Les commentaire JavaScript

- Les commentaires peuvent être utilisés pour expliquer le code JavaScript et le rendre plus lisible.
- Ils peuvent également être utilisés pour empêcher l'exécution, lors du test d'un code alternatif.
- Les commentaires sur une seule ligne commencent par //. Tout texte entre // et la fin de la ligne sera ignoré par JavaScript (ne sera pas exécuté).

```
// Change l'entete:
document.getElementById("myH").innerHTML = "My First Page";
// Change le paragraphe:
document.getElementById("myP").innerHTML = "My first paragraph.";
```

II. Les commentaire JavaScript

- Les commentaires multi-lignes commencent par / * et se terminent par * /. Tout texte entre / * et * / sera ignoré par JavaScript.

  Cet exemple utilise un commentaire sur plusieurs lignes (un bloc de commentaires) pour expliquer le code:

/* Ce code change l'entete avec id = "myH " et de paragraphe avec id = "myP " de ma page web : */

document.getElementById("myH").innerHTML = "My First Page"; document.getElementById("myP").innerHTML = "My first paragraph.";

II. Variables et typage

- JavaScript est un langage non typé. Cela signifie que le type d'une variable est défini uniquement au moment de l'exécution.
- La mise en œuvre d'une variable se réalise par l'intermédiaire du mot-clé var. L'interpréteur JavaScript a la responsabilité de créer la valeur du bon type en fonction de l'initialisation ou de l'affectation.
- Le langage n'impose pas l'initialisation des variables au moment de leur création et offre la possibilité d'en définir plusieurs en une seule instruction.

II. Variables et typage

- Le code suivant illustre la mise en œuvre de plusieurs variables :

```
// Variable initialisée avec une chaîne de caractères
var variable1 = "mon texte d'initialisation";
// Variable non initialisée
var variable2;
// Définition de plusieurs variables en une seule instruction
var variable3 = 2, variable4 = "mon texte d'initialisation";
```

- JavaScript permet de changer le type des variables au cours de l'exécution, comme l'illustre le code suivant :

```
var variable = "mon texte d'initialisation";
(...)
variable = 2;.
```

II. Variables et typage

- JavaScript définit plusieurs types, qui peuvent être classés en deux groupes : les types primitifs et les objets référencés.
- Les types primitifs correspondent à des données stockées directement dans la pile d'exécution, ces types étant définis de manière littérale directement à partir de valeurs.
- Le tableau ci-dessous récapitule les types primitifs définis par JavaScript.
- **Tableau 1 : Types primitifs de JavaScript**

Type	Description
`Boolean`	Type dont les valeurs possibles sont true et false.
`Null`	Type dont l'unique valeur possible est null. Une variable possède cette valeur afin de spécifier qu'elle a bien été initialisée mais qu'elle ne pointe sur aucun objet.
`Number`	Type qui représente un nombre.
`String`	Type qui représente une chaîne de caractères.
`Undefined`	Type dont l'unique valeur possible est undefined. Une variable définie possède cette valeur avant qu'elle soit initialisée.

II. Variables et typage

La méthode typeof permet de déterminer le type d'une variable. Elle renvoie la valeur object dans le cas de variables par référence, et ce quel que soit le type de l'objet.

- Le code suivant présente la détection de types primitifs :

```
var variable1;
alert(" type de variable1 : "+(typeof variable1));
// variable1 est de type undefined
var variable2 = null;
alert(" type de variable2 : "+(typeof variable2));
// variable2 est de type object
var variable3 = 12.3;
alert(" type de variable3 : "+(typeof variable3));
// variable3 est de type number
```

II. Variables et typage

La méthode typeof permet de déterminer le type d'une variable. Elle renvoie la valeur object dans le cas de variables par référence, et ce quel que soit le type de l'objet.

- Le code suivant présente la détection de types primitifs :

```
var variable1;
alert(" type de variable1 : "+(typeof variable1));
// variable1 est de type undefined
var variable2 = null;
alert(" type de variable2 : "+(typeof variable2));
// variable2 est de type object
var variable3 = 12.3;
alert(" type de variable3 : "+(typeof variable3));
// variable3 est de type number
```

II. Variables et typage

```
var variable4 = "une chaîne de caractères";

alert(" type de variable4 : "+(typeof variable4));

// variable4 est de type string

var variable5 = true;

alert(" type de variable5 : "+(typeof variable5));

// variable5 est de type boolean
```

II. Variables et typage

- JavaScript fournit des méthodes afin de convertir un type primitif en un autre. Il supporte les conversions de types primitifs en chaînes de caractères et de chaînes de caractères en nombres entiers ou réels.

Le code suivant illustre l'utilisation de la méthode toString pour les types boolean et number :

```
var booleen = true;
var variable1 = booleen.toString();
// variable1 contient la chaîne de caractère « true »
var nombreEntier = 10;
var variable2 = nomEntier.toString();
// variable2 contient la chaîne de caractère « 10 »
```

II. Variables et typage

- Dans le cas des nombres, la méthode toString peut être utilisée avec un paramètre pour spécifier la représentation du nombre souhaité. Par défaut, la base décimale est utilisée. Il est cependant possible de spécifier, par exemple, les valeurs 2 pour la base binaire et 16 pour l'hexadécimale.
- Le code suivant décrit l'utilisation de cette méthode :

```
var nombreEntier = 15;
var variable1 = nombreEntier.toString();
// variable1 contient la chaîne de caractère « 10 »
var variable2 = nombreEntier.toString(2);
// variable2 contient la chaîne de caractère « 1111 »
var variable4 = nombreEntier.toString(16);
// variable4 contient la chaîne de caractère « f »
```

II. Variables et typage

- Dans le cas des nombres, la méthode toString peut être utilisée avec un paramètre pour spécifier la représentation du nombre souhaité. Par défaut, la base décimale est utilisée. Il est cependant possible de spécifier, par exemple, les valeurs 2 pour la base binaire et 16 pour l'hexadécimale.
- Le code suivant décrit l'utilisation de cette méthode :

```
var nombreEntier = 15;
var variable1 = nombreEntier.toString();
// variable1 contient la chaîne de caractère « 10 »
var variable2 = nombreEntier.toString(2);
// variable2 contient la chaîne de caractère « 1111 »
var variable4 = nombreEntier.toString(16);
// variable4 contient la chaîne de caractère « f »
```

II. Variables et typage

- La création de nombres à partir de chaînes de caractères se réalise par l'intermédiaire des méthodes parseInt et parseFloat.
- Ces dernières prennent en paramètres des chaînes de caractères représentant des nombres. Comme précédemment, il est possible de spécifier la base utilisée afin de réaliser la conversion soit directement dans la chaîne elle-même, soit par l'intermédiaire d'un paramètre.
- Le code suivant illustre la création d'un nombre entier à partir d'une chaîne de caractères :

```
var entier1 = parseInt("15");
// entier1 contient le nombre 15
var entier3 = parseInt("f", 16);
// entier3 contient le nombre 15
```

- Le code suivant présente la création d'un nombre réel à partir d'une chaîne de caractères :

```
var reel = parseFloat("15.5");
// reel contient le nombre réel 15,5
```

II. Les opérateurs

- Le langage JavaScript fournit différents types d'opérateurs utilisables directement dans les applications.
- Le tableau 2 récapitule les types d'opérateurs supportés par JavaScript.

Tableau 2 : **Types d'opérateurs de JavaScript**

Type	Description
Affectation	Permet d'affecter une valeur ou une référence à une variable. L'unique opérateur de ce type est =. L'opérateur d'affectation peut être combiné à ceux de calcul afin de réaliser des calculs sur l'affectation. Par exemple, l'expression « valeur += 10 » correspond à « valeur = valeur + 10 ».
Calcul	Permet de réaliser les calculs de base sur les nombres.
Comparaison	Permet de réaliser des comparaisons entre différentes variables.
Concaténation	Permet de concaténer deux chaînes de caractères. L'unique opérateur de ce type est +.
Conditionnel	Permet d'initialiser la valeur d'une variable en se fondant sur une condition.
Égalité	Permet de déterminer si différentes variables sont égales.
Logique	Permet de combiner différents opérateurs de comparaison afin de réaliser des conditions complexes.
Manipulation de bits	Permet de manipuler des variables contenant des bits.
Unaire	Mot-clé du langage se fondant sur un seul élément.

II. Les opérateurs

- Les opérateurs unaires correspondent à des mots-clés ou symboles du langage qui se fondent sur un seul élément. C'est le cas de typeof, détaillé précédemment, qui détermine le type d'une variable, et de new, delete et instanceof, qui permettent respectivement de créer et libérer la mémoire allouée pour un objet ainsi que de déterminer finement son type.
- Les opérateurs ++ et -- permettent respectivement d'incrémenter et de décrémenter des nombres en les préfixant ou les suffixant. La position de l'opérateur par rapport au nombre permet de déterminer le moment où est affectée la variable sur laquelle il porte.
- Le code suivant illustre ce mécanisme :

```
var unNombre = 10;
var unAutreNombre = unNombre++;// unAutreNombre vaut 10 et unNombre vaut 11
unAutreNombre = ++unNombre; // unAutreNombre vaut 12 et unNombre vaut 12
```

II. Les opérateurs

- Les opérateurs + et - permettent de convertir une chaîne de caractères en nombre de la même façon que la méthode parseInt. Contrairement à l'opérateur +, - change en plus le signe du nombre.
- Le code suivant illustre la mise en œuvre des opérateurs + et - :

```
var chaine = "10"; // chaine est de type string

var nombre = +chaine; // nombre est de type number

var autreNombre = -chaine; // autreNombre est de type number et l'opposé de nombre
```

II. Les opérateurs

- Les opérateurs de calcul permettent de réaliser les calculs élémentaires entre nombres. Le langage JavaScript en définit cinq.
- Les deux premiers, dont les symboles sont + et -, concernent l'addition et la soustraction. Les deux suivants, dont les symboles sont * et /, correspondent à la multiplication et à la division. Le symbole % permet de récupérer le reste de la division entière d'un nombre par un autre (modulo).
- Le code suivant présente leur mise en oeuvre :

```
var unNombre = 26;
var unAutreNombre = 3;
var uneOperation = unNombre * unAutreNombre; // uneOperation contient 78
var uneAutreOperation = unNombre / unAutreNombre;
// uneAutreOperation vaut 8,666666666666666
var uneDerniereOperation = unNombre % 5; // uneDerniereOperation vaut 1
```

IAM
OUAGA

II. Les opérateurs

- JavaScript définit la valeur particulière Infinity pour les nombres.
- Elle correspond à une valeur littérale positive représentant un nombre infini. Avec la version 1.2 de JavaScript, cette valeur est accessible par l'intermédiaire des propriétés POSITIVE_INFINITY et NEGATIVE_INFINITY de la classe Number. Avec la version 1.3, Infinity est rattaché à l'objet Global de JavaScript.
- Ce nombre particulier est supérieur à tous les nombres, y compris lui-même. Il est obtenu, par exemple, lors d'une division par zéro. Une division par ce nombre donne le résultat 0.

II. Les opérateurs

- L'opérateur logique permet de combiner différents opérateurs de comparaison afin de réaliser des conditions complexes. Il existe trois opérateurs : !, && et ||.
- L'opérateur ! correspond à la négation. && correspond à l'opérateur ET, qui réalise des opérations logiques en se fondant sur deux opérandes.
- Le code suivant illustre la mise en œuvre de ces opérateurs en combinaison avec l'opérateur de comparaison :

```
var unTest = 10 < 11 && 15 > 16; // unTest contient false
var unAutreTest = 10 < 11 || 15 > 16; // unTest contient true
```

- Ce type d'opérateur peut être utilisé dans les structures de contrôle conditionnelles et de boucle.

II. Les opérateurs

- **Opérateur conditionnel** peut initialiser une variable dont la valeur se fonde sur le résultat d'une condition.
  - Sa syntaxe s'appuie sur les caractères ? et : afin de définir trois blocs.
- Le premier bloc correspond à la condition, le second à la valeur à utiliser lorsque la condition est vérifiée et le troisième à la valeur quand la condition ne l'est pas.
- Le code suivant illustre l'utilisation de cet opérateur :

```
var valeurCondition = (...);
var maValeur = valeurCondition == "vrai"? "La condition est vraie": "la condition est fausse";
```

II. Les structures de contrôle

- Il existe deux types de structures de conditions. La première est fondée sur la combinaison d'instructions if, else if et else.
- Il obéit à la syntaxe suivante :

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

II. Les structures de contrôle

- Il existe deux types de structures de conditions. La première est fondée sur la combinaison d'instructions if, else if et else.
- Il obéit à la syntaxe suivante :

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

II. Les structures de contrôle

- Il est possible d'utiliser autant de blocs else if que nécessaire.
- Le code suivant en donne un exemple d'utilisation :

```
var chaine = (...);
if( chaine == "une chaine" )
{
        alert("La variable chaine est égale à 'une chaine'");
}
else if( chaine== "une autre chaine" ) {
        alert("La variable chaine est égale à 'une autre chaine'");
}
else {
        alert("La variable chaine est différente de"+ " 'une chaine' et 'une autre chaine'");
}
```

II. Les structures de contrôle

- Le second type de structures permet de réaliser la même fonctionnalité que précédemment à l'aide de la combinaison d'instructions switch, case et default. Cela permet de mettre en œuvre des traitements en fonction de la valeur d'une variable.
- Il obéit à la syntaxe suivante :

```
switch( variable ) {
        case valeur:
        (...)
        break;
        default:
        (...)
}
```

II. Les structures de contrôle

- Il est possible d'utiliser autant de blocs case que nécessaire. Le code suivant en donne un exemple d'utilisation :

```
var chaine = (...);
switch( chaine ) {
case "une chaine":
        alert("La variable chaine est égale à 'une chaine'");
break;
case "une autre chaine":
        alert("La variable chaine est égale à 'une autre chaine'");
break;
default:
                alert("La variable chaine est différente de"+ "
'une chaine' et 'une autre                chaine'");
        }
```

II. Les boucles

- JavaScript définit quatre types de boucles, for, for…in, while et do…while.
- La première s'appuie sur le mot-clé for afin de spécifier à la fois les traitements d'initialisation réalisés à l'entrée de la boucle, la condition de sortie et les traitements à réaliser près chaque itération. Tant que la condition de sortie est vraie, l'itération continue.
- La syntaxe de la boucle for est la suivante :

```
for( traitements d'initialisation ; condition de fin ;traitements à effectuer après chaque itération)
 {
            (...)
}
```

II. Les boucles

- Il est recommandé de définir des variables locales dans les traitements d'initialisation afin de réduire leur portée à la boucle. Le code suivant en donne un exemple d'utilisation :

```
for( var cpt=0; cpt<10; cpt++ )
 {
        alert("Valeur du compteur: "+cpt);
}
```

II. Les boucles

- La boucle for…in est une variante de la précédente qui se sert du mot-clé for conjointement avec le mot-clé in, ce dernier permettant de spécifier la variable à utiliser.
- Cette boucle permet de parcourir tous les éléments des tableaux indexés ou des objets, comme nous les verrons par la suite. Sa syntaxe est la suivante :

```
for( variable in structure ) {
(...)
}
```

- Comme précédemment, il est recommandé d'utiliser des variables locales dans la déclaration de la boucle. Le code suivant en donne un exemple d'utilisation :

```
var tableauIndexe = {
"cle1": "valeur1",
"cle2": "valeur2"
};
for( var cle in tableau ) {
alert("Valeur pour la clé "+cle+": "+ tableauIndexe[cle]);}
```

II. Les boucles

- La boucle while permet de spécifier la condition de fin. Aussi longtemps que cette condition est vraie, les traitements sont répétés. La syntaxe de cette boucle est la suivante :

```
while( condition de fin ) {
        (...)
}
```

- Le code suivant en donne un exemple d'utilisation :

```
var nombre = 0;
while( nombre < 10 )
{
        nombre++;
}
```

II. Les boucles

- La dernière boucle est une variante de la précédente, qui permet d'effectuer le bloc avant la première condition.
- La syntaxe de cette boucle est la suivante :

```
do {
(...)
} while( condition de fin );
```

- Le code suivant en donne un exemple d'utilisation :

```
var nombre = 0;
Do
 {
            nombre++;
}
while( nombre < 10 );
```

II. Les Méthodes de base

- Le langage JavaScript fournit des méthodes de base pouvant être appelées directement dans les scripts.
- Nous avons détaillé précédemment les méthodes dont le nom commence par parse, méthodes permettant de convertir une chaîne de caractères en un type primitif.
- La méthode eval permet d'interpréter le code JavaScript contenu dans une chaîne de caractères.
- Le code suivant donne un exemple d'utilisation de cette méthode :

```
var codeJavaScript = "var maVariable = 10;" + "function incrementer(parametre) { return parametre+1; }";
eval(codeJavaScript);
var retour = incrementer(maVariable);
// retour contient la valeur 11
```

II. Les Méthodes de base

- JavaScript propose un ensemble de méthodes permettant de détecter le type et la validité de la variable passée en paramètre. Ces méthodes, dont le nom commence par is, s'avèrent très utiles puisque le langage JavaScript n'est pas typé/*.
- Le tableau 3 récapitule ces méthodes de détection de types et de validité des variables.

Tableau 3 : Méthodes de détection de types et de validité des variables

Méthode	Description
`isArray`	Détermine si le paramètre est un tableau.
`isBoolean`	Détermine si le paramètre est un booléen.
`isEmpty`	Détermine si un tableau est vide.
`isFinite`	Détermine si le paramètre correspond à un nombre fini.
`isFunction`	Détermine si le paramètre est une fonction.
`isNaN`	Détermine si la valeur du paramètre correspond à NaN (Not a Number).
`isNull`	Détermine si le paramètre est null.
`isNumber`	Détermine si le paramètre est un nombre.
`isObject`	Détermine si le paramètre est un objet.
`isString`	Détermine si le paramètre est une chaîne de caractères.
`isUndefined`	Détermine si le paramètre est indéfini, c'est-à-dire une référence non initialisée.

II. Les Méthodes de base

- Le code suivant illustre la mise en œuvre des plus utilisées de ces méthodes :

```
var tableau = [];
alert(isEmpty(tableau)); // Affiche true
alert(isFunction(tableau)); // Affiche false
var booleen = true;
alert(isBoolean(booleen)); // Affiche true
var chaine = "Ceci est un test";
alert(isString(chaine)); // Affiche true
```

III. Les tableaux

- Le langage JavaScript offre deux types de tableaux. Nous décrivons dans cette section leur création par l'intermédiaire de valeurs littérales. Les tableaux peuvent également être gérés par l'intermédiaire de la classe Array.
- Le premier type correspond à un tableau classique, géré par index.
- Le code suivant illustre la création et l'utilisation de ce type de tableau :

```
// Initialisation d'un tableau vide
var tableau1 = [];
tableau1[0] = "Le premier élément";
tableau1[1] = "Le seconde élément";
// Initialisation d'un tableau avec des éléments
var tableau2 = ["Le premier élément", "Le seconde élément" ];
tableau1[2] = "Le troisième élément
```

III. Les tableaux

- Le parcours d'un tableau de ce type se réalise en s'appuyant sur les indices avec la boucle for, comme dans le code suivant :

```
var tableau = [ "element1","element2" ];
(...)
for(var cpt=0 ; cpt<tableau.length ; cpt++)
{
  alert("Elément pour l'index "+cpt+" : "+tableau[cpt]);
}
```

- Bien qu'il soit possible de manipuler les tableaux de ce type en utilisant les indices, cette approche peut vite devenir fastidieuse.
- L'utilisation des méthodes de manipulation des tableaux est recommandée. Malgré le fait qu'elles soient rattachées à la classe Array, elles peuvent être utilisées directement avec un tableau défini de manière littérale.

III. Les tableaux

- Le tableau ci-dessous récapitule les méthodes permettant de manipuler les tableaux en JavaScript.

Tableau 4: Méthodes de la classe Array permettant de manipuler les tableaux

Méthode	Paramètre	Description
`concat`	Tableau à ajouter	Concatène un tableau à un autre.
`join`	Séparateur	Construit une chaîne de caractères à partir d'un tableau en utilisant un séparateur.
`pop`	-	Supprime et retourne le dernier élément d'un tableau.
`push`	Élément à ajouter	Ajoute des éléments à la fin d'un tableau.
`reverse`	-	Inverse l'ordre des éléments d'un tableau.
`shift`	-	Retourne le premier élément et le supprime du tableau. Décale vers la gauche tous ses éléments.
`slice`	Indices de début et de fin	Retourne une partie du tableau en fonction d'indices de début et de fin.
`sort`	Rien ou une fonction de tri	Trie les éléments d'un tableau. Par défaut, le tri s'effectue par ordre alphabétique. Une fonction de tri peut néanmoins être passée en paramètre afin de spécifier la stratégie de tri.
`splice`	Indices de début et de fin et éventuellement éléments à ajouter	Permet d'insérer, de supprimer ou de remplacer des éléments d'un tableau.
`toString`	-	Convertit un tableau en une chaîne de caractères.
`unshift`	Éléments à ajouter	Permet d'ajouter un élément au début d'un tableau et de décaler tous ses éléments vers la droite.

III. Les tableaux

- Détaillons les principales méthodes de manipulation des tableaux.
- La méthode concat ajoute un tableau à la fin d'un autre, comme dans le code suivant :

```
var tableau = [ "element1", "element2" ];
var tableauAAjouter = [ "element3", "element4" ];
//Retourne un tableau correspondant à la concaténation des deux
var tableauResultat = tableau.concat(tableauAAjouter);
```

III. Les tableaux

- La méthode join permet de calculer une chaîne de caractères à partir d'un tableau.
- Elle utilise le séparateur fourni en paramètre afin de séparer les différents éléments du tableau dans la chaîne. Le code suivant donne un exemple d'utilisation de cette méthode :

```
var tableau = [ "element1", "element2", "element3", "element4" ];
var chaine = tableau.join(",");
//chaine contient « element1,element2,element3,element4 »
```

III. Les tableaux

- La méthode reverse inverse simplement l'ordre des éléments d'un tableau, comme dans le code suivant :

```
var tableau = [ "element1", "element2", "element3", "element4" ];
var resultat = tableau.reverse();
/*resultat correspond au tableau [ "element4", "element3", "element2", "element1" ] */
```

- La méthode slice permet l'extraction d'une sous-partie d'un tableau en se fondant sur des indices de début et de fin. Ce dernier peut cependant être omis.
- Dans ce cas, sa valeur correspond à l'indice de fin de tableau. L'élément correspondant à l'indice de fin n'est pas pris en compte dans le sous-tableau.
- Le code suivant donne un exemple d'utilisation de cette méthode :

```
var tableau = [ "element1", "element2", "element3", "element4" ];
var resultat = tableau.slice(1,3);
/*resultat correspond au tableau [ "element2", "element3" ] */
```

IV. Les chaînes de caractère

- JavaScript gère les chaînes de caractères de manière similaire à d'autres langages tels que Java.
- Ces dernières peuvent cependant être définies littéralement, aussi bien avec des guillemets ou des apostrophes, comme dans le code suivant :

```
var chaine1 = "ma chaîne de caractère"; var chaine2 = 'mon autre chaîne de caractère';
```

- Le langage JavaScript introduit la classe String correspondante.
- Le code suivant illustre la façon de créer une chaîne de caractères par l'intermédiaire de cette classe :

```
var chaine1 = new String("ma chaîne de caractère");
```

Le tableau 5 récapitule les méthodes relatives à la manipulation de chaînes

IV. Les chaînes de caractère

Tableau 5 Méthodes de manipulation de chaînes de la classe *String*

Méthode	Paramètre	Description
`charAt`	Index du caractère dans la chaîne	Retourne le caractère localisé à l'index spécifié en paramètre.
`charCodeAt`	Index du caractère dans la chaîne	Retourne le code du caractère localisé à l'index spécifié en paramètre.
`concat`	Chaîne à concaténer	Concatène la chaîne en paramètres à la chaîne courante.
`fromCharCode`	Chaîne de caractères Unicode	Crée une chaîne de caractères en utilisant une séquence Unicode.
`indexOf`	Chaîne de caractères	Recherche la première occurrence de la chaîne passée en paramètre et retourne l'index de cette première occurrence.
`lastIndexOf`	Chaîne de caractères	Recherche la dernière occurrence de la chaîne passée en paramètre et retourne l'index de cette dernière occurrence.
`match`	Expression régulière	Détermine si la chaîne de caractères comporte une ou plusieurs correspondances avec l'expression régulière spécifiée.
`replace`	Expression régulière ou chaîne de caractères à remplacer puis chaîne de remplacement	Remplace un bloc de caractères par un autre dans une chaîne de caractères.
`search`	Expression régulière de recherche	Recherche l'indice de la première occurrence correspondant à l'expression régulière spécifiée.
`slice`	Index dans la chaîne de caractères	Retourne une sous-chaîne de caractères en commençant à l'index spécifié en paramètre et en finissant à la fin de la chaîne initiale si la méthode ne comporte qu'un seul paramètre. Dans le cas contraire, elle se termine à l'index spécifié par le second paramètre.
`split`	Délimiteur	Permet de découper une chaîne de caractères en sous-chaînes en se fondant sur un délimiteur.
`substr`	Index de début et de fin	Méthode identique à la méthode `slice`
`substring`	Index de début et de fin	Méthode identique à la précédente
`toLowerCase`	-	Convertit la chaîne de caractères en minuscules.
`toString`	-	Retourne la chaîne de caractère interne sous forme de chaînes de caractères.
`toUpperCase`	-	Convertit la chaîne de caractères en majuscules.
`valueOf`	-	Retourne la valeur primitive de l'objet. Est équivalente à la méthode `toString`.

IAM OUAGA

IV. Les chaînes de caractère

- Attardons-nous maintenant sur les principales méthodes de manipulation des chaînes de caractères.
- Tout d'abord, la méthode concat permet de retourner la concaténation de deux chaînes de caractères. Le code suivant illustre l'utilisation de cette méthode :

```
var uneChaine = "Une chaine.";
var uneAutreChaine = "Une autre chaine.";
var resultat = uneChaine.concat(uneAutreChaine);
// resultat contient « une chaine.Une autre chaine. »
```

IV. Les chaînes de caractère

- Les méthodes indexOf et lastIndexOf déterminent l'indice de première occurrence d'un caractère dans une chaîne de caractères en la parcourant respectivement depuis son début et depuis sa fin.
- Le code suivant illustre sa mise en oeuvre :

```
var uneChaine = "Le début de la chaine. Sa fin.";
var indicePointDebut = uneChaine.indexOf(".");
//La valeur de indicePointDebut est 10
var indicePointFin = uneChaine.lastIndexOf(".");
//La valeur de indicePointFin est 21
```

IV. Les chaînes de caractère

- La méthode replace remplace la première occurrence d'une chaîne de caractères par une autre, comme dans le code suivant :

```
var uneChaine = "Le début de la chaine. Sa fin.";
var resultat = uneChaine.replace(". "," ; ");
//resultat contient « Le début de la chaine ; Sa fin. »
```

- Cette méthode peut être également utilisée avec une expression régulière en paramètre afin de déterminer la chaîne de caractères à remplacer. Nous détaillons cette utilisation à la section relative aux expressions régulières

IV. Les chaînes de caractère

- Les méthodes toLowerCase et toUpperCase permettent respectivement de mettre une chaîne en majuscules et en minuscules, comme l'illustre le code suivant :

```
var uneChaine = "Une chaine";
var chaineMajuscule = uneChaine.toUpperCase();
// chaineMajuscule contient « UNE CHAINE »
var chaineMinuscule = uneChaine.toLowerCase();
// chaineMajuscule contient « une chaine »
```

Les méthodes match et search mettent en oeuvre des expressions régulières

V. Les fonctions

- Les fonctions représentent le concept de base de la programmation JavaScript afin de modulariser les traitements.
- Elles possèdent des spécificités par rapport à des langages tels que Java ou C++.
- Les fonctions constituent la clé de voûte du développement objet en JavaScript.
- Le mot-clé function permet de mettre en œuvre les fonctions selon la syntaxe suivante :

```
function nomDeLaFonction(parametre1, parametre2, ...)
    {
    //Code de la fonction
    }
```

- Comme dans la plupart des langages, une fonction est utilisée par le biais de son nom, suivi de parenthèses.
- Ces dernières permettent de délimiter la liste de ses paramètres, comme dans le code suivant :

V. Les fonctions

```
function test(parametre1, parametre2) {
return parametre1+" - "+parametre2;
}
var retour = test("param1","param2");
//La variable retour contient la valeur: "param1 – param2«
```

La valeur de retour, dans le cas où la fonction ne renvoie pas d'élément, est undefined.

- JavaScript offre la possibilité d'affecter une fonction à une ou plusieurs variables.
- La variable est alors de type function. L'affectation peut être réalisée à partir d'une fonction existante ou, au moment de la création de la variable, à l'aide d'une fonction anonyme.
- Le code suivant illustre l'affectation d'une fonction existante à une variable :

```
function test(parametre1, parametre2) {
return parametre1+" - "+parametre2;
}
var maFonction = test;
```

V. Les fonctions

- Dans le cas d'une fonction anonyme, la syntaxe est la suivante :

```
var maFonction = function(parametre1, parametre2, ...)
 {
                    //Code de la fonction
};
```

Le code suivant illustre la mise en oeuvre de ce mécanisme :

```
var maFonction = function (parametre1, parametre2)
{
                    return parametre1+" - "+parametre2;
}
```

- Lors de l'appel de la fonction, le nom de la variable est utilisé. Dans le cas de l'affectation d'une fonction existante, le nom de la variable est équivalent au nom de la fonction, comme dans le code suivant :

V. Les fonctions

```
function test(parametre1, parametre2) {
return parametre1+" - "+parametre2;
}
var maFonction = test;
var retour = maFonction("param1","param2");
```

- Le code suivant illustre la mise en œuvre d'une fonction anonyme :

```
var maFonction = function(parametre1, parametre2) {
return parametre1+" - "+parametre2;
}
var retour = maFonction("param1","param2");
```

V. Les fonctions

- Le concept de signature permet d'identifier complètement une fonction.
- la signature englobe à la fois:
  - le nom de la fonction,
  - les types de ses paramètres
  - et son type de retour.
  - En cas d'utilisation d'une fonction sur un objet, sa visibilité est également prise en compte.
- Le langage JavaScript n'utilise pas les signatures pour identifier les fonctions et se fonde uniquement sur leurs noms. Ainsi, deux fonctions globales ou d'un même objet possédant le même nom ne peuvent coexister avec ce langage.

V. Les fonctions

Soient les deux fonctions suivantes:

```
function maFonction(parametre1, parametre2) {
        return parametre1+" - "+parametre2;
}
function maFonction(parametre1) {
        return parametre1;
}
        var retour = maFonction("param1", "param2");
```

- Nous pourrions nous attendre à ce que le contenu de la variable retour soit la chaîne de caractères param1 - param2, alors que nous constatons que la variable contient la chaîne de caractères param1.
- La raison à cela est que la fonction maFonction appelée est la seconde, puisqu'elle a été définie en dernier et est donc la seule à être prise en compte.
- JavaScript introduit la variable arguments dans le corps la fonction pour résoudre ce problème.

V. Les fonctions

- Le code suivant illustre la façon de corriger le code de la section précédente pour supporter différentes signatures de fonctions :

```
function maFonction() {
    if( arguments.length == 1 ) {
                return arguments[0];
    }
    if( arguments.length == 2 ) {
                return arguments[0]+" - "+arguments[1];
    }
}
```

- var retour = maFonction("param1", "param2");
- //La valeur de la variable retour est « param1 – param2 »
- var retour = maFonction("param1");
- //La valeur de la variable retour est « param1 »

V. Les fonctions

```
<html>
<body>
    <p>Click the button to display the date.</p>
    <button onclick="displayDate()">The time is?</button>
    <script>
        function displayDate() {
        document.getElementById("demo").innerHTML = Date();
        }
    </script>
    <p id="demo"></p>
</body>
</html>
```

VI. L'orienté objet

- Un autre type de variable de JavaScript ce sont les objets.
- Un objet est plus adapté à décrire le monde tel qu'on le perçoit parce qu'un objet a des propriétés et des méthodes(fonctions).
- Un objet JavaScript se définit comme suite :

```
var object = {
        nomPropriete1 : valeurPropriete1
        nomPropriete2 : valeurPropriete2
        nomMethode1: function(){
        }
};
```

VI. L'orienté objet

- Créons un l'objet chien en exemple :

```
var chien= {
        nom : '' Milou ''
        couleur: '' blanc''
        age : 4
};
```

Pour accéder aux propriétés de l'objet chien:

Console.log(chien.nom);

Ou console.log(chien['' nom'']);

VI. L'orienté objet

- On peut accéder à toutes les propriétés de l'objet avec une boucle :

```
for( var propriété in chien){
        console.log(chien [propriété] );
}
```

- On peut en outre créer un objet avec le constructeur new

```
var chien = new Object();
chien.nom = ‘’choupette”;
chien.color = ‘’ blanc” ;
chien.age = 5 ;
dog.aboi= function(){ console.log(‘’wouf wouf ”)}
```

Pour exécuter la méthode : chien.aboi();

VI. L'orienté objet

- Avec cette façon de créer des objets on serait obligé de reproduire le même code si on veut créer un deuxième chien. Ce qui peut être assez fastidieux. Pour pallier à cette difficulté nous allons utiliser les fonctions constructeurs.
- Le constructeur est sera vu comme une usine de production d'objet.
- Si nous voulons créer des objets qui ont le même type alors nous utiliser les fonctions constructeurs.

```
function Chien (nomchien, couleurchien, agechien){
        this.nom = nomchien;
        this.couleur = couleurchien;
        this.age = agechien ;
}
```

Nom, couleur, age sont des proprietes de l'objet chien.

- Pour créer un objet de Chien :

```
var petitCaniche = new Chien(‘’ Choupette’’, ‘’ blanc’’, 5 );
console.log(petitCaniche);
```

// Nous affiche un objet de type Chien avec les propriete nom, couleur, age.