Web Development

Lab1 : JavaScript

- Téléchargez le support du TP1 de moodle.
- Observez l'architecture de l'application (Back Vs Front).

1) Les exercices à faire sont à rajouter dans de dossier « src », chaque exercice doit être codé dans un fichier à part sauf l'exo4 et exo5 qui peuvent être regroupés. Donc dans ce dossier il doit y avoir exo1.js, exo2.js, exo3.js et exo4-exo5.js.

JavaScript: practical activity n°1

Learning outcomes

- Work with functions (variable arguments count, default parameters...) ;
- Build higher-order functions and look at the array's standard methods ;
- Write code without side effect (ex. not mutating the input or global state) ;
- Introduce functional programming paradigm ;
- Create and manipulate literal objects ;
- Chrome DevTools demonstration (debugger, network tracing...).

Exercise 1. Implement sum(...terms)

Implement a `sum` function that accepts any count of numerical arguments and returns the sum. It raises a custom error in the absence of any argument.

```
console.log(sum()) // throws Error('At least one number is expected')
console.log(sum(1)) // prints 1
console.log(sum(1, 2, 3)) // prints 6
```

Exercise 2. Implement filter(array, predicate)

Implement a `filter(array, predicate)` function that returns a new array only containing items from `array` for which `predicate(item)` is truthy. The original `array` should not be mutated (no side effect allowed).

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

After you implemented it, take a look to the native Array.filter.

Exercise 3. Implement map(array, transform)

Implement a `map(array, transform)` function that returns a new array with each `item` from `array` replaced by `transform(item)`. The original `array` should not be mutated (no side effect allowed).

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

2) Pour l'exo4 et exo5 : complétez le code fourni dans le fichier « serveur.js ». Ce petit code permet de : A) récupérer le chemin du fichier de donnée, B) créer une route de type « get » répondant à « /data.csv », cette route permet de transmettre le fichier de donnée comme réponse. C) « app.listen » permet le lancer le serveur. Enfin, n'oubliez pas « node serveur.js » pour lancer le serveur !

After you implemented it, take a look to the native Array.map.

Exercise 4. Basic CSV parsing into literal objects

I share with you a simplified CSV dump of Apache contributors (original source). Because of the size of the file, you cannot simply inline it into your JavaScript code. Instead, I provide you a snippet that pulls the file from my website.

From your perspective, the entry point is the `processData` function.

```
// This code downloads a CSV file from your backend, 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( 'http://localhost:3000/data.csv' )
  .then(res => res.text())
  .then(processData)
  .catch(console.log)

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

For this question, parse `csvText` as an array of literal objects. Each object represents a contribution from someone to an Apache project. Therefore, each object should have the following properties:

- **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 ;

You must provide 2 implementations of the parsing function:

1. **Imperative-style programming**: you can use variables (`let`, `var`...) and `control flows` such (`white`, `for`, `if`...).
2. **Functional-style programming**: you can only use constants (no `let` or `var`) and are not allowed to mutate anything (ex `array[0] = 0` is prohibited). Methods such as `Array.map` are strongly encouraged.

Exercise 5. Computes stats about contributions

Compute and output to console the following metrics. Use of the functional programming paradigm is encouraged.

1. **The first project's name in ascending alphabetic order**. Ensure you compare in a case-insensitive manner and handle diacritics correctly.
2. **The number of unique contributors**. Unicity may be implemented with `array.filter()`.

3. **The average length of contributors' name**. Of course, you have to work on unique names. Remember the DRY principal.
4. **The most active contributor's name** (by number of projects). This is like grouping contribution by contributors' name, sorting by contribution count and eventually taking the first...
5. **TOP 10 of the most contributed projects**. There is again a groupby under the hook. Let's DRY!

Tips: There is not any built-in functions on `Array` to easily take unique values or grouping by criterion. For such processing, I recommend writing helper functions.

Tips: Try not to repeat yourself (DRY principal). Some intermediate computations for a metric could be reused for another one. Also, some logic patterns may be factorised as functions (cf. previous tips).

Bonus: test your code yourself

It is common to write specs (automatic tests) while producing your code, like I did. I share them, so you can test your own implementations. The test runner is mocha, so it runs your code with node.

Tips: of course, you need to adapt my tests, so they call your own code. This mainly depends on your code structure...

Download mocha specs (source code)

```
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)
```

Bonus tips

The previous exercise promoted the writing of helpers like `uniq()` or `groupBy()`. Popular libraries such as lodash implement them already.

Before using them in your frontend projects, think about the weight penalty. At the time of writing, minified size of lodash is about 72kB (to distribute over the network and to execute each time the JS runs in the browser).

In a general manner, lodash's functions cover edge cases that you will probably never meet in your project (which implies extra code). Also, some of them are now obsolete for the latest version of JavaScript.

3) Dans le dossier "tests-mocha", vous disposez du teste de l'exo1 et de l'exo2. Complétez en rajoutant les tests de l'exo3 et l'exo4 et exo5. Donc vous devez créer un fichier « exo3.spec.js » et un autre « exo4-exo5.spec.js ».