Programmation Web
JavaScript

Halim Djerroud (hdd@ai.univ-paris8.fr)
Olga Melekhova (melekhova@gmail.com)

Objectifs

— Travailler avec des fonctions (nombre d'arguments variables, paramètres par défaut...) ;
— Construisez des fonctions d'ordre supérieur et regardez les méthodes standard du tableau ;
— Écrire du code sans effet de bord (ex. ne pas muter l'entrée ou l'état global) ;
— Introduire le paradigme de la programmation fonctionnelle ;
— Crée et manipule des objets littéraux ;
— Démonstration de Chrome DevTools (débogueur, traçage réseau...).

Exercice 1. Implémenter sum(...terms)

Implémentez une fonction `sum` qui accepte n'importe quel nombre d'arguments numériques et renvoie la somme. Il génère une erreur personnalisée en l'absence de tout argument.

```
console.log(sum()) // soulever une Erreur ('At least one number is expected')
console.log(sum(1)) // affiche 1
console.log(sum(1, 2, 3)) // affiche 6
```

Exercice 2. Implémenter un filtre(tableau, prédicat)

Implémentez une fonction `filter(array, predicate)` qui renvoie un nouveau tableau contenant uniquement les éléments de `array` pour lesquels `predicate(item)` est vrai. Le `array` d'origine ne doit pas être modifie (aucun effet bord autorisé).

```
const array = [1, 2, 3, 4, 5]
const filteredArray = filter(array, item => item > 2) // [3, 4 5]
```

Après l'avoir implémentée, consulter la documentation de la fonction suivante : Array.filter.

Exercice 3. Implémenter map(array, transform)

Implémentez une fonction `map(array, transform)` qui renvoie un nouveau tableau avec chaque `item` de `array` remplacé par transform(item). Le `array` d'origine ne doit pas être modifié (aucun effet de bord autorisé).

```
const array = [1, 2, 3, 4, 5]
const doubled = map(array, item => item * 2) // [2, 4, 6, 8, 10]
```

Après l'avoir implémentée, consulter la documentation de la fonction `Array.map`.

Excercise 4. Analyse CSV de base en objets littéraux

Je partage avec vous un dump CSV simplifié des contributeurs Apache (Original source). En raison de la taille du fichier, vous ne pouvez pas simplement l'intégrer dans votre code JavaScript. Au lieu de cela, je vous fournis un extrait qui extrait sur le site Web de Thomas Veillard.

De votre point de vue, le point d'entrée est la fonction `processData`.

```
// This code downloads a CSV file from my website, reads it as text
// and calls `processData(csvText)` on success. Do not worry about
// the details about `fetch` for now, as we will cover them later.
fetch('https://thomas-veillard.fr/wp-content/uploads/2021/07/
               apache-contributors-projects.csv')
  .then(res => res.text())
  .then(processData)
  .catch(console.log)

function processData (csvText) {
  // write your code here
}
```

Pour cette question, parser le fichier csv comme un tableau d'objets littéraux. Chaque objet représente une contribution de quelqu'un à un projet Apache. Par conséquent, chaque objet doit avoir les propriétés suivantes :

- *username* of the contributor, originally called `svn_id` in the CSV ;
- *realName* of the contributor, originally called `real_name` in the CSV ;
- *website* of the contributor, which should be `null` if empty in CSV ;
- *projectName*, originally called `project_name` in the CSV ;

Vous devez fournir 2 implémentations de la fonction d'analyse :

1. **Programmation de style impératif :** vous pouvez utiliser des variables (`let, var...`) et contrôler des flux tels que (`white, for, if...`).
2. **Programmation de style fonctionnel :** vous ne pouvez utiliser que des constantes (pas de let ni de var) et vous n'êtes pas autorisé à modifier quoi que ce soit (ex `array[0] = 0` est interdit). Les méthodes telles que `Array.map` sont fortement encouragées.

Exercice 5. Calculer des statistiques sur les contributions

Calculez et affichez pour consoler les métriques suivantes. L'utilisation du paradigme de la programmation fonctionnelle est encouragée.

1. **Nom du premier projet par ordre alphabétique croissant**. Assurez-vous de comparer sans tenir compte de la casse et de gérer correctement les signes diacritiques.
2. **Le nombre de contributeurs uniques**. Unicity peut être implémenté avec `array.filter()`.
3. **La longueur moyenne du nom des contributeurs**. Bien sûr, vous devez travailler sur des noms uniques.
4. **Nom du contributeur le plus actif** (par nombre de projets). C'est comme regrouper les contributions par nom de contributeurs, trier par nombre de contributions et éventuellement prendre la première...
5. **TOP 10 des projets les plus contribués**. Il y a encore un groupby à utliser.

***Astuces** : Il n'y a pas de fonctions intégrées sur Array pour prendre facilement des valeurs uniques ou un regroupement par critère. Pour un tel traitement, je recommande d'écrire des fonctions helper.*

Bonus : testez votre code vous-même

Il est courant d'écrire des spécifications (tests automatiques) lors de la production de votre code, comme je l'ai fait. Je les partage, afin que vous puissiez tester vos propres implémentations. Le testeur est moka, il exécute donc votre code avec node.

Conseils : bien sûr, vous devez adapter mes tests, afin qu'ils appellent votre propre code. Cela dépend principalement de la structure de votre code... : Télécharger les spécifications moka (code source)

Conseils bonus

L'exercice précédent a encouragé l'écriture d'aides comme `uniq()` ou `groupBy()`. Les bibliothèques populaires telles que lodash les implémentent déjà.

Avant de les utiliser dans vos projets frontend, pensez à la pénalité de poids. Au moment de la rédaction, la taille minifiée de lodash est d'environ 72 Ko (à distribuer sur le réseau et à exécuter à chaque fois que le JS s'exécute dans le navigateur).

De manière générale, les fonctions de lodash couvrent des cas extrêmes que vous ne rencontrerez probablement jamais dans votre projet (ce qui implique du code supplémentaire). De plus, certains d'entre eux sont désormais obsolètes pour la dernière version de JavaScript. Si le sujet vous intéresse, il y a une bonne littérature.

```
➜  00. correction git:(master) ✗ npm run test

> correction-tp1@1.0.0 test /home/thomas/Téléchargements/correction_tp_1/00. correction
> mocha --timeout 5000 specs/**/*


  ex. 1
    sum(...terms)
      ✔ should raise an error if not term to sum
      ✔ should sum 1 term
      ✔ should sum 2 terms
      ✔ should sum 3 terms

  ex. 2
    filter(array, predicate)
      ✔ should return a new array
      ✔ should keep only items for those the predicate is true
      ✔ should not mutate the original array

  ex. 3
    map(array, transform)
      ✔ should return a new array
      ✔ should transform every item of the array
      ✔ should not mutate the original array

  ex. 4
    parseCsvImperative(csvText)
      ✔ should ignore the header line
      ✔ should have [username, realName, website, projectName] on each contribution
      ✔ should parse username, realName and projectName
      ✔ should set website to null if not provided
    parseCsvFunctional(csvText)
      ✔ should ignore the header line
      ✔ should have [username, realName, website, projectName] on each contribution
      ✔ should parse username, realName and projectName
      ✔ should set website to null if not provided

  ex. 5
    pullAndAnalyzeCsv()
      ✔ should compute various statistics about contributors (1458ms)


  19 passing (1s)
```

FIGURE 1 – Sortie attendue lorsque tous les tests sont verts.