ENSEIRB-Matmeca
2021-2022

1ère année - Filière Info
PG104

TD n°1 - Présentation de Ecmascript

Exercice 1: Mise en place de l'environnement Javascript

Le moteur d'exécution de Javascript que nous allons utiliser se nomme Node.js (cf. `https://nodejs.org`). Il consiste en un exécutable `node`, qui peut :

— soit être utilisé comme boucle d'interaction, en tapant simplement la commande `node`

— soit interpréter un fichier de code `file.js` donné en tapant la commande `node file.js`

Il est possible de vérifier la version de `node` installée sur votre machine avec la commande `node --version`. Les versions les plus récentes ont le numéro de version majeur 14.

▷ Sur les machines de l'Enseirb-Matmeca, la commande de `node` accessible pas défaut correspond à une version ancienne. La version la plus récente (à utiliser pour ces TDs) est dans le répertoire `/opt/nodejs/14.15.4/bin`. Pour éviter de taper à chaque fois le chemin complet, il est nécessaire de l'ajouter dans le `PATH` (en tapant la commande `export PATH=/opt/nodejs/14.15.4/bin:$PATH` dans le terminal). Enfin, pour éviter de répéter cette manipulation à chaque séance, il est recommandé de l'ajouter dans le fichier `.bash_export`.

Il existe différentes manières plus ou moins raffinées d'éditer des fichiers Javascript, la plus simple étant certainement d'utiliser votre éditeur favori et d'exécuter le code dans un terminal [1].

1. Copier le code suivant depuis le fichier de sources [2] (disponible depuis la page du cours) dans un fichier sur votre machine avec l'extension `.js`. L'évaluer dans un terminal.

```
function factorial(n) {
    if (n <= 1)
        return 1;
    else
        return n * factorial(n-1);
}
```

2. Exécuter le fichier précédent avec la commande `node`. Ajouter au fichier un appel à la fonction `console.log` pour afficher un calcul de la fonction `factorial`.
3. Lancer la boucle d'interaction `node`, et charger le fichier à l'intérieur en utilisant la commande `.load file.js` (avec un "`.`" devant "`load`") [3]. Utiliser la fonction `factorial` depuis l'intérieur de

1. Vous êtes d'ailleurs prié de ne pas procéder à des installations complexes *pendant* le TD. Avant ou après la séance, faites comme bon vous semble.
2. Et surtout *ne le copiez pas* à partir de ce fichier PDF que vous êtes en train de lire, ou alors vous choisissez de gérer vous même les éventuelles erreurs de transcription.
3. Noter qu'il ne s'agit pas d'une commande Javascript, mais d'une commande spéciale de l'interpréteur `node`.

la boucle d'interaction. Utiliser `.exit` pour sortir de la boucle d'interaction.

▷ Pour cet enseignement, on pourra privilégier l'éditeur `emacs` en mode Javascript. Il se place dans le mode approprié en ouvrant un fichier `.js`. Au besoin, utiliser la commande `M-x javascript-mode` pour activer le mode Javascript. Pour éviter d'alterner en permanence entre l'éditeur et un terminal, il est possible d'utiliser dans `emacs` la commande `M-x compile` qui exécute par défaut un `make`, mais qu'il est très simple de remplacer par la commande que l'on souhaite, comme par exemple `node td1.js`.

Exercice 2: Quelques expérimentations avec `npm`

Avec le moteur Node.js est livré un gestionnaire de paquetages extrêmement populaire nommé `npm` (cf. `https://www.npmjs.com`). Ce gestionnaire permet d'ajouter avec facilité d'autres bibliothèques Javascript ainsi que des utilitaires plus ou moins pratiques.

1. Se placer dans un répertoire dédié aux TDs de PG104, et appliquer la commande `npm init -y`. Cette commande permet de créer un fichier `package.json` permettant de configurer les commandes liées à `node`.
2. Installer le paquetage calc avec la commande `npm install calc`. Cette commande va télécharger un ensemble de paquetages placés dans le sous-répertoire `node_modules`.

Lorsqu'un paquetage est installé, il ajoute parfois avec lui un ensemble d'exécutables. Noter que `npm` est connu pour comporter des paquetages avec des failles de sécurité, et donc qu'il faut faire attention en exécutant du code téléchargé automatiquement. Les exécutables installés se situent dans le sous-répertoire `./node_modules/.bin`.

3. Exécuter la commande `./node_modules/.bin/calc "3*5/7"`.

▷ On veillera à respecter aux mieux des standards de codage lorsque l'on écrit du code Javascript. Il n'existe pas de standard absolu en Javascript, néanmoins il est fortement recommandé de garder les mêmes conventions dans un code donné. La page de Mozilla `https://developer.mozilla.org/en-US/docs/MDN/Guidelines/Code_guidelines/JavaScript` fournit un ensemble de recommandations sensé.

Le compilateur Typescript peut être installé à l'aide de `npm`, et nous allons compiler un fichier d'exemple pour expérimenter avec :

4. Installer le paquetage typescript avec la commande `npm install typescript`.
5. Copier le code suivant dans un fichier avec l'extension `.ts` :

```
function factorial(n: number) : number {
    if (n <= 1)
        return 1;
    else
        return n * factorial(n-1);
}

console.log(factorial("5"));
```

⚠ La compilation de ce fichier va créer par défaut un fichier du même nom avec l'extension `.js`. Par défaut, il n'y a *aucun* avertissement lors de la création de ce fichier si un fichier de ce nom existe déjà. *You have been warned.*

6. Compiler le fichier avec la commande `./node_modules/.bin/tsc ⟨file.ts⟩ --outFile ⟨other.js⟩`
7. Corriger l'erreur dans le fichier, le recompiler, et l'exécuter.

Exercice 3: Quelques lignes de Ecmascript

La spécification du langage Ecmascript spécifie très précisément les types de données manipulables dans le langage. Ici, nous allons en manipuler quelques un pour se faire la main, en commençant par les valeurs numériques :

▷ Dans les cours, Javascript est utilisé en mode strict (`node --use_strict`) et toutes les variables doivent impérativement être définies avec `let` ou `const`.

1. L'opérateur `**` permet de faire des exponentiations de nombre. Quelle est la plus grande puissance de 2 que l'on peut représenter de manière exacte en Javascript ? Quel est le plus grand entier que l'on peut représenter ?
2. Construire le grand entier `const twobig = 2n`, et vérifier que sur ces valeurs, il est possible de calculer bien plus loin qu'avec les nombres précédents.
3. Quel est le type de `twobig` ? Quelles sont les méthodes que l'on peut appeler sur les valeurs de ce type ? (Les méthodes en question sont décrites à `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/BigInt`)
4. Que se passe t'il lorsqu'on essaie d'additionner `twobig` avec un (petit) entier ? Avec une chaîne de caractères ?

Les chaînes de caractères possèdent déjà largement plus de méthodes.

5. Compter sur la page `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String` le nombre de méthodes pour un objet de la classe `String`.
6. Utiliser la méthode `includes` pour tester si une chaîne en inclut une autre.
7. Pouvez-vous expliquer la différence syntaxique entre une fonction, une méthode de classe, et une méthode d'instance ?

Exercice 4: Quelques fonctions simples

Écrire et tester les fonctions suivantes :

- `square`, qui calcule le carré d'un nombre passé en paramètre,
- `discriminant`, qui calcule le discriminant $b^2 - 4ac$ d'un trinôme $ax^2 + bx + c$ donné par ses coefficients a, b et c.
- `evalQuadratic`, qui évalue un trinôme $ax^2 + bx + c$ donné par ses coefficients a, b et c en un point x donné.
- `root1` et `root2` qui, étant donné un trinôme $ax^2 + bx + c$, calculent respectivement la valeur de la première racine et de la deuxième la racine du trinôme.

- `caracQuadratic` qui, étant donné un trinôme ax^2+bx+c, indique si le trinôme aura 2 racines réelles, 1 racine réelle ou 2 racines complexes.

Bonus : tester les fonctions avec la bibliothèque `mocha` (`https://mochajs.org`).

Exercice 5: Bases numériques (style impératif)

1. Écrire une fonction `convert` utilisant un style *impératif* (i.e utilisant une boucle) et transformant un nombre entier en la chaîne de caractères de sa représentation binaire.

   Exemple :

   ```
   convert(666) ;; → "1010011010"
   ```

2. Étendre cette fonction pour qu'elle fonctionne en n'importe quelle base ≤ 9. Pour les plus aventureux, gérer les bases ≤ 35 (l'utilisation de la méthode `charCodeAt()` pourrait servir).

   Exemple :

   ```
   convert2Base(666, 2)  ;; → '1010011010'
   convert2Base(666, 3)  ;; → '220200'
   convert2Base(666, 16) ;; → '29A'
   ```

3. Pour tester votre fonction, écrire la fonction `unconvert` qui prend une chaîne de caractères et une base et calcule la valeur entière correspondante.