JavaScript

La programmation objet

Bases

Portée de variables

Une des dificultés de JS est liée à la portée des variables.

Celles-ci sont soit :

- Globales
- relatives au contexte où elles sont déclarées

Les tableaux

Les tableaux JavaScript ne sont pas vraiment des tableaux au sens classique

- Leur contenu peut être arbitraire
- Ils n'ont pas de dimension

Itérateurs

Un itérateur est un objet qui permet de parcourir une collection. C'est l'équivalent d'un for ... in ...

Tout objet possède un itérateur accessible par la propriété __iterator__

```
var obj = {nom:"Ray", username:"F451", dept:"Mars"};

var it = Iterator(obj);

try {
 while (true) { document.write(it.next() + "<br>\n"); }
} catch (err if err instanceof StopIteration) {
 document.write("End of record.<br>\n");
} catch (err) {
 document.write("Unknown error: " + err.description + "<BR>\n");
}
```

Générateurs

Un générateur est une source de données, fonctionnant généralement en relation avec un itérateur. Sa fonction est de rendre un valeur à chaque appel, par l'instruction `yield`.

La particularité du générateur est de mémoriser son contexte d'exécution d'un appel à l'autre. On s'en sert donc précisément pour délivrer des séries (ex. Fibonacci)

```
function fib() {
  var i = 0, j = 1;
  while (true) { yield i;  var t = i;  i = j;  j += t; }
}

var g = fib();
for (var i = 0; i < 10; i++) { document.write(g.next() + "<br>\n"); }
```

Arguments

Une fonction peut gérer ses arguments de manière implicite avec l'objet `arguments` :

```
var perimetre = function () {
   if ( arguments.length == 1) return arguments[0]* 4;
   else if (arguments.length == 2) return 2 * (arguments[0] + arguments[1]);
   else {
      for (i = 0; i < arguments.length; i++) p += arguments[i];
      return p;
   }
}
```

Arguments

JavaScript sait aussi maintenant gérer les nombres d'arguments variables :

```
var perimetre = function (...x, y) {
   [ ... ]
}
```

Expressions régulières

Déclaration d'expressions régulières :

```
var reg = /\d{2}\.?\w+/igmy
var reg = new RegExp("\d{2}\.?\w+","i","g","m","y");
```

Globalement, les regexp s'utilisent dans 3 contextes :

```
var match = reg.test(str);
// teste le pattern-matching
var match = reg.exec(str);
 // retourne un tableau de correspondance du pattern-matching
var str = str.replace(reg, expression);
// la méthode replace de String accepte les regexp
```

Exceptions

JavaScript gère les comme les autres langages try , catch , throw :

```
var div = function (x, y) {
   if ( y == 0) {
       throw : { name : 'DivideByZeroError', message : 'Division par zéro' }
   }
   [ ... ]
}

try { div(x, y) } catch (e) { log(e.message)}
```

AJAX

`XMLHttpRequest` est un objet non standard créé par Microsoft comme composant ActiveX et que tous les fabricants de navigateurs ont adopté. Son principal intérêt est de pouvoir faire de la communication asynchrone avec le serveur.

Il est possible de lancer des requêtes cross-server, en respectant les protections qui empêchent les attaques dites XSS.

Une requête AJAX comprend deux volets :

- Le lancement de la requête
- La gestion de la réponse

AJAX : la requête

```
if (window.XMLHttpRequest) { // Mozilla, Safari,...
        httpRequest = new XMLHttpRequest();
        if (httpRequest.overrideMimeType) {
           httpRequest.overrideMimeType('text/xml');
           // Voir la note ci-dessous à propos de cette ligne
        }
     }
     else if (window.ActiveXObject) { // IE
        try {
           httpRequest = new ActiveXObject("Msxml2.XMLHTTP");
        }
        catch (e) {
           try {
              httpRequest = new ActiveXObject
("Microsoft.XMLHTTP");
           }
           catch (e) {}
        }
     }

     if (!httpRequest) {
        alert('Abandon :( Impossible de créer une instance
XMLHTTP');
        return false;
     }
```

Initialisation de l'objet `XMLHttpRequest`. IE utilise son propre objet pour des raisons historiques : `ActiveXObject`.

On vérifie que le navigateur sait gérer les requêtes.

AJAX : l'envoi

```
httpRequest.onreadystatechange = function() {
   qqchose(httpRequest);
};
httpRequest.open('GET', url, true);
httpRequest.send(null);
```

onreadystatechange permet de lancer la requête en mode asynchrone

L'objet peut également gérer d'autres évènements comme :

onprogress
onloadstart
onload
onloadend
onerror
ontimeout
onabort

AJAX : la réponse

```
function qqchose(httpRequest) {

    if (httpRequest.readyState == 4) {
        if (httpRequest.status == 200) {
            alert(httpRequest.responseText);
        } else {
            alert('Echec...');
        }
    }

}
```

`readystate` change de valeur à chaque changement d'état de la requête. 4 indique que la réponse est complète.

Les états :
0 = UNSENT
1 = OPENED
2 = HEADERS RECEIVED
3 = LOADING
4 = DONE

JSON

Présentation

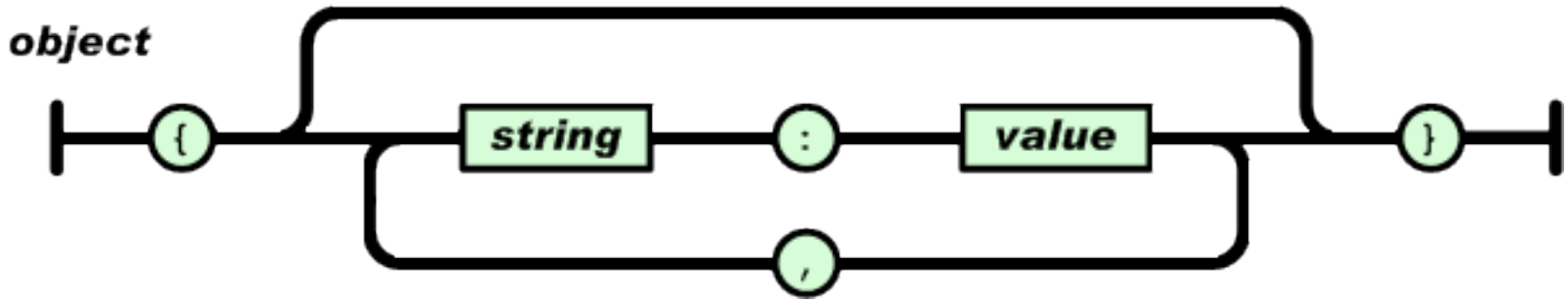
JSON (JavaScript Object Notation) est un format léger d'échange de données. Il est lisible pour les humains et facile à analyser pour les machines.

Il est complétement indépendant de tout langage de programmation, mais ses conventions sont proches de langages comme C, PHP, Python, Perl, etc.

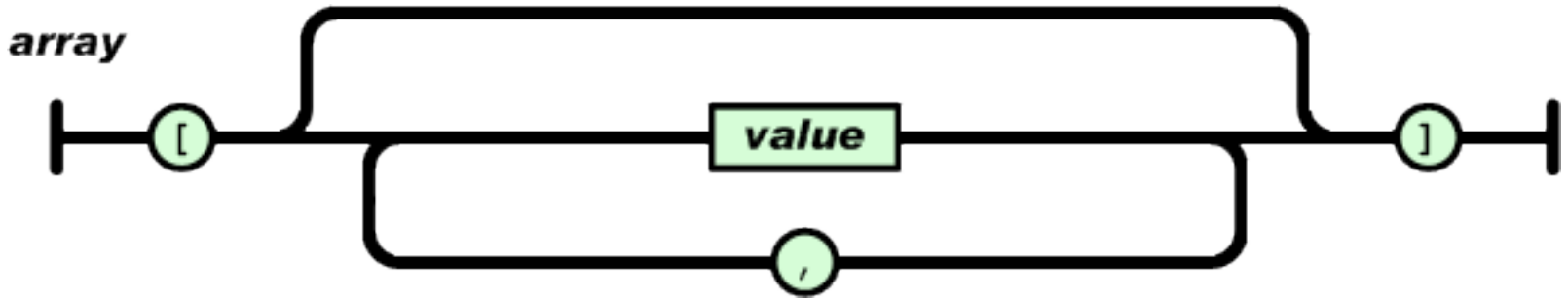
JSON est bâtis sur deux types de structures :

- .des collections de paires clef/valeur (comme des struct, des tableaux de hashage, etc.)
- des listes non ordonnées de valeurs (des tableaux)

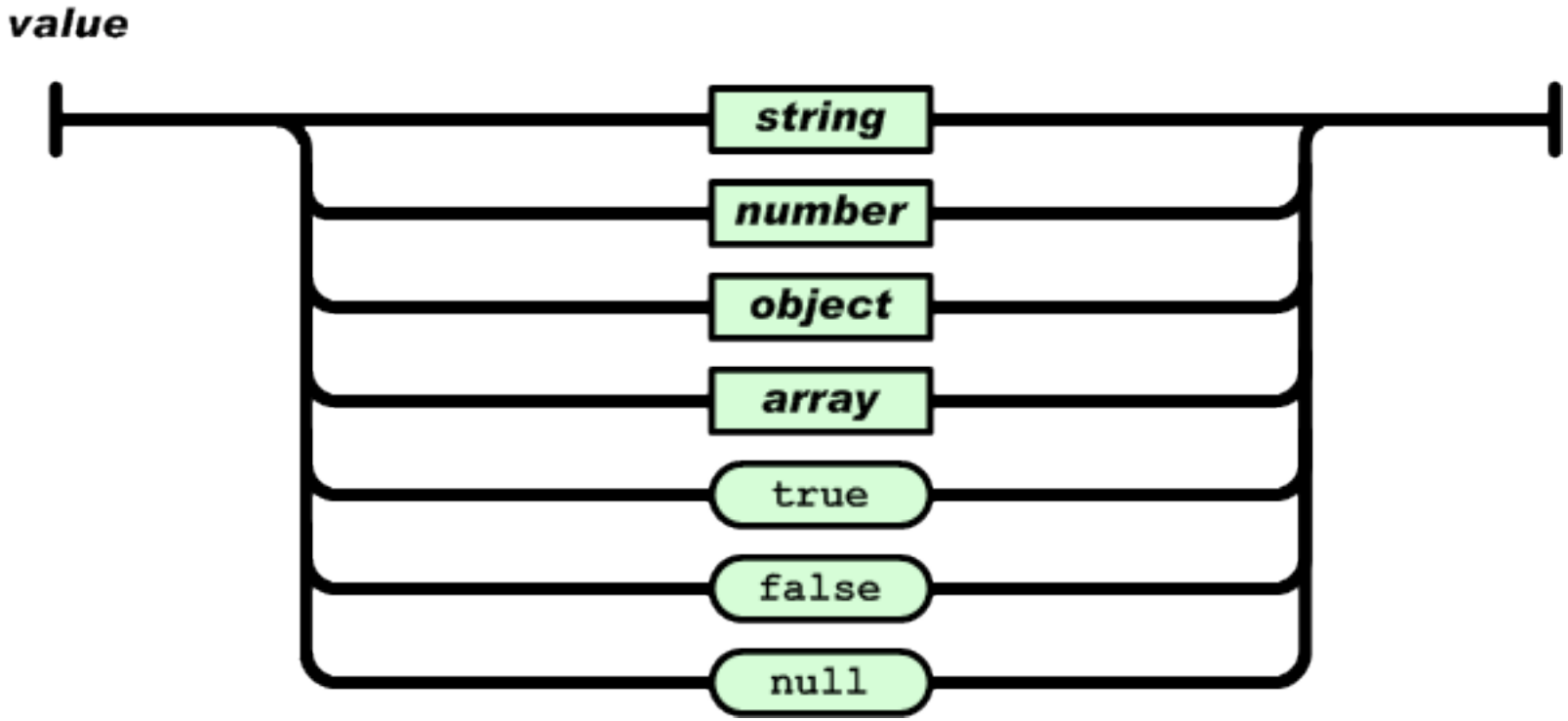
Objets



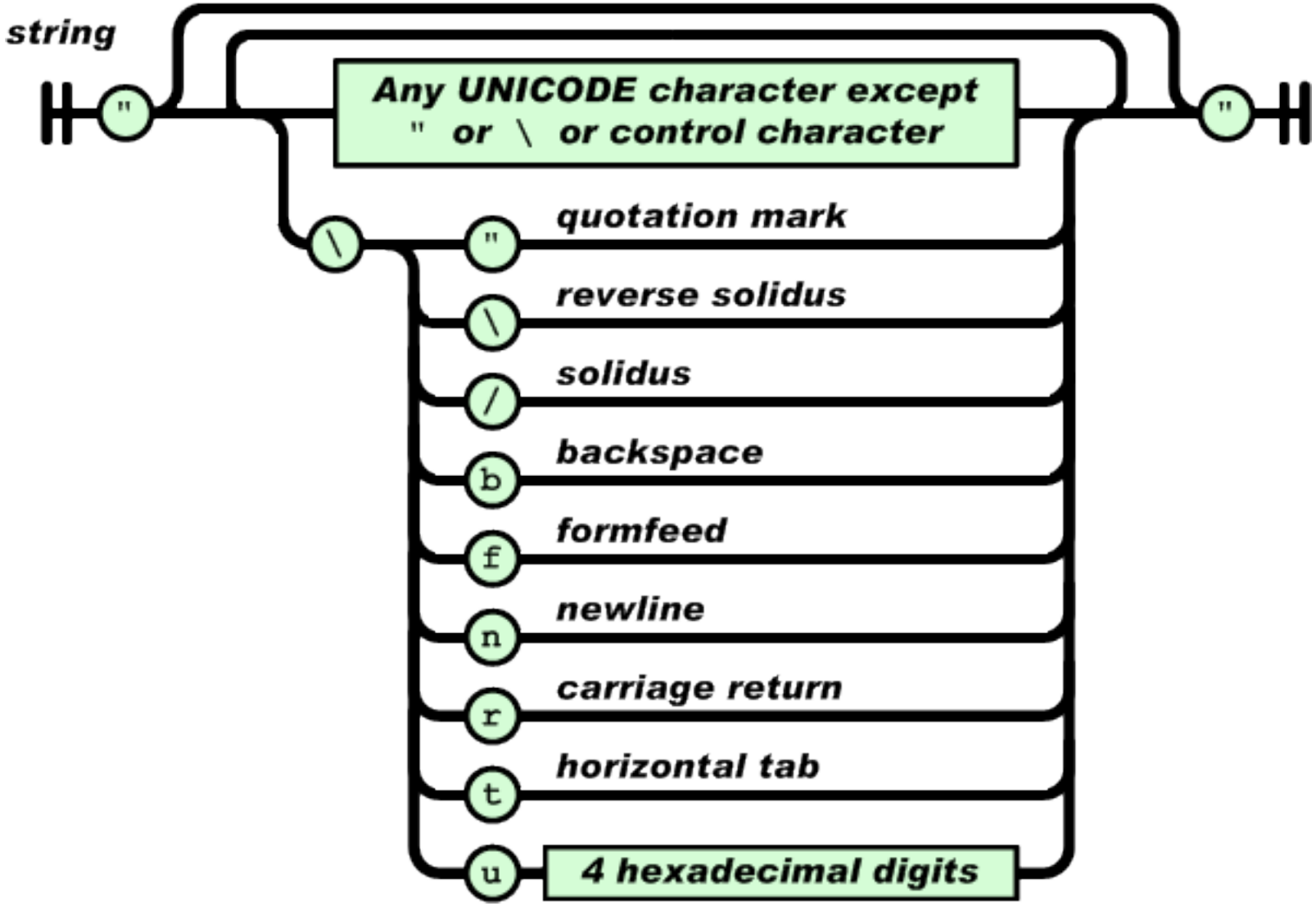
Tableaux



Valeurs



Chaînes


string
"
Any UNICODE character except
" or \ or control character
"
\
"
quotation mark
\
reverse solidus
/
solidus
b
backspace
f
formfeed
n
newline
r
carriage return
t
horizontal tab
u
4 hexadecimal digits

Exemple

```
{ ‘voiture’ :
   {  ‘roues’ : 4,
      ‘propulsion’ : [‘essence’, ‘diesel’, ‘électricité’],
      ‘ parties’ : [
          ‘chassis’ : ... ,
           ‘moteur’ : { .... },
           ‘alimentation’ : ‘ batteries’
                    ]
   },
  ‘vélo’ :
   { ‘roues’ : 2,
     ‘parties’ : [
           ‘moteur’ : false,
           ‘alimentation’ : ‘dynamo’
}
```

Production

Des données au format JSON peuvent être très facilement produites dans des nombreux langages de programmation (et même manuellement).

PHP possède deux fonctions de conversion `json_encode` et `json_decode` qui permettent de transformer un objet en JSON.

Python inclut une bibliothèque `json`.

Ruby = `JSON.decode` & `JSON.encode`

Ceci permet de sérialiser très facilement des objets ou de produire des flux analysables en JavaScript via des requêtes AJAX

POO en JS

Les objets

Création

La manière la plus simple de déclarer un objet en JavaScript : utiliser JSON

```
var x = {
   'hauteur' : 30,
   'largeur' : 50,
   'aire' : function () { return this.x * this.y)
   }
```

Copie, Clonage

L'affectation (=) se fait toujours en créant une référence vers la variable cible.

Ainsi :

```
var x = 25;
var y = x;
```

indique que `y` est un 'alias' de `x`.

Pour créer une nouvelle variable, on utilise la fonction `clone`

Prototypes

Les objets en JavaScript n'ont pas de classe. JS est un langage à prototype.

chaque objet créé peut constituer un prototype pur la création d'autres objets, qui 'héritent' des propriétés de leur 'ancêtre'.

Chaque objet JavaScript a donc des capacités de prototypage, que l'on utilise via la fonction `prototype` (ou `fn`)

```
var Carre.prototype.perimetre = function (cote)
    {return 4 * cote}
```

Réflection

On peut très facilement trouver le type d'un objet grâce à l'opérateur typeof

```
if (typeof x !== 'function') { ... }
```

Enumération

La structure de contrôle `for ( ... in ...)` permet de traverser un objet et de manipuler ses composants

```
for (slot in my_object) {
   if (typeof myobject[name] == 'number') {
      ...
   }
   ...
```

N.B. On utilise ici la notation « tableau » pour les objets, la notation traditionnelle étant ambiguë. Les deux notations sont équivalentes en JS

Effacer

Effacer un objet ou la propriété d'un objet se fait par l'opérateur `delete`.

```
delete x.hauteur
```

Les fonctions

Fonctions

Les fonctions sont des objets et sont donc la deuxième manière native des créer des objets en JS.

Les fonctions, en tant qu'objets :

- sont liées à : Function.prototype (lié à Object.prototype)
- ont deux propriétés particulières : le contexte d'exécution et le code de la fonction
- peuvent être traitées comme tout autre objet et donc être :
  - des valeurs pour des variables
  - des arguments de fonctions
  - des valeurs de retour
  - etc.

Créer un objet

On peut très facilement créer un objet via une fonction :

```
var Rectangle = function (x, y) {
   this.largeur = x;
   this.longueur = y;
   this.aire = function () {
      return this.x * this.y
   }
}
```

Cet objet étant un prototype, il peut engendrer de nouveaux objets :

```
var r1 = new Rectangle (10, 20);
```

this

`this` est la référence à l'objet lui-même.
Mais pour des raisons de portée, l'emploi de `this` est très problématique

```
function someFunc (p1) {
   var this.x = p1;
   something.on("click", function() {
      console.log(this.x);
   });
};
```

Faux : les contextes de this sont différents

`call` est `apply` permettent d'exécuter des fonctions dans un contexte donné.

Utiliser prototype

Il est bien sûr très simple d'ajouter de nouvelles fonctionnalités aux objets ainsi créés :

```
Rectangle.prototype.perimetre = function () {
      return 2 * (this.x + this.y);
   }
}
```

Fonctions anonymes

Une fonction anonyme est un objet qui est pas explicitement lié à une variable.

On peut les exécuter grâce à l'opérateur () qui est un évaluateur.

```
(
 function facto (n) {
   if (n == 1) return 1 ;
   else return n* facto(n-1) ;
 }
)(5) ;
```

Fermetures

Dans un langage de programmation, une **fermeture** ou clôture (en anglais, **closure**) est une fonction qui capture des références à des **variables libres** dans l'environnement lexical. Une fermeture est donc créée, entre autres, lorsqu'une fonction est définie dans le corps d'une autre fonction et fait référence à des arguments ou des variables locales à la fonction dans laquelle elle est définie.

```
function ajouteur(nombre) {
  function ajoute(valeur) {
    return valeur + nombre;
  }
  return ajoute;
}
```

nombre est une variable libre dans l'environnement de la fonction **ajoute**

Curryfication

La « curryfication » est une technique directement héritée du lambda-calcul, qui n'accepte des fonctions que d'un seul argument.

Curry a été l'auteur d'un résultat très important, démontrant la possibilité de considérer des fonctions à plusieurs arguments comme des compositions de fonctions un-aires.

La curryfication est donc une technique qui permet de produire des « fonctions partielles », en fixant la valeur d'un des arguments. On appelle cela une application partielle.

Curryfication

```
Function.prototype.curry = function() {
   var fn = this, args = Array.prototype.slice.call(arguments);
   return function() {
     return fn.apply(this, args.concat(
       Array.prototype.slice.call(arguments)));
   };
 };
```

Curryfication

```
Function.prototype.partial = function(){
  var fn = this, args = Array.prototype.slice.call(arguments);
  return function(){
    var arg = 0;
    for ( var i = 0; i < args.length && arg < arguments.length; i++ )
      if ( args[i] === undefined )
        args[i] = arguments[arg++];
    return fn.apply(this, args);
  };
};
```

Les objets

Les objets globaux

- Array
- Boolean
- Date
- Function
- Iterator
- Number
- Object
- RegExp
- String

Remarque: **attr** et **prop** peuvent différer légèrement. **prop** retourne un booléen, alors que **attr** retourne la valeur de l'attribut.

Se méfier aussi que ces fonctions modifient et se basent sur le DOM (pas sur le code HTML)

JS et DOM

Architecture du DOM

- Node
  - CharacterData
  - Attr
  - Element
    - HTMLElement
  - Document
    - HTMLDocument
  - Comment
- NodeList
- NamedNodeMap
- DOMImplementation

Le DOM n'est pas un langage programmation. C'est un ensemble d'interfaces qui modélise l'accès aux document et au navigateur. En pratique, le DOM est écrit en JavaScript, mais il est possible de réaliser d'autres implémentations du DOM pour d'autres usages. Les principaux objets du DOM sont :

API du DOM

- element
- window
- document
- event
- style
- range
- selection
- Form
- Table

Les principales interfaces

JavaScript avancé

Tableaux en compréhension

Il es possible désormais de définir des tableaux en compréhension sur le mode de Python ou Ruby :

```
var pairs = [i for (i in range(0, 21)) if (i % 2 == 0)];
```

La fonction `range` est un générateur qui va fournir toute la suite des nombres entiers entre 0 et 20, par exemple

let

let permet de gérer la portée d'un bloc de données ou d'une fonction. C'est à la fois un instruction, une expression et une définition.

1. En tant qu'instruction, let permet de définir une portée lexicale pour un ensemble de variables. Cette portée est limitée au bloc associé à l'instruction.

```
let (x = x+10, y = 12) {
  document.write(x+y + "<br>\n");
}
```

let

`let` permet de gérer la portée d'un bloc de données ou d'une fonction. C'est à la fois un instruction, une expression et une définition.

2. En tant que définition, `let` est une alternative à `var` qui permet la redéfinition de variables dans des contextes imbriqués.

```
function letTest() {
  let x = 31;
  if (true) {
    let x = 71; // variable différente
    alert(x); // 71
  }
  alert(x); // 31
}
```

let

let permet de gérer la portée d'un bloc de données ou d'une fonction. C'est à la fois un instruction, une expression et une définition.

3. En tant qu'expression, let d'associer des variables à une expression unique.

```
document.write ( let (x = x + 10, y = 12) x+y  + "\n");
```

Un bloc implicite est associé à let.

Assignation déstructurante

L'assignation déstructurante permet une notation compréhensive de variables, assimilable à un tableau.

Exemple très simple : `[a, b] = [b, a];`

Mode strict

Le mode d'exécution strict est apparu récemment en JS et déclenche des exceptions dans un certain nombre de cas que JS ne signalait pas avant.

On peut trouver sur le site de MSDN la liste des exceptions.

On l'active très simplement en déclarant use strict au début du script.

```
"use strict";

// Déclenchement d'une erreur.
testvar = 5;
// Toutes les variables doivent être déclarées
```