Chapitre 6
Tableaux

1. Tableaux à dimension unique

Nous avons entrevu dans le chapitre Développement à partir d'algorithmes le potentiel des tableaux à dimension unique et à dimensions multiples, voyons leur prise en compte sous JavaScript.

1.1 Syntaxe

En JavaScript, un tableau à dimension unique est une variable mémoire "composite" dans laquelle il va être possible de stocker plusieurs données indépendantes, y compris de types différents, avec une indexation de chacune des valeurs par un numéro (ou indice).

L'accès à chaque donnée du tableau se fera donc par l'intermédiaire de cette valeur d'indice.

Une particularité quant à cet indice, sa valeur pour la première cellule du tableau est 0.

Le langage JavaScript fournit plusieurs façons de créer un tableau :

– la syntaxe littérale,

– la syntaxe dite "Programmation orientée objet".

Avec une syntaxe littérale, la déclaration d'un tableau de nom `tabSemaine` de sept cellules contenant les libellés des jours d'une semaine se fait comme suit :

```
var tabSemaine = ["Lundi", "Mardi", "Mercredi", "Jeudi",
"Vendredi", "Samedi", "Dimanche"];
```

Vous noterez que la déclaration s'est accompagnée de l'initialisation de chacune des cellules du tableau `tabSemaine` (de la cellule d'indice 0 à la cellule d'indice 6).

Avec une syntaxe "Programmation orientée objet", vous auriez :

```
var tabSemaine = new Array("Lundi", "Mardi", "Mercredi", "Jeudi",
"Vendredi", "Samedi", "Dimanche");
```

Nous aurions pu déclarer le tableau `tabSemaine` sans lui affecter des valeurs. Des affectations ultérieures peuvent être envisagées, comme par exemple pour le Lundi :

```
tabSemaine[0] = "Lundi";
```

Ce qui est vraiment particulier dans la gestion des tableaux sous JavaScript est l'extrême souplesse autorisée :

– pas de dimensionnement a priori (il est toujours possible d'étendre la taille du tableau en fonction des besoins),
– possibilité de mélanger dans un même tableau des données de types différents,
– possibilité d'utiliser des tableaux associatifs (tableaux pour lesquels les indices sont remplacés par des valeurs textuelles).

Pour accéder dans un traitement au contenu d'une valeur de tableau rangée à une position d'indice particulière, la syntaxe sera :

```
document.write("Le 4ième jour de la semaine est " + tabSemaine[3]);
```

Remarque

Il faut toujours se rappeler que la numérotation des indices débute à zéro.

Enfin, sachez que JavaScript propose une multitude de méthodes s'appliquant sur les tableaux (`Array`). Vous pourrez facilement par ces méthodes insérer, supprimer, repérer des éléments d'un tableau. Il existe même des méthodes de tri (`sort`, `reverse`) pour classer facilement les valeurs contenues dans un tableau sans avoir recours à l'écriture fastidieuse d'un algorithme de tri.

1.2 Exercice n°14 : Décompte des nombres pairs dans un tableau

Sujet

Détermination du nombre de nombres pairs dans un tableau (saisie préalable des valeurs à prévoir au clavier).

Corrigé (partiel) en JavaScript

```
/* Déclaration de variables locales */
/*
i          : Compteur de boucle
nbPairs    : Cumul du nombre de nombres pairs
tableau    : Tableau des nombres
*/
var i, nb_pairs;
var tableau = new Array;

/* Initialisations */
nbPairs = 0;
for (i=1; i<=5; i++)
{
    tableau[i] = parseInt(prompt("tableau[" + i +"] : "));
}

/* Détermination du nombre de nombres pairs dans le tableau */
for (i=1; i<=5; i++)
{
    if (tableau[i]%2 == 0)
    {
        nbPairs = nbPairs + 1;
    }
}

/* Affichage du résultat */
document.write("Le tableau contient " + nbPairs + " nombres pairs");
```

Commentaires du code JavaScript

Rien de vraiment nouveau n'est présenté dans ce script hormis le calcul du modulo. Ce calcul sert ici à déterminer la parité de chaque contenu de cellules du tableau. Il est réalisé par l'intermédiaire l'opérateur `%`.

Vous aurez peut-être noté que dans ce script la cellule d'indice 0 n'a pas été utilisée (la numérotation par la boucle `for` débute à 1). Ce choix rend sans doute plus compréhensible l'algorithme (il n'y a que les informaticiens qui s'accommodent de la numérotation à partir de zéro !).

2. Tableaux à dimensions multiples

Il est fréquent que l'on ait besoin de tableau à dimensions multiples pour gérer des problématiques, notamment en mathématique, en statistique...

JavaScript offre cette possibilité.

2.1 Syntaxe

Comme pour les tableaux à dimension unique, JavaScript permet de déclarer les tableaux à dimensions multiples de plusieurs façons :

– avec une syntaxe littérale,

– avec une syntaxe dite "Programmation orientée objet".

Avec une syntaxe dite "Programmation orientée objet" (encore appelée JSON - *JavaScript Object Notation*), la déclaration d'un tableau de nom `tabMatrice` de deux lignes subdivisées en quatre colonnes avec initialisation se fait comme suit :

```
/* Déclaration du tableau tabMatrice */
var tabMatrice tableau = new Array();

/* Déclaration de la première "ligne" du tableau tabMatrice */
tabMatrice[0]=new Array()

/* Initialisation des 4 "colonnes" de la première "ligne" */
tabMatrice[0][0] = "Un";
```

```
tabMatrice[0][1] = "Deux";
tabMatrice[0][2] = "Trois";
tabMatrice[0][3] = "Quatre";

/* Déclaration de la deuxième "ligne" du tableau tabMatrice */
tabMatrice[1]=new Array()

/* Initialisation des 4 "colonnes" de la deuxième "ligne" */
tabMatrice[1][0] = "Onze";
tabMatrice[1][1] = "Douze";
tabMatrice[1][2] = "Treize";
tabMatrice[1][3] = "Quatorze";
```

2.2 Exercice n°15 : Mini-tableur

Sujet

Soit le tableau `tb` à deux dimensions comportant quatre lignes et cinq colonnes. Réaliser les traitements suivants :

– saisir au clavier des valeurs dans les trois premières lignes et les quatre premières colonnes (on conserve la dernière ligne et la dernière colonne libres pour des additions de lignes et de colonnes),
– additionner les colonnes en dernière ligne et les lignes en dernière colonne.

Corrigé (partiel) en JavaScript

```
 /* Déclaration de variables locales */
var tb = new Array(5);
var numLigne, numColonne;
var valeur;

/* Déclaration de 5 "colonnes" par "ligne" pour le tableau tb */
for (var numLigne=1; numLigne<tb.length; numLigne++)
{
    /* Création des "colonnes" (numérotées de 0 à 5) */
    tb[numLigne]=new Array(6);
}

/* Initialisation du tableau tb (3 lignes * 4 colonnes) par une saisie clavier */
for (numLigne = 1; numLigne <= 3; numLigne++) {
    for (numColonne = 1; numColonne <= 4; numColonne++) {
        valeur = parseInt(prompt("tableau[" + numLigne + "][" + numColonne + "] = "));
        tb[numLigne][numColonne] = valeur;
    }
}

/* Mise à zéro des totaux en ligne n°4 */
```

```
for (numColonne=1; numColonne<=5; numColonne++)
{
    tb[4][numColonne] = 0;
}

/* Mise à zéro des totaux en colonne n°5 */
for (numLigne=1; numLigne<=4; numLigne++)
{
    tb[numLigne][5] = 0;
}

/* Détermination des totaux en ligne n°4 et en colonne n°5 */
for (numLigne=1; numLigne<=3; numLigne++)
{
    for (numColonne=1; numColonne<=4; numColonne++)
    {
        /* Totalisation en ligne n°4 */
        tb[4][numColonne] = tb[4][numColonne]
        + tb[numLigne][numColonne];
        /* Totalisation en colonne n°5 */
        tb[numLigne][5] = tb[numLigne][5]
        + tb[numLigne][numColonne];
        /* Totalisation générale en ligne n°4-colonne n°5 */
        tb[4][5] = tb[4][5] + tb[numLigne][numColonne];
    }
}

/* Affichage du total général */
/* NB : Total de 78 étant donné la technique de remplissage retenue
du tableau tb */
document.write("Total général en tb[4][5] = " + tb[4][5]);
```

Commentaires du code JavaScript

Le tableau `tb` est dans un premier temps déclaré comme étant un tableau à une seule dimension avec cinq cellules, implicitement numérotées de 0 à 4 comme suit :

```
var tb = new Array(5);
```

Vous noterez que la ligne de numéro zéro ne sera pas utilisée par la suite. Ce choix a été fait car l'utilisation ultérieure de cette ligne ne serait pas très intuitive.

Dans un second temps les lignes de ce tableau sont elles-mêmes subdivisées en colonnes, comme ceci :

```
/* Déclaration de 5 "colonnes" par "ligne" pour le tableau tb */
for (var numLigne=1; numLigne<tb.length; numLigne++)
{
    /* Création des "colonnes" (numérotées de 0 à 5) */
    tb[numLigne]=new Array(6);
}
```

Chapitre 4
La plateforme Node.js

1. Présentation de Node.js

Node.js est un environnement permettant d'exécuter du code JavaScript hors d'un navigateur. À l'heure de la rédaction de cet ouvrage, il repose sur le moteur JavaScript V8 développé par Google pour ses navigateurs Chrome et Chromium.

Son architecture est modulaire et événementielle. Il est fortement orienté réseau en possédant pour les principaux systèmes d'exploitation (Unix/Linux, Windows, Mac OS) de nombreux modules réseau (dont voici les principaux par ordre alphabétique : DNS, HTTP, TCP, TLS/SSL, UDP). De ce fait, il remplace avantageusement, dans le cadre qui nous intéresse ici (c'est-à-dire la création et la gestion d'applications web), un serveur web tel qu'Apache.

Créé par Ryan Lienhart Dahl en 2009, cet environnement est devenu rapidement très populaire pour ses deux qualités principales :

– Sa légèreté (en corollaire de sa modularité).
– Son efficacité induite par son architecture monothread (en corollaire de la gestion événementielle que propose nativement l'environnement JavaScript).

Intégrer Node.js dans le développement d'applications web participe donc à la logique actuelle de rendre les opérations d'accès aux données les moins bloquantes possible (pour dépasser la problématique dite du « bound I/O » selon laquelle, avant toute autre cause, la latence globale d'une application est due au temps de latence des accès aux données).

Node.js permet donc, pour les applications web, de créer des serveurs extrêmement réactifs.

Dans ce qui suit, vous allez :

– Installer et tester Node.js sous Linux, Windows ou macOS.
– Créer un serveur HTTP renvoyant une chaîne de caractères.
– Mettre en œuvre un module.
– Créer un serveur HTTP utilisant le module express invoqué sur une route REST et renvoyant des données formatées en JSON, d'abord en totalité, puis filtrées sur une propriété.

Dans ce chapitre, nous n'introduirons que quelques modules (et fonctions) de Node.js.

La documentation complète des modules est disponible à cette adresse : https://nodejs.org/api/

2. Installation et test de Node.js

2.1 Création du fichier de test

Pour tester Node.js, vous allez dans un premier temps créer un code JavaScript qui va être le plus simple possible, et le faire exécuter par Node.js.

Créez donc le fichier testDeNode.js qui ne comprend qu'une ligne de code :

```
console.log("Test de Node");
```

2.2 Installation et test de Node.js sous Ubuntu

- Pour installer Node.js sous Ubuntu, le plus simple est d'utiliser la commande `curl` et le gestionnaire de paquets en ligne de commande (`apt-get`) :

```
curl -sL https://deb.nodesource.com/setup_11.x | sudo -E bash -
sudo apt-get install nodejs
```

Il est à noter qu'au jour de l'écriture de cet ouvrage, la version courante de Node.js est la 11. Pour une version ultérieure, il suffit de changer son numéro de version dans la commande d'installation.

- Un lien symbolique nommé `node` peut être créé pour lancer plus naturellement vos serveurs :

```
sudo ln -s /usr/bin/nodejs /usr/bin/node
```

- Ouvrez un terminal (shell) et exécutez le fichier de test :

```
node testDeNode.js
```

La chaîne de caractères « Test de Node » s'affiche !

Une procédure complète est en ligne sur http://doc.ubuntu-fr.org/nodejs.

2.3 Installation et test de Node.js sous Windows

L'installation de Node.js et son test sous Windows vont se dérouler en quatre étapes :

- Téléchargez l'installateur Windows Installer en vous rendant sur le site officiel de Node.js : https://nodejs.org/en/download
- Exécutez l'installateur (le fichier .msi précédemment téléchargé) en acceptant les conditions d'utilisation et le paramétrage par défaut.
- Redémarrez votre ordinateur.
- Ouvrez l'invite de commandes et exécutez le fichier de test :

```
node testDeNode.js
```

La chaîne de caractères « Test de Node » s'affiche !

2.4 Installation et test de Node.js sous Mac OS

L'installation de Node.js et son test sous Mac Os vont se faire en trois étapes :

- Téléchargez le package d'installation pour Mac Os (Macintosh Installer) en vous rendant sur le site officiel de Node.js : https://nodejs.org/en/download/
- Ouvrez un terminal et installez le package :

```
pkg install nomPackage.pkg
```

- Exécutez le fichier de test :

```
node testDeNode.js
```

La chaîne de caractères « Test de Node » s'affiche !

3. La modularité de Node.js

3.1 Les modules et les packages

Une des principales forces de Node.js est d'être modulaire (et notamment de proposer de nombreux modules réseau). Si certains de ces modules sont installés directement en même temps que Node.js, la plupart doivent être installés à la demande.

Lors de la création d'une application qui exige l'installation de modules, deux méthodes sont possibles pour effectuer celle-ci :

– Directement avec le gestionnaire de modules npm (et son option `install`).
– Indirectement (mais toujours avec npm) via la spécification des dépendances de l'application (c'est-à-dire des modules nécessaires à celle-ci) dans un fichier nommé package.json.

Un module est utilisé dans une application avec la fonction `require()` :

```
var moduleDansVotreApplication = require('<nomDuModule>');
```

Votre application Node.js peut être elle-même réutilisée comme module sous certaines conditions qui seront présentées ultérieurement.

Attardons-nous sur un point un peu subtil : la distinction entre modules et packages.

Les modules sont les briques conceptuelles d'une application Node.js. Un module peut être organisé en plusieurs codes JavaScript et dépendre d'autres modules. Ainsi, toutes les ressources nécessaires à un module (les codes, le fichier package.json spécifiant ses dépendances...) sont regroupées dans un package qui, de fait, est un dossier.

Donc, si les deux termes sont quasiment interchangeables, le terme « module » renvoie plus à la fonctionnalité globale, et « package » au dossier et à l'organisation des fichiers de code qui se trouvent dans celui-ci.

3.1.1 Le gestionnaire de modules de Node.js : npm

npm (*Node.js Package Manager*) est le gestionnaire de modules de Node.js (il est installé avec celui-ci).

Les modules sont installés globalement dans le dossier node_modules, situé au niveau des répertoires système si l'option `-g` est utilisée :

```
npm install -g <module>
```

ou sinon (sans l'option `g`) dans le répertoire courant (mais également dans un dossier nommé node_modules).

3.1.2 Spécification des dépendances : le fichier package.json

Pour spécifier les dépendances nécessaires à la création d'une application Node.js (c'est-à-dire les modules associés aux packages nécessaires à celle-ci), il est recommandé de créer un fichier nommé package.json.

Dans le contexte d'un fichier package.json, nous ne parlerons plus que de packages (et non de modules).

Voici un schéma minimal de ce fichier :

```
{
   "name":        "<nom de l'application>",
   "version":     "<version de l'application>",
   "description": "<description de l'application>",
   "author":      "<nom de l'auteur de l'application>",
   "main":        "<code à exécuter comme point d'entrée>",
```

```
    "scripts": {
        "start": "node <code à exécuter>"
    },
    "dependencies": {
      "<nom du package>": "<version minimale du package à installer>",
      ...
    }
}
```

Pour installer les modules nécessaires, la commande suivante doit être exécutée :

```
npm install
```

Et voici celle qui va lancer le serveur :

```
npm start
```

Pour créer un squelette de fichier *package.json*, utilisez la commande suivante :

```
npm init --yes
```

Dans ce cas, la valeur de la propriété `main` est initialisée à index.js. Expliquons un peu l'intérêt de cette propriété :

Si votre application devient un package (comprenant un ou plusieurs codes réutilisables), la propriété `main` désigne le code qui est le point d'entrée dans le package lors de l'exécution de l'instruction `require()`.

3.2 Création d'un premier serveur Node.js de test

Vous allez écrire votre premier serveur (le bien classique « Hello World ») en utilisant le module HTTP qu'offre Node.js.

▶ Saisissez le code de ce serveur dans le fichier helloAvecNode.js :

```
var http = require('http');

var server = http.createServer((request, => response){
    response.end('Hello World de Node.js');
});

server.listen(8888);
```