javascript : les tableaux

L'objet `Array`

en javascript :

- la notion de **tableaux** n'est pas conventionnelle
  par rapport à la plupart des langages

exemple-array.js

L'objet `Array`

en javascript :

- la notion de **tableaux** n'est pas conventionnelle
  par rapport à la plupart des langages
- le type (objet) `Array` est prédéfini

exemple-array.js

L'objet `Array`

en javascript :

- la notion de **tableaux** n'est pas conventionnelle
  par rapport à la plupart des langages
- le type (objet) `Array` est prédéfini
- il permet de représenter des tableaux
  ... mais qui sont aussi des collections (listes, piles, files)

exemple-array.js

L'objet `Array`

en javascript :

- la notion de **tableaux** n'est pas conventionnelle
  par rapport à la plupart des langages
- le type (objet) `Array` est prédéfini
- il permet de représenter des tableaux
  ... mais qui sont aussi des collections (listes, piles, files)
- met à disposition des méthodes

exemple-array.js

L'objet `Array`

en javascript :

- la notion de **tableaux** n'est pas conventionnelle
  par rapport à la plupart des langages
- le type (objet) `Array` est prédéfini
- il permet de représenter des tableaux
  ... mais qui sont aussi des collections (listes, piles, files)
- met à disposition des méthodes
  - `join()`, `reverse()`, `sort()`, `push()`, `pop()`, etc.

exemple-array.js

Ce qui est « classique »

- **création**
  - constructeur `Array()`

```
var tab = new Array(10);    // tableau de 10 éléments
```

  - expression littérale

```
var tabVide = [];
var couleur = ["trefle", "carreau", "coeur", "pique"];
```

- **lecture/écriture des éléments**
  - premier élément : indice 0

```
couleur[0];         // -> "trefle"
tab[0];             // -> undefined
tab[1] = 3;
```

- propriété `length`

```
tab.length;      // -> 10
tabVide.length; // -> 0
```

- **parcourir** un tableau (itérer sur les éléments)
  - à l'aide d'une boucle `for`

```
var tabNombres = [1, 3, 12, -4, 7];
var sommeElements = 0;
for(var i = 0; i < tabNombres.length; i++) {
  sommeElements = sommeElements + tabNombres[i];
}                                    // sommeElements vaut 19
```

  - utilisation des boucles « `for in` »

```
var tabNombres = [1, 3, 12, -4, 7];
var sommeElements = 0;
for(var index in tabNombres) {
  sommeElements = sommeElements +  tabNombres[index];
}                                    // sommeElements vaut 19
```

NB : `index` parcourt les indices pas les valeurs

Ce qui est moins classique...

- les éléments d'un tableaux **ne sont pas typés**

```
var tablo = [12, "timoleon", true, 0, "autre"];
```

- les tableaux sont **dynamiques**
  - leur taille n'est pas figée
  - l'utilisation d'un indice suffit à définir l'élément associé

```
var monTablo = [];      // monTablo est vide
monTablo.length;        // -> 0
monTablo[0] = 1;        // on "ajoute" l'élément d'indice 0
monTablo[1] = 33;       // on "ajoute" l'élément d'indice 1
monTablo.length;        // -> 2
```

  - pas de contrainte de continuité des indices, ils peuvent être « clairsemés » (*sparse*) ($\neq$ « dense »)

```
monTablo[12] = -5;    // on "ajoute" l'élément d'indice 12
2 in monTablo; //-> false ceux d'indice 2,...,11 n'existent pas
```

- itération avec **for in** ne parcourt que les éléments « définis »

```
var monTablo = [1,2];      // éléments d'indice 0 et 1 "définis"
monTablo[3] = 4;           // on "ajoute" l'élément d'indice 3
var somme = 0;             // l'élément d'indice 2 n'existe pas
for(var index in monTablo) {
  somme = somme +  monTablo[index];
}                                          // somme vaut 1+2+4 = 7
```

mais peut poser problèmes « à cause des objets »

- la propriété **length**
  - ne correspond pas au nombre d'éléments si *clairsemé*
  - est plus grand que le dernier indice « défini »
    vaut généralement cet indice + 1

```
monTablo.length;           // -> 3
monTablo[10] = -4;
monTablo.length;           // -> 11
```

Usage

- attention danger
  - pas de typage
  - pas de notion de « dépassement » des bornes du tableau
  - certains éléments peuvent être non définis

    peut être testé via `undefined`
- nous nous limiterons à un usage « classique » des tableaux

  $\Rightarrow$ dense, typé et de taille définie
- tableaux et chaînes de caractères
  - les chaînes de caractères se comportent comme des tableaux non mutables

```
var s = "timoleon";
s.length;                  // -> 8
s[2];                      // -> "m", équivalent de s.charAt(2)
```

mais les méthodes de l'objet `Array` ne sont pas accessibles

Tableaux multi-dimensionnels

- il s'agit de tableaux de tableaux
- chacun des tableaux éléments peut avoir sa propre « taille »

```
var tt = [[1, 2], [3, 4], [5,6]];
tt.length;                                          // -> 3
tt[2][0];                                           // -> 5
tt[1].length;                                       // -> 2
tt[0];                                              // -> [1,2]
var ttt = [[true], ["a", "b", "c"], [10,-4]];
```

```
var tabMult = new Array(10);                        // 10 "lignes"
for (var i = 0; i < tabMult.length; i++) {
   tabMult[i] = new Array(10);                      // 10 "colonnes"
   for(var j = 0; j < tabMult[i].length; j++) {
     tabMult[i][j] = i*j;
   }
}
alert("7 x 8 = "+tabMult[7][8]);                     // -> "7x8 = 56"
```

à suivre...

javascript : agir sur les éléments