Chapitre 2-2
Conditionnement des traitements

1. Présentation de la syntaxe

La structure conditionnelle en JavaScript est très proche syntaxiquement de celle vue précédemment en langage descriptif algorithmique.

Tout à fait classiquement le bloc d'instructions à exécuter dans le cas où la condition testée est vraie est délimité par un jeu d'accolades ({}). Il est aussi possible de prévoir une séquence d'instructions alternative avec le mot-clé `else`. Cette séquence sera aussi encadrée par des accolades.

Le langage JavaScript est assez permissif quant au positionnement de ces accolades. Ainsi vous rencontrerez dans les scripts les constructions suivantes :

```
if (condition)
{
        Actions_1;
}
else
{
        Actions_2;
}
```

ou

```
if (condition) {
        Actions_1;
}
else {
        Actions_2;
}
```

ou

```
if (condition) { Actions_1; } else { Actions_2; }
```

avec :

– condition représentant un test de comparaison générant un résultat booléen `true` ou `false`,
– `Actions_1` et `Actions_2` représentant des séquences d'instructions (en général réparties sur plusieurs lignes).

Remarque

Attention pour effectuer un test de comparaison en égalité l'opérateur est le double égal (==) à ne pas confondre avec le simple égal (=) qui sert à effectuer les affectations.

Dans les tests de comparaison avec des constantes on préférera par exemple :

```
if (5 == compteur)
{
        Actions_1;
}
else
{
        Actions_2;
}
```

à

```
if (compteur == 5)
{
        Actions_1;
}
else
{
        Actions_2;
}
```

car l'étourderie consistant à confondre le == avec le = est plus facile à détecter dans le premier cas, JavaScript renvoyant une erreur. Dans le second cas (si vous mettez un = au lieu d'un ==) la séquence `Actions_1` sera systématiquement exécutée car l'évaluation de compteur = 5 donnerait toujours `true` (JavaScript estimant que l'affectation est réussie).

2. Exemples

Remarque

Pour faciliter le repérage des exercices JavaScript, la numérotation vue dans le chapitre Développement à partir d'algorithmes sera conservée.

2.1 Exercice n°6 : Polynôme du second degré

Sujet

Calculer les racines d'un polynôme du second degré Ax^2+Bx+C (avec A<>0 dans l'absolu mais ce test ne sera pas effectué ici). Les valeurs A, B et C seront saisies au clavier.

Corrigé (partiel) en JavaScript

```
/* Saisie des paramètres */
a = prompt("a :");
b = prompt("b :");
c = prompt("c :");

/* Calcul du discriminant */
delta = (parseInt(b) * parseInt(b)) - (4 * parseInt(a) * parseInt(c));

/* Affichage des paramètres */
document.write("a : " + a + "<br />");
document.write("b : " + b + "<br />");
document.write("c : " + c + "<br />");
document.write("Discriminant delta : " + delta + "<br />");

/* Détermination des racines */
if (delta < 0)
{
   document.write("Pas de solutions");
}
else
{
   if (delta == 0)
   {
         document.write("Solution unique : " + (-b / (2 * a)));
   }
   else
   {
         document.write("Solution n°1 : " + (-b +  Math.sqrt(delta)) /
         (2 * a) + "<br />");
         document.write("Solution n°2 : " + (-b -  Math.sqrt(delta)) /
```

```
            (2 * a));
    }
}
```

Commentaires du code JavaScript

Vous remarquerez qu'il a fallu imbriquer deux structures conditionnelles pour traiter le problème posé. La non-imbrication des structures était possible (équivalent de trois **Si ... Finsi** successifs au niveau algorithmique) mais cette solution n'aurait pas été optimale. Vous noterez aussi le soin particulier apporté au niveau de l'alignement des accolades et également au niveau du décalage (indentation) des blocs d'instructions.

Pour déterminer la racine carrée du discriminant (`delta`), il a fallu avoir recours à la méthode `sqrt` de l'objet JavaScript `Math`. Nous aurons l'occasion de revoir cet objet plus loin dans ce livre. Le calcul de la racine carrée aurait aussi pu être effectué par une élévation de `delta` à la puissance 0.5. L'opération de l'élévation à la puissance (exponentiation) est notée `^`.

2.2 Exercice n°8 : Impression du libellé d'un mois

Sujet

Imprimer en lettres le mois correspondant à un numéro donné au clavier (compris entre 1 et 12). Le contrôle de la saisie n'est pas à prévoir.

Corrigé (partiel) en JavaScript

```
/* Déclaration de variables locales */
var mois;

/* Saisie du numéro de mois */
mois = parseInt(prompt("Numéro du mois (1 à 12) :"));

/* Affichage du résultat */
switch (mois)
{
   case 1:
         document.write("Mois n° " + mois +" : Janvier");
         break;
   case 2:
         document.write("Mois n° " + mois +" : Février");
         break;
   case 3:
         document.write("Mois n° " + mois +" : Mars");
         break;
   case 4:
         document.write("Mois n° " + mois +" : Avril");
```

```
            break;
      case 5:
            document.write("Mois n° " + mois +" : Mai");
            break;
      case 6:
            document.write("Mois n° " + mois +" : Juin");
            break;
      case 7:
            document.write("Mois n° " + mois +" : Juillet");
            break;
      case 8:
            document.write("Mois n° " + mois +" : Août");
            break;
      case 9:
            document.write("Mois n° " + mois +" : Septembre");
            break;
      case 10:
            document.write("Mois n° " + mois +" : Octobre");
            break;
      case 11:
            document.write("Mois n° " + mois +" : Novembre");
            break;
      case 12:
            document.write("Mois n° " + mois +" : Décembre");
            break;
      default:
            document.write("Erreur de saisie sur le n° de mois");
}
```

Commentaires du code JavaScript

Plutôt que, comme dans l'exercice précédent, d'imbriquer des structures conditionnelles (fastidieux à écrire dans notre cas), la structure `switch` est ici utilisée.

Notez au niveau de la saisie du numéro du mois dans la variable `mois` (non typée au niveau de sa déclaration) qu'une conversion est demandée par l'intermédiaire de la méthode `parseInt`. La variable `mois`, une fois la saisie et la conversion effectuées, sera de type entier pour la suite du traitement.

La variable `mois` est testée par la structure conditionnelle `switch`. Dans le cas où la variable `mois` vaut 1, l'affichage de l'intitulé "`Janvier`" est effectué, et ainsi de suite.

Dans le cas où une saisie erronée du numéro de mois est faite alors l'instruction intégrée dans le cas `default` est exécutée.

Revenons aussi sur le rôle primordial de l'instruction `break`. En son absence en fin de chacun des cas, dès que la condition est vérifiée le traitement associé est déclenché mais ce serait également le cas pour tous les traitements suivants (même si la condition n'est pas respectée pour ces traitements). Par exemple, la saisie de la valeur 5 en tant que numéro de mois générerait l'affichage de `Mai`, `Juin`, `Juillet`, ..., `Décembre`.

Chapitre 3
Node.js

1. Présentation

Node.js est une plateforme bien adaptée pour les développeurs qui souhaitent utiliser JavaScript côté serveur. Elle permet de créer des applications web et autres types d'applications réseau de manière assez efficace comme des API, des services de streaming de données en temps réel, des serveurs de jeux, des applications IoT (internet des objets). En utilisant le moteur JavaScript V8 de Google Chrome, Node.js apporte JavaScript dans l'environnement serveur, ce qui facilite le partage de code entre le client et le serveur et simplifie le processus de développement.

L'un des aspects intéressants de Node.js est sa manière de gérer les opérations d'entrée/sortie (I/O). Au lieu d'utiliser plusieurs threads pour chaque requête, Node.js fonctionne sur un modèle événementiel et asynchrone. Cela signifie que lorsqu'une tâche I/O est en cours, Node.js peut passer à une autre sans attendre que la première soit terminée. Ce modèle est particulièrement utile pour les applications qui doivent gérer de nombreuses connexions simultanément.

Il bénéficie également d'un écosystème très actif, grâce à son gestionnaire de paquets npm. Les développeurs ont accès à une multitude de modules open source qu'ils peuvent intégrer dans leurs projets, ce qui permet de gagner du temps et de faciliter le développement d'applications. Que vous ayez besoin d'un framework pour construire votre application, d'une bibliothèque pour vous connecter à une base de données ou d'un outil pour automatiser certaines tâches, il y a de fortes chances que vous trouviez ce qu'il vous faut dans l'écosystème de Node.js.

Node.js est reconnue pour sa simplicité d'utilisation. Il convient à une large gamme de projets, des petits sites web aux applications d'entreprise plus complexes. La communauté autour de cette plateforme est également un grand atout, car elle est très active et offre de nombreuses ressources comme des tutoriels et de la documentation, pour aider les développeurs à progresser et à résoudre les problèmes qu'ils pourraient rencontrer.

2. L'installation de Node.js 18

2.1 Sous Ubuntu

Vous pouvez installer Node.js sous Ubuntu en utilisant le gestionnaire de paquets apt. C'est la méthode la plus basique et la plus rapide. Voici les étapes à suivre :

- Ouvrez un terminal sur votre système Ubuntu.
- Mettez à jour la liste des paquets disponibles :

```
sudo apt update
```

- Installez Node.js et npm (le gestionnaire de paquets de Node.js) :

```
sudo apt install nodejs npm
```

- Assurez-vous que l'installation s'est déroulée correctement en vérifiant les versions de Node.js de npm :

```
node -v
npm -v
```

Ces commandes affichent les versions respectives de Node.js et de npm installées sur votre système. Cette méthode installe Node.js et npm à partir des dépôts officiels d'Ubuntu, ce qui facilite les mises à jour et la gestion des packages.

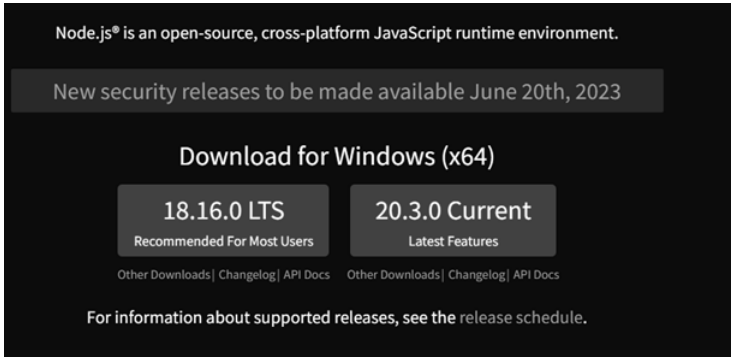
Remarque

La version de Node.js fournie avec apt peut être légèrement plus ancienne que la version LTS (Long Term Support) la plus récente. Si vous souhaitez installer une version plus récente de Node.js, il suffit de la télécharger et de l'installer manuellement en utilisant le site officiel de Node.js.

Une fois l'installation terminée, vous pouvez utiliser Node.js pour développer des applications JavaScript côté serveur.

2.2 Sous Windows

- Rendez-vous sur le site officiel de Node.js à l'adresse : https://nodejs.org



Boutons de téléchargement sur le site nodejs.org

- La page d'accueil propose deux versions de Node.js : LTS et Current. Cliquez sur le bouton **18.16.0 LTS** pour télécharger la dernière version LTS de Node.js (à l'heure de la rédaction de cet ouvrage).

Remarque

La version LTS est recommandée pour la plupart des utilisateurs, car elle est plus stable et bénéficie d'un support à long terme. La version Current est la version la plus à jour. Elle possède des fonctionnalités possiblement non encore stables. Vous pouvez suivre le cycle de développement de Node.js via le lien GitHub : https://github.com/nodejs/Release.

- Une fois le téléchargement terminé, ouvrez le fichier d'installation (.msi) que vous avez téléchargé.
- L'assistant d'installation de Node.js démarre. Suivez les instructions de l'assistant pour configurer l'installation. Vous pouvez généralement accepter les paramètres par défaut, sauf si vous avez des besoins spécifiques.
- Vous pouvez sélectionner les composants supplémentaires à installer. La plupart du temps, il est recommandé de cocher les cases **Automatically install the necessary tools for Node.js** et **Add to PATH** pour que Node.js soit accessible depuis n'importe quel emplacement dans le terminal.
- Cliquez sur le bouton **Next** et suivez les instructions de l'assistant pour terminer l'installation.
- Pour vérifier que Node.js est installé correctement, ouvrez une fenêtre de terminal (invite de commandes ou PowerShell) et exécutez les commandes suivantes :

```
node -v
npm -v
```

Ces commandes affichent respectivement la version de Node.js et la version de npm installées sur votre système.

2.3 Sous Mac

- Rendez-vous sur le site officiel de Node.js à l'adresse : https://nodejs.org/en/download

Les différents packages selon l'environnement

- La page d'accueil propose deux versions de Node.js : LTS et Current. Cliquez sur le bouton **LTS** pour télécharger la dernière version LTS de Node.js.
- Une fois le téléchargement terminé, ouvrez le fichier d'installation (.pkg) que vous avez téléchargé.
- L'assistant d'installation de Node.js démarre. Suivez les instructions de l'assistant pour configurer l'installation. Vous pouvez généralement accepter les paramètres par défaut, sauf si vous avez des besoins spécifiques.
- Vous pouvez sélectionner les composants supplémentaires à installer. La plupart du temps, il est recommandé de cocher les cases **Automatically install the necessary tools for Node.js** et **Install npm package manager** pour installer npm (le gestionnaire de paquets de Node.js).
- Cliquez sur le bouton **Install** et suivez les instructions de l'assistant pour terminer l'installation.
- Pour vérifier que Node.js est installé correctement, ouvrez une fenêtre de terminal (invite de commandes ou PowerShell) et exécutez les commandes suivantes :

```
node -v
npm -v
```

Ces commandes affichent respectivement la version de Node.js et la version de npm installées sur votre système.

3. Les modules

3.1 La notion de module

En Node.js, un module est une unité de code réutilisable qui encapsule un ensemble de fonctionnalités spécifiques. Il permet d'organiser et de structurer le code en le séparant en différentes parties, ce qui facilite la maintenance, la collaboration et la réutilisation du code. Les modules sont basés sur le système de modules CommonJS, qui est une spécification pour les modules JavaScript. Chaque fichier JavaScript dans Node.js est considéré comme un module, et les variables, les fonctions et les objets définis dans ce fichier ne sont pas accessibles depuis d'autres modules par défaut.

Pour utiliser un module dans Node.js, vous devez l'importer à l'aide de l'instruction `require()`. L'instruction `require()` prend en paramètre le chemin du module que vous souhaitez importer, qu'il s'agisse d'un module natif de Node.js, d'un module installé via npm ou d'un module personnalisé que vous avez créé.

Voici un exemple d'utilisation d'un module dans Node.js (cet exemple suppose d'avoir un fichier appelé math.js qui exporte des fonctions mathématiques) :

```
// math.js
exports.add = function(a, b) {
  return a + b;
};

exports.multiply = function(a, b) {
  return a * b;
};
```

Dans un autre fichier JavaScript, vous pouvez importer le module math.js et utiliser ses fonctions :

```
// app.js
const math = require('./math.js');

console.log(math.add(2, 3)); // Affiche 5
console.log(math.multiply(4, 5)); // Affiche 20
```

Dans cet exemple, nous avons utilisé `require()` pour importer le module math.js en spécifiant le chemin relatif vers le fichier. Ensuite, nous avons accédé aux fonctions `add` et `multiply` exportées par le module et nous les avons utilisées dans notre fichier app.js.

C'est ainsi que les modules sont utilisés et importés en Node.js. Ils permettent de découper le code en modules distincts pour une meilleure organisation et une réutilisation facile.

3.2 Le gestionnaire de module npm

Le gestionnaire de module npm (*Node Package Manager*) est un outil qui permet aux développeurs de gérer les packages et les dépendances de leurs projets Node.js. Il est livré avec Node.js et est utilisé pour installer, mettre à jour et supprimer des packages, ainsi que pour gérer les versions des packages installés. Il est également utile pour partager des modules entre les projets et pour travailler avec des bibliothèques tierces. Lorsque vous créez un nouveau projet, vous pouvez initialiser npm en exécutant la commande `npm init`. Cela crée un fichier package.json qui stocke les informations sur votre projet, y compris les dépendances et les scripts. Lorsque vous avez besoin d'ajouter une nouvelle dépendance à votre projet, vous pouvez l'installer en utilisant la commande `npm install`. Il est aussi possible d'installer des packages pour les utiliser dans n'importe quel projet. Enfin, npm dispose d'un grand nombre de packages gratuits et open source. Il est donc courant de trouver une solution à un problème en cherchant une bibliothèque npm. Cela permet aux développeurs de gagner du temps et de se concentrer sur la logique métier.

Le dossier node_modules est un répertoire utilisé par Node.js pour stocker les modules installés via npm. Lorsque vous installez un module à l'aide de la commande `npm install`, les fichiers du module sont téléchargés depuis le registre npm et placés dans le dossier node_modules de votre projet. Chaque module installé via npm a son propre dossier dans node_modules, et ce dossier porte le nom du module en question. À l'intérieur de ce dossier, vous trouverez les fichiers JavaScript, les dépendances et autres ressources spécifiques au module.

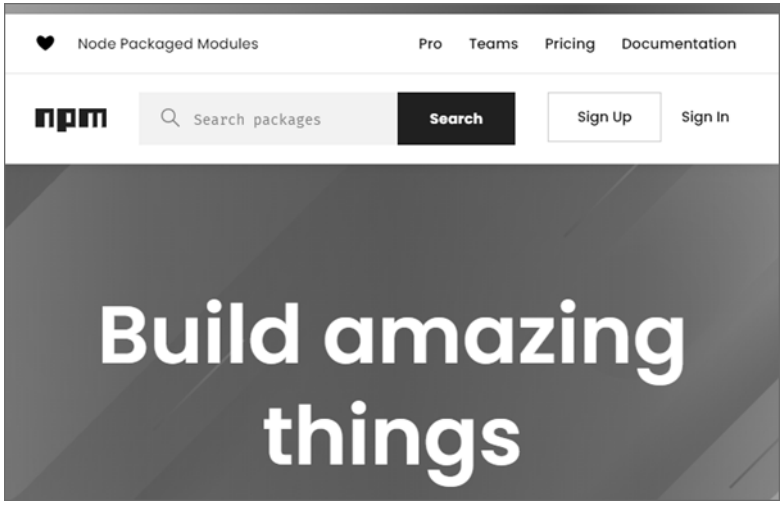
Par exemple, si vous installez le module Express, un dossier express est créé dans node_modules. Ce dossier express contient tous les fichiers nécessaires à l'utilisation du module Express dans votre projet.

Lorsque vous utilisez des modules dans votre code, Node.js recherche automatiquement ces modules dans le dossier node_modules. Vous pouvez les importer à l'aide de l'instruction `require()` sans spécifier le chemin complet du module.

Remarque

Les modules installés occupent de la place en stockage. Il est courant de supprimer le dossier node_modules ou alors de l'ignorer lorsque l'on versionne un projet. Le dossier node_module n'est pas nécessaire pour le stockage, car toutes les informations utiles sont présentes dans packages.json. En revanche, le dossier node_modules est indispensable pour le lancement de l'application (il suffit de le générer de nouveau via la commande `npm install`).

3.3 npmjs.com



Présentation du site npmjs.com