Sainte-Norme du JavaScript

Dont le titre est écrit en taille 42

Groupe SportsFun

Version 0.1.0

Rédigé par :

Sylvain Garant,

Avant propos

La présente norme est avant tout une proposition, il est impératif que la majorité des membres du groupe soient d'accord ou finissent par être convaincus par l'utilité de celle-ci et d'un nombre maximum de ses points (autrement ça sert à rien…).

Je sais que certains points vont faire jaser dans les chaumières, il faudra néanmoins que vous gardiez en tête que, avec toute l'humilité du monde, je suis la personne pro-norme ayant le plus d'expérience en JavaScript du groupe, les autres étant plus habitués à la norme C ça risque de vous faire tout drôle. Donc gardez ces données en tête et entamez la lecture de cette norme avec ouverture d'esprit, vous allez vous y faire, et vous vous rendrez compte plus tard que cette façon de procéder aide beaucoup à la lecture (et donc à la maintenabilité) d'un langage tel que le JavaScript ; car il s'agit d'un langage tellement permissif qu'il est très aisé de finir par faire n'importe quoi, et qu'on se retrouve avec un joli code spaghetti al dente. Après certains petits points ne sont que des préférences personnelles qu'on peut changer si ça ne convient pas à tout le monde, bien entendu.

Néanmoins, j'ai commencé l'écriture de cette norme à la place d'un cours de droit, j'approche actuellement des deux heures de rédaction, il est donc évident que cette norme n'est pas exhaustive, et, comme marqué sur la page de garde, elle est soumise à un versionnage. C'est-à-dire qu'on la mettra à jour en fonction de vos retours, et tout au long des deux ans et demi de l'EIP.

Aussi si quelqu' un aime le front-end et veut la rendre plus jolie, allez-y, c'est pas moi qui vais vous en empêcher.

Norme

1. Généralités

- Les fichiers doivent posséder une entête donnant le nom de la personne l'ayant créé (le format de celui-ci est laissé à la liberté de l'imagination de l'auteur, tant qu'il renseigne son nom et que le header ne fait pas un nombre indécent de lignes)

((Cas particulier : Ben doit inclure dans son entête le num de sa sœur))

- Il faut utiliser l'ES6 à chaque fois qu'il est possible, par exemple :

- Lors d'un passage d'une fonction callback dans un appel de fonction

> fonctionLol(42, 'prout', (param1, param2) => { console.log('zizi violet'); });

- Lors d'une déclaration de variable

> let maVariable;

- Lors de la création d'un objet, avec les mots clés associés (voir class, constructor & compagnie)

- Lors de définition d'objet quelconque, la clé de chaque valeur NE DOIT PAS être un string :

> let obj = { maCle: 'lol' };

Et non pas :

> let obj = { 'maCle': 'lol' };

Renseignez-vous sur l'ES6, c'est la version de 2015, GrafikArt a fait une vidéo à ce sujet :

<https://www.youtube.com/watch?v=5146X8FSBUQ>

Il n'est en revanche pas nécessaire de supprimer la clé d'une déclaration de fonction à l'intérieur d'un objet, par exemple :

> let obj = {

> maFonction() { /\* Faire des trucs \*/ }

> };

Ceci est valable en ES6, mais il est plus lisible de faire :

> let obj = {

> maFonction: function() { /\* Faire des trucs \*/ }

> };

- Les tabulations et indentations en général sont de 4 espaces (configuration par défaut dans WebStorm)

- Delete les variables 'newées' dans la mesure du possible :

> let truc = new THREE.NobodyIsGoingToReadThatLol() ;

> delete truc;

- Limiter, dans la mesure du possible, l'utilisation de variables globales. La déclaration accidentelle de variables globales devrait être endiguée par l'utilisation du mot clé 'let' de l'ES6.

Au cas où il n'est pas possible de faire autrement (et ça va être beaucoup plus fréquemment le cas qu'en C/C++), donner un nom distinctif à la variable, de manière à ne pas entrer en confusion dans d'autres morceaux de code :

Exemple banco :

> var mainCamera;

Exemple pas banco :

> var i;

- Un grand nombre de points de cette norme héritent de la norme C de l'école, comme le fait de mettre un espace après un mot clé du langage :

> if (truc)

Et non pas :

> if(truc)

Il faut également mettre des parenthèses autour de la valeur de retour d'une fonction :

> return (truc);

Et non pas :

> return truc;

Pareil autour des opérateurs :

> let variable = 1 + 1;

Et non pas :

> let variable = 1+1;

Vous avez compris.

PS : Respectez également dans la mesure du possible l'indentation de la norme C.

PS2 : Codez en anglais.

2. Déclaration

- Les strings s'écrivent avec de simples quotes :

> let maString = 'prout';

- Les variables doivent être initialisées avant utilisation :

> let variable2fou = 42;

- Lors de définition d'objet inline, il doit y avoir un espace après l'accolade d'ouverture, avant l'accolade de fermeture, après chaque ':' et chaque ',' :

> let truc = { poil: 'jaune', casquette: 'alenver' };

- Lors de définition d'objet sur plusieurs lignes, c'est sensiblement la même chose, mais en appliquant la règles des accolades :

> let truc = {

> poil: 'jaune',

> casquette: 'alenver'

> };

À noter que l'accolade fermante est alignée avec le début de la ligne de définition (le } est à la même colonne que le 'l' de 'let'), et que les champs définis à l'intérieur de l'objet ne sont décalés que d'une indentation.

- Le nom des variables doit être écrit en lowerCamelCase :

> let maVariableAuNomBeaucoupTropLong;

- Le nom des fonctions ne faisant pas partie d'un objet doit être écrit en snake\_case :

> function ma\_fonction\_au\_nom\_pas\_moins\_long\_que\_la\_variable\_precedante();

- Le nom des fonctions faisant partie d'un objet doit être écrit en UpperCamelCase :

> function MaFonctionVoilaVousAvezCompris();

- Éviter, dans la mesure du possible, de ne pas mettre de mot clé lors d'une déclaration de variable dans une fonction :

> function() {

> truc = 42;

> }

Ceci fonctionne, mais 'truc' devient une variable globale, et comme ce n'est pas toujours ce qu'on souhaite (quand bien même ce soit le cas, réfléchissez pour faire autrement, ou reportez-vous au point sur les variables globales), il est préférables de procéder comme suit :

> function() {

> let truc = 42;

> }

3. Structure

- Les accolades ouvrantes sont en fin de ligne, précédées par un espace :

> if (salut == 'c'est cool') {

> console.log('mdr');

> }

- Lorsqu'un bloc ne contient qu'une instruction, ne pas mettre d'accolade :

> if (machin)

> truc;

Cette règle s'applique aussi pour les fonctions anonymes de l'ES6 :

> (param1, param2) => console.log('cote de porc');

- Pour aller plus loin avec le point juste au dessus, lorsqu'il n'y a qu'un paramètre pour une fonction anonyme ES6, ne pas mettre de parenthèses :

> param => truc();

Et lorsqu'il n'y a pas de paramètre, mettre un underscore :

> \_ => truc();

- Lors de l'implémentation d'une fonction de plus de 10 lignes, il doit y avoir un espace au début et à la fin du bloc, de cette manière, le code reste aéré :

> fonction fouBarre() {

>

> console.log('ptdr');

> console.log('ptdr');

> console.log('ptdr');

> console.log('ptdr');

> console.log('ptdr');

> console.log('ptdr');

>

> }

Ça n'empêche pas les gens qui veulent être propres d'appliquer cette consignes pour les fonctions de moins de 10 lignes.

Il est possible de ne pas appliquer cette règle s'il s'agit d'une fonction utilitaire, de debug ou à laquelle vous êtes normalement le seul à avoir accès si ça ne vous plaît pas de procéder comme ça, on n'est pas des bêtes.

- Lors de l'implémentation d'une fonction de plus de 25 lignes, regrouper les instructions cohérentes entre elles et séparer les autres par un saut de ligne, et dans l'idéal mettre un commentaire qui nomme la partie :

> function init() {

>

> // Scene

> let scene = new THREE.Scene();

> scene.background = new THREE.Color(0x000000);

>

> // Camera

> let cam = new THREE.PerspectiveCamera(0, 0, 0, 0);

> cam.position.z = -15;

> scene.add(cam);

>

> // Light

> createLights();

>

> // … ETC

> }

Il ne faut pas oublier que le JavaSCRIPT est un langage de SCRIPTing, procéder comme ça n'est donc pas « sale », comme pourraient dire les rageux.

4. Bonnes pratiques

- Court-circuitez vos algorithmes le plus possible, en utilisant par exemple des 'return', 'break' ou 'continue' dans vos boules ou autre structures de contrôle. Le JS étant un langage de scripting, ce procédé peut faire gagner un temps de calcul non négligeable à la longue :

> for (cest\_tipar\_pour\_1\_milliard\_de\_tours\_de\_boucle) {

> if (break\_plus\_rekt\_egal\_brekt)

> break;

> }

5. Base de données

Cette section va rester vide pour le moment, nous n'avons pas besoin de base de donnée pour le POC. Il nous en faudra néanmoins une pour le projet post-POC, je créé donc cette partie car il semble assez évident que nous allons utiliser MongoDB, et, d'expérience, si on ne pose pas des limites dès le début, ça va finir par être n'importe quoi, et je ne suis pas un grand fan des librairies de schéma (comme Mongoose) qui brident le potentiel de Mongo (si on utilise ce genre d'outil alors là c'est la fête du slip, autant utiliser le SQL direct tant qu'on y est, tiens !)

On mettra donc ici les règles qu'il faudra appliquer pour respecter l'architecture de la DB qu'on créera plus tard, au moment de la conception du programme.