JavaScript
Les Objets

Préparé par : Larbi HASSOUNI

Tout est objet en JavaScript

En JavaScript, toutes les valeurs sont des objets à l'exception des valeurs primitives.

Les valeurs primistives sont :

- les strings
- Les nombres,
- Les valeurs booléennes true et false
- null
- undefined.

Création des objets JavaScript

JavaScript dispose de plusieurs façons pour créer de nouveaux objets:

- Définir et créer un seul objet en utilisant un **objet literal**.
- Définir et créer un seul objet en utilisant le mot **clé new** (new Object())
- Définir une **fonction constructeur d'objet**, puis créer des objets en utilisant la fonction.

Création d'un objet littéral

- C'est la façon la plus simple pour créer un objet JavaScript.
- En utilisant un objet littéral, nous définissons et créons en même temps un objet par une seule instruction.
- Un objet littéral est une liste de paires nom:valeur placées entre deux accollades {}.
- L'exemple de la diapositive suivante crée un objet littéral avec 4 propriétés et une métode:

```
var etudiant = {
        cne : 1611442233,
        prenom: "Ali",
        nom : "Moha",
        age : 18,
        nomComplet : function() {
                return this.prenom + " " + this.nom;
        }
};
```

Les propriétés sont : cne, prenom, nom, age et nomComplet.

La méthode est : nomComplet

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<p id="para1"></p>
<p id="para2"></p>
<script>
var etudiant = {
    cne : 1611442233,
    prenom: "Ali",
    nom : "Moha",
    age : 18,
    nomComplet : function() {
       return this.prenom + " " + this.nom;
    }
};
document.getElementById("para1").innerHTML = etudiant.nomComplet();
</script>
</body>
</html>
```

Création d'un objet avec new

L'exemple ci-dessous montre comment créer un objet avec new:

```
var etudiant = new Object();
etudiant.cne = 1611442233;
etudiant.prenom = "Ali";
etudiant.nom = "Moha";
etudiant.age = 18;
etudiant.nomComplet = function() {
      return this.prenom + " " + this.nom;
};
```

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<p id="para1"></p>

<script>
var etudiant = new Object();
etudiant.cne = 1611442233;
etudiant.prenom = "Ali";
etudiant.nom = "Moha";
etudiant.age = 18;
etudiant.nomComplet = function() {
        return this.prenom + " " + this.nom;
};
document.getElementById("para1").innerHTML = etudiant.nomComplet();
</script>
</body>
</html>
```

Les deux exemples précédents sont équivalents.

Il n'ya donc pas besoin d'utiliser new Object().

Pour une meilleure performance et davantage de simplicité, utiliser un objet literal au lieu de new Object().

Utilisation d'une fonction constructeur d'objet

- Les deux exemples précédents sont limités à plusieurs niveaux, puisqu'ils ne permettent de créer qu'un seul objet.
- Souvent, nous souhaitons créer un type objet, puis l'utiliser pour créer plusieurs objet sur la base de ce type.
- La méthode standard pour créer un "type d'objet" est d'utiliser une fonction de constructeur d'objet.

```
<!DOCTYPE html>
<html><head><meta charset="UTF-8">
<title>Insert title here</title></head><body>
<p id="para1"></p><p id="para2"></p>
<script>
function Etudiant(cne, prenom, nom, age, nomComplet) {
    this.cne = cne;
    this.prenom = prenom;
    this.nom = nom;
    this.age = age;
    this.nomComplet = nomComplet;
}
var moha = new Etudiant("161112222", "Ali", "Moha", 18, function(){
                        return this.prenom + " " + this.nom;});
var toto = new Etudiant("161112222", "Najib", "Toto", 18, function(){
                        return this.nom + " " + this.prenom + " age : " + this.age;});
document.getElementById("para1").innerHTML = "Moha :  " + moha.nomComplet();
document.getElementById("para2").innerHTML = "Toto :  " + toto.nomComplet();
</script>
</body></html>
```

Les objects JavaScript sont Mutables

- Les objects sont mutables: Ils sont adressés par référence et non par valeur.
- Si x est un object, l'instruction : var y = x
  ne céera pas un nouveau objet copie du premier, mais une autre référence vers l'objet x
- Tout changement effectué sur y affectera x également.
- x et y représentent le même objet.

```
<!DOCTYPE html><html><head>
<meta charset="UTF-8">
<title>Insert title here</title></head><body>
<p id="para1"></p>
<script>
var etudiantMoha = {
    cne : 1611442233,
    prenom: "Ali",
    nom : "Moha",
    age : 18,
    nomComplet : function() {
       return this.prenom + " " + this.nom;
    }
};
var etudiantAli = etudiantMoha;
etudiantAli.age = 25;

document.getElementById("para1").innerHTML = etudiantMoha.nom +
" est agé de " + etudiantMoha.age + " ans";
</script></body></html>
```

Proprietés JavaScript

- Les Propriétés sont des variables associées à un objet JavaScript.
- Un objet JavaScript est une collection de propriétés non ordonnées.
- Il est possible d'effectuer les opérations suivantes:
  - Ajouter une nouvelles propriété,
  - Supprimer une propriété existante,
  - Modifier la valeur d'une propriété
  - Rendre une propriété à lecture unique,
  - Etc…

Accès aux propriétés d'un objet JavaScript

- Pour accéder à une propriété d'un objet, nous pouvons utiliser trois syntaxe différentes:
- *nomObjet.propriete* // etudiant.age
- *nomObject*["*propriete*"] // etudiant["age"]
- *nomObject*[*expression*] // var p = "age"; etudiant[p]
  expression doit s'évaluer en une propriété

La boucle for...in

- L'instruction JavaScript **for...in** permet d'efectuer une boucle sur toutes les propriétés d'un objet.
- La Syntaxe est:

```
for (<variable> in <objet>) {
    <instruction>
}
```

- Les instructions du corps de la boucle vont s'exécuter pour chaque propriété de l'objet.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head><body><p id="para1"></p>
<script>
var etudiant = {
cne : 1611442233,
    prenom: "Ali",
    nom : "Moha",
    age : 18,
    nomComplet : function() {
       return this.prenom + " " + this.nom;
    }
};
var str="";
for(var p in etudiant){
str += p + " = " + etudiant[p] + "<br>";
}
document.getElementById("para1").innerHTML = str;
</script></body></html>
```

Ajout de nouvelles Propriétés

- Il est possible d'ajouter une nouvelle propriété à un objet existant en lui affectant une valeur.
- La syntaxe est:

  <objet>.<nouvellePropriété> = <valeur>;

- Exemple:

  etudiant.taille=185;

Suppression d'une propriété d'un objet

- Il est possible de supprimer une propriété à un objet existant.
- La syntaxe est:

  Delete <objet>.<propriété>;

- Exemple:

  delete etudiant.age;

Delete supprime à la fois la valeur et la propriété

Attributs d'une Propriété

- Une propriété a un nom et une valeur (value).
- value fait partie des attributs d'une propriété.
- Les autres attributs sont : enumerable, configurable, et writable.
- Les attributs definissent comment la propriété est accessible (est elle accessible en lecture?, en écriture?)
- En JavaScript, tous les attributs d'une propriété peuvent être lues, mais seule l'attribut value peut être modifiée (et uniquement lorsque la propriété est "writable").

Méthodes d'un objet JavaScript

- Les méthodes JavaScript sont les actions qui peuvent être effectuées sur des objets.
- Une méthode JavaScript est une propriété contenant une définition de fonction.
- Vous créez une méthode d'objet avec la syntaxe suivante:

  methodName : function() { code lines }

- Vous accédez à une méthode d'objet avec la syntaxe suivante:

  objectName.methodName()

Ajout de nouvelles Méthodes

- L'ajout de nouvelles méthodes à un objet se fait à l'intérieur de la fonction constructeur:
- Exemple:

```
function person(firstName, lastName, age, eyeColor) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.age = age;
    this.eyeColor = eyeColor;
    this.changeName = function (name) {
        this.lastName = name;
    };
}
```

Prototypes d'objets JavaScript

- Chaque objet JavaScript a un prototype.
- Le prototype d'un objet est à son tour un objet.
- Tous les objets JavaScript héritent leurs propriétés et méthodes à partir de leur prototype.

Prototypes JavaScript

- Les objets créés en utilisant un objet literal, ou avec new Object(), héritent à partir du prototype appelé Object.prototype.
- Les objects créés avec new Date() héritent de Date.prototype.
- Object.prototype est au sommet de la chaine des prototypes.
- Tous les objet JavaScript (Date, Array, RegExp, Function, ...) héritent de Object.prototype.

Création d'un prototype

- La méthode standard pour créer un prototype d'objet est d'utiliser un
- Example

```
function Person(first, last, age, eyecolor) {
    this.firstName = first;
    this.lastName = last;
    this.age = age;
    this.eyeColor = eyecolor;
 }
```

Avec une fonction constructeur, vous pouvez utiliser le mot-clé new pour créer de nouveaux objets du même prototype.

- Example
  - ✓var myFather = new Person("John", "Doe", 50, "blue");
  - ✓var myMother = new Person("Sally", "Rally", 48, "green");

- La fonction constructeur est le prototype des objets Person.
- Il est considéré comme une bonne pratique de commencer le nom d'une fonction constructeur avec une majuscule.

Ajout de propriétés et méthodes aux objets

- Parfois, vous voulez ajouter de nouvelles propriétés (ou méthodes) à un objet existant.
- Parfois, vous voulez ajouter de nouvelles propriétés (ou méthodes) à tous les objets existants d'un type donné.
- Parfois, vous voulez ajouter de nouvelles propriétés (ou méthodes) à un prototype d'objet.

- **Ajout d'une propriété à un objet**
  - L'ajout d'une nouvelle propriété à un objet existant est facile:
- Exemple

```
myFather.nationality = "English";
```

- La propriété sera ajouté à myFather. Pas à myMother. Non à tous les autres objets de Person.
- Ajout d'une méthode à un objet
  - L'Ajout d'une nouvelle méthode à un objet existant est également facile:
- Exemple

```
myFather.name = function () {
    return this.firstName + " " + this.lastName;
};
```

  - La méthode sera ajouté à myFather. Pas à myMother. Non à tous les autres objets de Person.

Ajout de propriétés à un Prototype

- Vous ne pouvez pas ajouter une nouvelle propriété à un prototype de la même façon que vous ajoutez une nouvelle propriété à un objet existant, parce que le prototype n'est pas un objet existant.
- Pour ajouter une nouvelle propriété à un constructeur, vous devez l'ajouter à la fonction constructeur:
- Exemple

```
function Person(first, last, age, eyecolor) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eyecolor;
  this.nationality = "English"
}
```

- Les propriétés d'un prototype peuvent avoir des valeurs de prototype (qui sont valeurs par défaut pour les objets créés avec le prototype).

Ajout de méthodes à un prototype

- Une fonction constructeur peut également définir des méthodes:
- Example

```
function Person(first, last, age, eyecolor) {
    this.firstName = first;
    this.lastName = last;
    this.age = age;
    this.eyeColor = eyecolor;
    this.name = function() {return this.firstName + " " + this.lastName;};
}
```

Utilisation de la propriété prototype

- La propriété prototype vous permet d'ajouter de nouvelles propriétés à un prototype existant:
- Example

```
function Person(first, last, age, eyecolor) {
        this.firstName = first;
        this.lastName = last;
        this.age = age;
        this.eyeColor = eyecolor;
    }
    Person.prototype.nationality = "English";
```

- **Ne modifiez que vos propres prototypes. Ne jamais modifier les prototypes d'objets JavaScript standard.**

- La propriété prototype permet également d'ajouter de nouvelles méthodes pour un prototype existant:
- Example

```
function Person(first, last, age, eyecolor) {
    this.firstName = first;
    this.lastName = last;
    this.age = age;
    this.eyeColor = eyecolor;
}
Person.prototype.name = function() {
    return this.firstName + " " + this.lastName;
};
```

ECMA 6 : Classes

```
class Employee {

        constructor(hireDate, monthlySalary) {
                this.hireDate = hireDate;
                this.monthlySalary = monthlySalary;
        }

        getMonthlySalary() {
                return `I make ${this.monthlySalary} each month.`
        }
}
```

```
class Person extends Employee {
	constructor(fName, lName, profession, age, hireDate, monthlySalary) {
		// Properties
		super(hireDate, monthlySalary);
		this.fName = fName;
		this.lName = lName;
		this.profession = profession;
		this.age = age;
	}
	// Methods
	saySomething() {
		return `What's up? I am a ${this.age}-yr. old ${this.profession} and
			was hired on ${this.hireDate}.`;
	}
}
```

```
var me = new Person("Gabriel", "Cánepa", "developer", "33", "2016-03- 22", 1575);
console.log(me.saySomething());
console.log(me.getMonthlySalary());
```

The let keyword

```
function guessName() {
      var sayName = "John Doe";
      if (sayName === "John Doe") {
            let msg = `the name is ${sayName}`
            console.log(`Inside the block ${msg}.`)
      }
      console.log(`Outside the block ${msg}.`)
}
```

Constructeurs JavaScript Prédéfinis

- JavaScript possède des constructeurs prédéfinis pour la création des objets natifs. Ce sont:
- var obj = new Object(); // crée un nouveau objet de type Object
  var str = new String(); // crée un nouveau objet de type String
  var nbr = new Number(); // crée un nouveau objet de type Number
  var bool= new Boolean() // crée un nouveau objet de type Boolean
  var tab = new Array(); // crée un nouveau objet de type Array
  var re = new RegExp(); // crée un nouveau objet de type RegExp
  var fn = new Function(); // crée un nouveau objet de type Function
  var dte = new Date(); // crée un nouveau objet de type Date
- Remarque: l'objet Math ne figure pas dans la liste, car il ne peut pas être utilisé avec new.

```
<!DOCTYPE html>
<html><head><meta charset="UTF-8"><title>Insert title here</title>
</head><body><p id="para1"></p>
<script>
var obj = new Object();     // crée un nouveau objet de type Object
var str = new String();     // crée un nouveau objet de type String
var nbr = new Number();     // crée un nouveau objet de type Number
var bool= new Boolean()     // crée un nouveau objet de type Boolean
var tab = new Array();      // crée un nouveau objet de type Array
var re = new RegExp();     // crée un nouveau objet de type RegExp
var fn = new Function();   // crée un nouveau objet de type Function
var dte = new Date();       // crée un nouveau objet de type Date
document.getElementById("para1").innerHTML =
"obj: " + typeof obj + "<br>" +"str: " + typeof str + "<br>" +
"nbr: " + typeof nbr + "<br>" +"bool: " + typeof bool + "<br>" +
"tab: " + typeof tab + "<br>" +"re: " + typeof re + "<br>" +
"fn: " + typeof fn + "<br>" +"dte: " + typeof dte + "<br>";
</script>
<p>Il n'ya pas besoin d'utiliser String(), Number(), Boolean(), Array(), et
RegExp()</p></body></html>
```

JavaScript
Tableaux

Qu'est ce que et comment créer un tableau?

- **Qu'est ce qu'un tableau?**
  - Un tableau est une variable qui peut contenir plusieurs valeurs, et qui permet d'accéder à ces valeurs en utilisant un indice.
- **Comment Créer un tableau?**
  - Utiliser un tableau literal.
    - Syntaxe:
      - var *array_name* = [*item1*, *item2*, ...];
    - Exemple:
      - var arbres = ["olivier", "pommier", "cerisier"];
  - Utiliser new Array()
    - Syntaxe:
      - var array_name = new Array(item1, item2, …);
    - Exemple:
      - Var arbres = new Array("olivier", "pommier", "cerisier");

- Remarques:
  - La première syntaxe est plus simple et offre de meilleures performances lors de l'exécution.
  - La syntaxe "new Array" complique le code et peut produire des résultats inattendus.
  - Il est donc recommandé d'utiliser la première syntaxe.

Accéder aux éléments d'un tableau

- La synatxe est:
  - <nomTableau>[i]
- Exemple:
  - var unarbre = arbres[0]; (accède au premier élément du tableau; ="olivier"))
  - arbres[1] = "Amandier"; (modifie le 2ème élément du tableau)
- Remarque:
  - Contrairement aux autres langages tels que C et Java, un tableau JavaScript peut contenir des éléments de type différent.
  - Exemple:

```
var arbres=["olivier", "pommier", "amandier"];
var tab=[10, "abcd", Date.now, somme, arbres]; //Contient des éléments de  # types.
```

Les tableaux sont des objets

- Les tableaux sont un type spécial d'objets.
- L'opérateur **typeof** retourne "object" pour une variable tableau.
- Un tableau a donc des propriétés et méthodes.
- Propriété length:
  - La propriété length retourne la longueur du tableau, c'est-à-dire le nombre de ses éléments.
  - **var arbres=["olivier", "pommier", "amandier"];**
  - arbres.length → 3
- Remarque:
  - var tab1=[]; //Crée un tableau vide
  - var tab2 = new Array(); Crée un tableau vide
  - tab1.length = tab2.length = 0;
  - tab1[1]=10; ajoute un élément au tableau

Comment déterminer si une variable est un tableau?

- L'opérateur typeof ne peut pas être utiliser pour déterminer si une variable est de type tableau car il retourne object.
- Utiliser la méthode isArray ou l'opérateur instanceof
  - var tab = [10, 20, 30, 40, 50];
  - typeof tab; → object
  - Array.isArray(tab); → true
  - tab instanceof Array; → true

ARRAY LENGTH

```
var colors = ['White', 'Red', 'Green'];
//length of an array
console.log(colors.length); // 3

// You can also assign to the length property.
colors.length = 2;
console.log(colors); // (2) ["White", "Red"]
- Green has been removed

colors.length = 0;
console.log(colors); // []

colors.length = 3;
console.log(colors); // (3) [empty × 3]
```

ITERATING OVER ARRAYS

A common operation is to iterate over the values of an array, processing each one in some way. Here is simplest format:

```
var colors = ['White', 'Red', 'Green'];
for (var n = 0; n < colors.length; n++) {
  console.log(colors[n]);
}

// Output
// "White"
// "Red"
// "Green"
```

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<script>
var tab =[10, 20, 30, 40, 50];
document.write("Longueur du tableau : " + tab.length +"<br>");
document.write("type du tableau :" + typeof tab + "<br>");
document.write("tab est il un tableau ? " + Array.isArray(tab) + "<br>");
document.write("tab est instance de Array ? ")
document.write(tab instanceof Array);
document.write("tab est instance de Array : " + (tab instanceof Array))
</script>

</body>
</html>
```

Méthodes des tableaux

- La force des tableaux JavaScript réside dans les méthodes qu'ils peuvent exécuter.
- **Méthode toString() : Convertir un tableau en une String**
  - La méthode **toString()** convertit un tableau en une String formé par la suite de ses éléments séparés par une virgule.
  - Example
  - var arbres = ["Olivier", "Pommier", "Amandier"];
    tab.toString(); → Olivier,Pommier,Amandier
- Méthode **join()** : Joindre les éléments d'un tableau pour former une string.
  - Join() se comporte comme toString(), mais en plus elle peut specifier le séparateur
  - Exemple
  - var arbres = ["Olivier", "Pommier", "Amandier"];
    tab.join(" * "); → Olivier*Pommier*Amandier

Méthodes des tableaux (suite)

- Méthode push : empiler un ou plusieurs élément dans le tableau.
  - La syntaxe est :
    - nomTableau.push[elt1, elt2, …]
  - Exemple:
    - var arbres = ["Olivier", "Pommier", "Amandier"]
    - Arbres.push("Oranger", "Noyer");
      - →arbres = ["Olivier", "Pommier", "Amandier", "Oranger", "Noyer"]
- Méthode pop : dépiler un élément (=supprimer le dernier élément)
  - La syntaxe est :
    - nomTableau.pop();
  - Exemple:
    - var arbres = ["Olivier", "Pommier", "Amandier"]
    - arbres.pop(); →arbres = ["Olivier", "Pommier"]

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<p id="para1"></p>
<p id="para2"></p>
<p id="para3"></p>
<script>
var arbres=["Olivier", "Pommier", "Amandier"];
arbres.push("Oranger", "Noyer");
document.getElementById("para1").innerHTML = arbres;
arbres.pop();
document.getElementById("para2").innerHTML = arbres;
document.getElementById("para3").innerHTML = "Nouvelle longueur du tableau :
" + arbres.length;
</script>
</body>
</html>
```

Méthodes des tableaux (suite)

- Méthode shift : décaler les éléments vers la gauche (=supprimer le 1er élément)
  - La syntaxe est :
    - nomTableau.shift()
  - Exemple:
    - var arbres = ["Olivier", "Pommier", "Amandier"]
    - arbres.shift();
      - →arbres = ["Pommier", "Amandier"]
- Méthode unshift : insére un ou plusieurs éléments au début du tableau et décale les autres vers la droite.
  - La syntaxe est :
    - nomTableau.unshift(element1, element2, ….);
  - Exemple:
    - var arbres = ["Olivier", "Pommier", "Amandier"]
    - arbres.unshift("Bananier"); →arbres = ["Bananier", "Olivier", "Pommier", "Amandier"]

Méthodes des tableaux (suite)

- Méthode delete : supprime un élément du tableau et laisse sa position vide (undefined)
  - La syntaxe est :
    - delete nomTableau [indiceElement];
  - Exemple:
    - var arbres = ["Olivier", "Pommier", "Amandier"]
    - Delete arbres[0];
      - →arbres = [, "Pommier", "Amandier"]
- Méthode splice : supprime et insère de nouveaux éléments à une position donnée.
  - La syntaxe est :
    - nomTableau.splice(position, nombreElementASupprimer, element1, element2, ….);
  - Exemple:
    - var arbres = ["Olivier", "Pommier", "Amandier"]
    - arbres.splice(1, 0, "Bananier"); →arbres = ["Olivier", "Bananier", "Pommier", "Amandier"]

Méthodes des tableaux (suite)

- Méthode sort : trier un tableau par ordre alphabétique
  - La syntaxe est :
    - nomTableau.sort();
  - Exemple1:
    - var arbres = ["Olivier", "Pommier", "Amandier"]
    - arbres.sort();
      - →arbres = ["Amandier", "Olivier" ,"Pommier"]
  - Exemple2: sort() considère les nombres comme des Strings
    - var montant = [100, 20, 5, 15]
    - montant.sort();
      - →montant = [100, 15, 20, 5] les données sont triés par ordre alphabétique
- Méthode reverse : inverse l'ordre des éléments.
  - La syntaxe est :
    - nomTableau.reverse();
  - Exemple:
    - var arbres = ["Olivier", "Pommier", "Amandier"]
    - arbres.reverse(); →arbres = ["Amandier", "Pommier", "Olivier"]

Méthodes des tableaux (suite)

- Méthode sort : trier un tableau de nombres en utilisant une fonction de comparaison
- Pour trier un tableau de nombres il faut passer à la méthode sort une fonction de comparaison comme argument.
  - Exemple1: tri par ordre croissant
    - var montants = [50, 20, 15, 30]
    - montants.sort(function(a,b){return a-b}); → [15, 20, 30, 50]
    - montants.sort(function(a,b){return a>b}); → produit la même chose
  - Exemple2: tri par ordre décroissant
    - var montants = [50, 20, 15, 30]
    - montants.sort(function(a,b){return b-a}); → [50, 30, 20, 15]
    - montants.sort(function(a,b){return b>a}); → produit la même chose
- Comment sort utilise la fonction de comparaison : function(a,b) {return a-b}
  - Pour trier deux éléments a et b, sort commence par évaluer la fonction pour les arguments a et b.
    en calculant a-b. si le résultat est <0, a est placé avant b, si le résultat est >0, a est placé après b. si le résultat est nul, l'ordre n'est pas changé.

Méthodes des tableaux (suite)

- Méthode concat : fusion des éléments de deux tableaux
  - La syntaxe est :
    - nonTableau1.concat(nomTableau2);
  - Exemple:
    - var arbres = ["Olivier", "Pommier", "Amandier"]
    - var plantes = ["Brocolli", "Epinard", "Chou"]
    - arbres.concat(plantes];
      - →["Olivier", "Pommier", "Amandier", "Brocolli", "Epinard", "Chou"]
    - Les tableaux arbres et plantes ne sont pas modifiés.
- Méthode slice : extrait une partie du tableau
  - La syntaxe est :
    - nomTableau.slice(position1, position2); //Extrait les élément de position1 à position2-1
  - Exemple:
    - var x = ["a", "b", "c", "d", "e"];
    - var y = x.slice(1,4); extraire x[1] à x[3], x[4] n'est pas inclus.
    - y→["b", "c", "d"] , x n'est pas modifié
    - var z = x.slice(2); extrait les éléments de x[2] jusqu'à la fin. z → ["c", "d", "e"];

JavaScript
Objet Date

JAVASCRIPT DATE OBJECT

JavaScript date object is used to create dates and times. The Date object range is -100,000,000 days to 100,000,000 days relative to 01 January, 1970 UTC.

These are the ways to create a date object in JavaScript :

a) new Date().

[for example today = new Date()].

b) new Date(milliseconds).

[for example inauguration_day = new Date("August 15, 1997 10:05:00")].

c) new Date(dateString).

[for example inauguration_day = new Date(97,8,15)].

d)new Date(year, month, day, hours, minutes, seconds, milliseconds).

[for example inauguration_day = new Date(97,8,15,10,05,0)].

JAVASCRIPT DATE OBJECT – METHODS (1/5)

```
date.getDate(); // Returns the day of the month (from 1-31).
date.getDay(); // Returns the day of the week (from 0-6).
date.getFullYear(); // Returns the year (four digits).
date.getHours(); // Returns the hour (from 0-23).
date.getMilliseconds(); // Returns the milliseconds (from 0-999).
date.getMinutes(); // Returns the minutes (from 0-59).
date.getMonth(); // Returns the month (from 0-11).
date.getSeconds(); // Returns the seconds (from 0-59).
date.getTime(); // Returns the number of milliseconds since midnight Jan
                //1, 1970.
date.getTimezoneOffset(); // Returns the time difference between UTC
                          //time and local time, in minutes.
```

JAVASCRIPT DATE OBJECT – METHODS (2/5)

```
date.getUTCDate(); // Returns the day of the month, according to
                   //universal time (from 1-31).
date.getUTCDay(); // Returns the day of the week, according to
                  //universal time (from 0-6).
date.getUTCFullYear(); // Returns the year, according to universal
                       //time (four digits).
date.getUTCHours(); // Returns the hour, according to universal time
                       //(from 0-23).
date.getUTCMilliseconds(); // Returns the milliseconds, according to
                           //universal time (from 0-999).
date.getUTCMinutes(); // Returns the minutes, according to universal
                       //time (from 0-59).
date.getUTCMonth(); // Returns the month, according to universal
                       //time (from 0-11).
date.getUTCSeconds(); // Returns the seconds, according to universal
                       //time (from 0-59).
```

JAVASCRIPT DATE OBJECT – METHODS (3/5)

```
date.setDate(day); // Sets the day of the month of a date object.
date.setFullYear(year, month, day); // Sets the year (four digits) of a
                    //date object.
date.setHours(hour, min, sec, millisec); // Sets the hour of a date
                    //object.
date.setMilliseconds(millisec); // Sets the milliseconds of a date
                    //object.
date.setMinutes(min, sec, millisec); // Set the minutes of a date
                    //object.
date.setMonth(month, day); // Sets the month of a date object.
date.setSeconds(sec, millisec); // Sets the seconds of a date object.
date.setTime(millisec); // Sets a date and time by adding or
                    //subtracting a specified number of milliseconds
                    //to/from midnight January 1, 1970.
```

JAVASCRIPT DATE OBJECT – METHODS (4/5)

```
date.setUTCDate(day); // Sets the day of the month of a date object,
            //according to universal time.
date.setUTCFullYear(year, month, day); // Sets the year of a date object,
            //according to universal time (four digits).
date.setUTCHours(hour, min, sec, millisec); // Sets the hour of a date
            //object, according to universal time.
date.setUTCMilliseconds(millisec); // Sets the milliseconds of a date
            //object, according to universal time.
date.setUTCMinutes(min, sec, millisec); // Set the minutes of a date
            //object, according to universal time.
date.setUTCMonth(month, day); // Sets the month of a date object,
            //according to universal time.
date.setUTCSeconds(sec, millisec); // Set the seconds of a date object,
            //according to universal time.
```

JAVASCRIPT DATE OBJECT – METHODS (5/5)

date.toDateString(); // Converts the date portion of a Date object into a readable string.

date.toISOString(); // Returns the date as a string, using the ISO standard.

date.toJSON(); // Returns the date as a string, formatted as a JSON date.

date.toLocaleDateString(); // Returns the date portion of a Date object as a string, using locale conventions.

date.toLocaleTimeString(); // Returns the time portion of a Date object as a string, using locale conventions.

date.toLocaleString(); // Converts a Date object to a string, using locale conventions.

date.toString(); // Converts a Date object to a string.

date.toTimeString(); // Convert the time portion of a Date object to a string.

date.toUTCString(); // Converts a Date object to a string, according to universal time.

date.valueOf(); // Returns the primitive value of a Date object.