Programmation web sur client

TD1.1 - Les bases de JavaScript

Matthieu Nicolas

Préambule

Tout d'abord, dans le fichier `README.md`, indiquez votre nom et prénom (vos noms et prénoms si vous êtes à plusieurs sur une machine).
Le code complété du TP, ainsi que vos notes prises lors des différents exercices vous invitant à étudier des morceaux de codes, sont à rendre via votre repo GitHub. N'hésitez pas à faire des commits et pushs réguliers.

Au cours du TD, je mettrai à disposition de nouveaux supports de cours et exercices. Pour les récupérer, il est nécessaire d'ajouter mon propre repo en tant que repo distant du votre. Pour ce faire, utiliser la commande suivante:

```
git remote add classroom https://github.com/MatthieuNICOLAS/2019-lp-ciasie-prog-web-td1.git
```

Il ne vous restera plus qu'à exécuter la commande `git pull classroom master` lorsque je vous l'indiquerai.

Dans le cadre de ce module, nous aurons besoin de NodeJS, en version 12. Il existe plusieurs manières de l'installer. Pour cela, nous recommandons d'utiliser l'outil `nvm` (`https://github.com/nvm-sh/nvm`), disponible pour Linux et Mac. Il permet d'installer, de gérer plusieurs versions de Node et de passer de l'une à l'autre aisément. Si vous êtes sur Windows, l'outil `nvm-windows` (`https://github.com/coreybutler/nvm-windows`) semble offrir les mêmes fonctionnalités.
Vous pouvez sinon installer Node depuis le site officiel (`https://nodejs.org/fr/`).

Exercice 1 - Variables

Dans le dossier `/1-variables/`, vous trouverez des morceaux de code correspondant à la déclaration et à la manipulation de variables. Ces exemples ont pour but de vous faire comprendre les différences de comportement entre les multiples mots clés disponibles en JavaScript pour déclarer des variables.

Afin de ne pas causer un conflit qui fausserait le résultat, il est important d'exécuter ces morceaux de code de manière totalement isolée.

Pour cela, vous pouvez les exécuter:

- soit dans des onglets différents de votre navigateur
- soit un par un via la commande `node path/to/fichier.js` (recommandé)

Consignes

1. Exécuter les morceaux de codes contenus dans le dossier `/1-variables/1-let/`.
   Observer les différents résultats obtenus.
   Que pouvez-vous en conclure sur le comportement de `let`?

2. Exécuter les morceaux de codes contenus dans le dossier `/1-variables/2-const/`.
   Observer les différents résultats obtenus.
   Que pouvez-vous en conclure sur le comportement de `const`?

3. Exécuter les morceaux de codes contenus dans le dossier `/1-variables/3-var/`.
   Observer les différents résultats obtenus.
   Que pouvez-vous en conclure sur le comportement de `var`?

Remarque

Pour la suite de ce module, nous considérerons que `var` n'existe pas. Son comportement *particulier* a provoqué bien des bugs. Nous utiliserons donc uniquement les mots-clés `let` et `const`, qui ont été ajoutés en **ES6** pour justement remplacer `var`.

Exercice 2 - Échauffement

Dans le fichier `/2-warm-up.js`

1. Écrire la fonction `min`.
   Cette fonction prend en paramètres 2 nombres, `a` et `b`, et retourne le minimum des 2.

2. Écrire la fonction `fizzBuzz`.
   Cette fonction prend en paramètre un entier `n`.
   Elle affiche dans la console "Fizz" si `n` est un multiple de 3, "Buzz" si `n` est un multiple de 5 ou "FizzBuzz" s'il s'agit à la fois d'un multiple de 3 et de 5. Dans le reste des cas, elle se contente d'afficher la valeur de `n`.

3. Écrire la fonction `power`.
   Cette fonction prend en paramètres 2 entiers, `x` et `y`, et retourne x^y. Le calcul du résultat est fait en utilisant une boucle, et non pas en utilisant la fonction `Math.pow`.

4. Écrire la fonction `powerRec`.
   Cette fonction prend en paramètres 2 entiers, `x` et `y`, et retourne x^y. Le résultat est ici calculé à l'aide d'un appel récursif.
   Rappel: $x^y = x * x^{y-1}$.

Exercice 3 - Tableaux

Dans le fichier `3-arrays.js`

1. Écrire la fonction `createArrayFromValues`.
   Cette fonction prend 3 paramètres `a`, `b` et `c`, et retourne un tableau composé de ces 3 valeurs.

2. Écrire la fonction `createRange`.
   Cette fonction prend 2 entiers en paramètre, `a` et `b`, et retourne un tableau contenant tous les entiers de `a` à `b` si `a` <= `b` ou de `b` à `a` sinon.

3. Écrire la fonction `computeSum`.
   Cette fonction prend en paramètre un tableau de nombres `array` et retourne la somme des éléments.

4. Écrire la fonction `computeAvg`.
   Cette fonction prend en paramètre un tableau de nombres `array` et retourne la moyenne de ses éléments. Pour parcourir les éléments du tableau, nous utiliserons une boucle `for ... of` (voir `https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Instructions/for...of`).

Exercice 4 - Objets

Dans le fichier `4-objects.js`

1. Écrire la fonction `computeStats`.
   Cette fonction prend en paramètre un tableau de nombres `array` et retourne un objet ayant les 4 propriétés suivantes : `length`, `min`, `max` et `average`.

2. Écrire la fonction `createPerson`.
   Cette fonction prend 5 paramètres, `lastName`, `firstName`, `yearOfBirth`, `monthOfBirth` et `dayOfBirth`, et retourne un objet ayant les 3 propriétés suivantes : `lastName`, `firstName` et `birthday`. Pour représenter `birthday`, nous utiliserons le type `Date` (voir `https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Date`)

3. Écrire la fonction `displayPerson`.
   Cette fonction prend en paramètre un objet `person` ayant pour propriétés `lastName`, `firstName` et `birthday`, et affiche les informations de l'objet sous la forme suivante "<firstName> <lastName> - <birthday>".

4. Écrire la fonction `setAge`.
   Cette fonction prend en paramètre un objet `person` ayant pour propriétés `lastName`, `firstName` et `birthday`. Elle calcule l'âge de la personne et l'ajoute en tant que nouvelle propriété de l'objet, `age`. Cette fonction ne renvoie aucune valeur.

5. Dans le cadre d'une même exécution, appliquer `setAge` sur un objet `person` défini par vos soins puis afficher son état avec `console.log`.
   Observer le résultat obtenu.
   Qu'en déduisez-vous ?

6. Écrire la fonction `anonymizePerson`.
   Cette fonction prend en paramètre un objet `person` ayant pour propriétés `lastName`, `firstName` et `birthday`, et supprime ses propriétés `lastName` et `firstName`. Pour cela, nous utiliserons l'instruction `delete`.

Exercice 5 - Functions

Dans le fichier `5-functions.js`

1. Écrire la fonction `generateMultiplier`.
   Cette fonction prend un nombre `x` en paramètre et retourne une nouvelle fonction. La fonction retournée accepte elle-même un nombre `y` en paramètre et renvoie $x * y$ comme résultat.

2. Écrire la fonction `filterArray`.
   Cette fonction prend deux paramètres, `array` un tableau et `filterFn` une fonction de filtrage. `filterFn` doit prendre un paramètre et retourner un booléen.
   `mapArray` renvoie un nouveau tableau uniquement composé des éléments de `array` pour lesquels `filterFn` renvoie `true`.
   Exemple:
   `mapArray([1, 2, 3], function isOdd (n) { return n % 2 === 1 }) = [1, 3]`

3. Écrire la fonction `mapArray`.
   Cette fonction prend deux paramètres, `array` un tableau et `transformFn` une fonction de transformation. `transformFn` doit prendre un paramètre et retourner un résultat.
   `mapArray` renvoie un nouveau tableau composé des éléments de `array` auxquels nous aurons appliqué `transformFn`.
   Exemple:
   `mapArray([1, 2, 3], function power2 (n) { return n * n }) = [1, 4, 9]`

Exercice 6 - Égalités

1. Exécuter le morceau de code contenu dans le fichier `/6-equalities/1-coercion.js`.
   Observer les différents résultats obtenus.
   Que pouvez-vous en conclure sur le comportement de `==` et de `===`?

2. Exécuter le morceau de code contenu dans le fichier `/6-equalities/2-zero-zero.js`.
   Observer les différents résultats obtenus.
   Que pouvez-vous en conclure sur le comportement de `==`?

3. Exécuter le morceau de code contenu dans le fichier `/6-equalities/3-NaN.js`.
   Observer les différents résultats obtenus.
   Que pouvez-vous en conclure sur la valeur `NaN`?

4. Exécuter le morceau de code contenu dans le fichier `/6-equalities/4-object.js`.
   Observer les différents résultats obtenus.
   Que pouvez-vous en conclure sur la méthode pour comparer des objets ?

Exercice 7 - Closures

Dans le dossier `7-closures`

1. Exécuter le morceau de code contenu dans le fichier `/7-closures/1-call.js`.
   Observer le résultat obtenu.
   Que pouvez-vous en conclure sur la fonction `c` par rapport à la variable `b`?

2. Exécuter le morceau de code contenu dans le fichier `/7-closures/2-definition.js`.
   Observer le résultat obtenu.
   Que pouvez-vous en conclure sur la fonction `logger` par rapport à la variable `count`?

Exercice 8 - Fonctions avancées

Dans le fichier `8-functions-advanced.js`

1. Écrire la fonction `createSequence`.
   Elle reçoit 2 paramètres : une valeur initiale `init` et une valeur d'incrément `step`, et retourne une fonction qui délivre à chaque appel les valeurs successives de la séquence démarrant à `init` et incrémentées de `step`.

2. Écrire la fonction `createFibonacci`.
   Elle permet de parcourir la suite de Fibonacci. La fonction reçoit 2 arguments qui sont les 2 valeurs initiales de la suite, et retourne une fonction qui, à chaque appel, délivre les valeurs successives de la suite.
   Rappel : la suite de fibonacci est la suite $u_n = u_{n-1} + u_{n-2}$