L’entrainement de Monseigneur Node – Partie 1

Version 4.5





1. Introduction

Monseigneur Node est un honorable prêtre *JavaScript* de Villeville qui a créé sa propre branche : *NodeJS*. Ayant voué sa vie à répandre sa vision de la sainte parole du JavaScript au travers de ses nombreux écrits. Sa réputation est désormais connue des contrées de l’Islande jusqu’aux confins de la Bulgarie. Respecté et honoré, Monseigneur Node vécut heureux, jusqu’à ce jour fatidique où il découvrit un cheveu gris.

Il comprit alors que le temps de passer le flambeau était arrivé et donc qu’il devrait trouver un successeur. Alors que vous faisiez le tour des stands du marché, vous entendez alors : “C’EST MON DISCIPLE !”. En effet, c’est vous qu’il désignait. Il vous amena donc jusqu’à son lieu de culte où il vous présenta la sainte documentation du NodeJS.

Votre initiation commence dès maintenant.

1. Consignes

* Pour ce projet-là, il vous sera demandé de créer un repository avec le nom : cc\_entrainement\_de\_monseigneur\_node.
* N’oubliez pas de push régulièrement !
* En cas de question, pensez à demander de l’aide à votre voisin de droite. Puis de gauche. Ou inversement. Puis demandez enfin à un Cobra si vous êtes toujours bloqué(e).
* Pensez à faire valider chaque partie que vous réaliserez à un Cobra lorsque vous l’aurez terminée.
* N’hésitez pas à faire des bonus et à ajouter des fonctionnalités lorsque votre projet sera terminé et validé.
* Vous avez tout à fait le droit d’utiliser internet pour trouver des réponses ou pour vous renseigner.

1. Installation du framework

Afin de pratiquer au mieux, Monseigneur Node utilise des outils bien étranges. Il appelle cela “[framework](https://fr.wikipedia.org/wiki/Framework)” et “terminal”. Ses outils lui permettent en tant que prêtre de prêcher sa religion.

Afin de mieux vous imprégner de celle-ci, il y a une prière particulière à faire qui dépend de quelle caste vous appartenez. Même si vous ne comprenez pas le sens de ces prières, Monseigneur Node souhaite que vous les connaissiez.

Si vous appartenez à la caste *Windows* ou *MacOS* :

Il vous suffit dans un premier temps d’aller sur le site de [NodeJS](https://nodejs.org/fr/) . Cliquez sur downloads puis choisissez le système qui correspond au vôtre s’il n’est pas déjà proposé. Ensuite, vous n’aurez qu’à suivre le processus d’installation.

Si vous appartenez à la caste *Linux* :

Il vous suffira d’utiliser votre gestionnaire de paquets habituel en précisant que c’est le paquet “nodejs”que vous souhaitez.

Maintenant que vous avez appris ces prières, il va vous falloir utiliser l’outil dénommé “terminal”. Celui-ci vous permet de faire beaucoup de choses, dont de créer un dossier de travail et un document “.js”dans lequel vous pourrez réécrire la sainte documentation. Pour être sûr d’avoir bien compris ce que vous avez écrit, vous pouvez ouvrir aussi le terminal avec des commandes “magiques” : “CMD” pour la caste Windows, “Terminal”pour les castes Linux et MacOS.

À nouveau, ces commandes dépendant de votre caste. Monseigneur Node vous les a notés sur une feuille mémo qui est la suivante :

|  |  |  |
| --- | --- | --- |
|  | Windows | Linux et Mac |
| Créer un fichier | Possibilité de le faire via l’éditeur de texte | Dépend de votre éditeur de texte |
| Se déplacer dans les dossiers | cd dossier1**\**dossier2**\**dossier3 | cd dossier1**/**dossier2**/**dossier3 |
| Se déplacer en arrière dans les dossiers | cd ..**\**..**\**.. | cd ..**/**..**/**.. |
| Afficher les fichiers du dossier courant | dir | ls |

Pensez bien à créer un document et à vous rendre dans le dossier où se trouve celui-ci avec votre terminal pour la bonne continuation de votre apprentissage.

Afin de vérifier votre bonne compréhension de la sainte documentation vous devez utiliser le terminal. Il faudra utiliser la commande suivante afin de “lancer” votre document :

node votreDocument.js

1. Les variables

Le premier souvenir que vous avez de cette initiation au NodeJS, c’est de vous être fait taper sur les doigts pour avoir osé dire que le Java et le JavaScript “Bah ça revient au même non ?”. Maintenant vous saurez que ce sont deux religions bien différentes, bon sang de bonsoir !

Maintenant, passons aux choses sérieuses : parmi les bases qui vous seront nécessaires il y a les variables. Monseigneur Node aimerait vous faire découvrir les différents types, la portée des variables, les différents types d’affichages et la conversion de types.

* 1. Les types

Monseigneur Node vous explique dans un premier temps que le *JavaScript* n'impose pas de définir le type d'une variable. Ce type est déduit de la valeur stockée dans la variable et peut donc changer au fur et à mesure de l'exécution du programme : on dit que JavaScript est un langage à typage dynamique.

Monseigneur Node vous explique que pour déclarer une variable, il vous suffit d’écrire le mot-clé "let” suivi du nom de la variable.

let A;

Maintenant que vous avez une variable, Monseigneur Node vous explique comment l’afficher. Pour cela, vous utiliserez “console.log()” . Vous devriez donc avoir :

let A;

console.log(A);

L’étape suivante qu’il vous demande, c’est de lancer votre fichier avec ce bout de code et regardez alors la valeur qui vous est affichée. Vous devriez avoir “undefined” affiché dans le terminal. En effet, vous avez défini la variable A mais vous ne l’avez pas initialisé. Si vous souhaitez que A soit un nombre nous pouvons faire par exemple :

let A = 42; //pour un int (entier)

let A = 4.2; //pout un float (nombre à virgule)

let A = getRandomValue() //pour renvoyer une valeur aléatoire

⚠ **Attention** : *getRandomValue()* n’est pas une vraie fonction, c’est un exemple !

Monseigneur Node vous explique une variable peut aussi être le résultat d’une opération. Par exemple :

let A = (42 + “0”) / (4\*\*2);

console.log(A);

Le résultat affiché sera “26.25”. Mais il vous explique que faire cela est péché. Car additionner un int avec une string et diviser le tout par 4 puissances 2 (donc au carré). Vous êtes fortement encouragés à aller vous renseigner plus tard de pourquoi c’est possible et qu’est-ce qui se passe exactement et pourquoi ça reste un mal de faire ça.

Dite à Monseigneur Node ce que devrait afficher chacune des lignes suivantes :

console.log(4 + 2):

console.log(“4 + 2”):

console.log(“4” + “2”):

Voici le résultat que vous devez obtenir :

6

4 + 2

42

Pourquoi demanderez-vous sûrement ?

Monseigneur Node vous explique que dans le premier cas, on va additionner des int donc simple opération mathématique. Dans le deuxième cas, on demande à afficher le contenu d’une phrase, soit une string. Dans le dernier cas, on fait une opération de concaténation de string.

Mais on peut aussi déclarer une variable constante. Pour cela, il suffit de rentrer le mot-clé “const”. Donc quand vous déclarez la variable, vous devez ensuite l’initialiser, car vous ne pourrez pas changer sa valeur plus tard. Si vous le faites, l’interpréteur vous signalera que vous essayez de faire une action impossible.

const a = 0;

a= 1;

console.log(a);

Vous aurez :

a = 1:

^

TypeError: Assignment to constant variable.

Pour résumer :

on déclare une variable avec “let” ou “const”: let A

un nombre “int” s’écrit tel quel : let A = 42

* une phrase “string” s'écrit entre guillemets : let A = “Hello”
* une caractère “char” s’écrit entre apostrophes: let A = ‘E’
* un tableau “array” s’écrit avec crochets : let A = [0, 1, 2, 3, 4, 5]
* un booléen “bool” se définit par vrai “true” ou faux “false” : let A = true ou let A = false
* ⚠ **Attention** : Dans la religion du JavaScript, les tableaux commencent à 0. Donc pour récupérer le premier élément d’un tableau, il faut demander l’index 0 ! Pour accéder à l’index d’un tableau il suffit de faire nomDeLaVariable[index].

Voici comment se découpe une string :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| String | ‘H’ | ‘e’ | ‘l’ | ‘l’ | ‘o’ |
| Index dans la string | 0 | 1 | 2 | 3 | 4 |

* 1. La portée des variables

Monseigneur Node veut maintenant vous expliquer la “portée” ou autrement appelé : scope. On appelle **“**scope”d'une variable la portion du code dans laquelle cette variable est visible et donc utilisable, vous informe Monseigneur Node. Les variables déclarées avec “let” et “const” ont une portée de type bloc : elles ne sont visibles qu'au sein du bloc de code dans lequel elles sont déclarées, ainsi que dans tous les sous-blocs éventuels.

Dans de nombreuses autres religions, un bloc de code est délimité par une paire d'accolades ouvrantes et fermantes.

let num1 = 0:

{

num1 = 1; // OK : num1 est déclarée dans le bloc parent

const num2 = 0;

}

console.log(num1); // 1

console.log(num2): // Erreur : num2 n’est pas visible ici

* 1. Différents types d’affichages

Monseigneur Node va vous enseigner les types d’affichages qui existent. Ici vous verrez différentes manières d’afficher un texte avec des valeurs. Monseigneur Node va commencer par vous montrer les bases.

let A = (42 + “0”) / (4 \*\* 2);

Dans le JavaScript, on peut décider d’afficher une string "J’ai" à laquelle on ajoute la valeur de la variable désirée “A” et à laquelle on ajoute à nouveau une string"€"**.** Ce qui donnerait :

console.log(“J’ai” + A + “€”);

Maintenant, admettons qu’on ne veuille pas faire des ajouts successifs comme ça. Pour cela dans plusieurs religions, ils utilisent une paire d'accents graves seuls ou backticks. C’est ce caractère : **`** ce n’est pas une apostrophe ! Et afin de dire quelle variable on veut afficher, il suffit d’écrire votre variable là-dedans ${}. Ce qui donnerait :

console.log(‘J’ai ${A} €’);

Comme Monseigneur Node vous a expliqué le résultat sera le même, mais c’est juste le type d’affichage qui n’est pas pareil. Pour faire ces opérations, Monseigneur Node utilise aussi cela, par exemple :

const x = 3:

const y = 7;

console.log(‘${x} + ${y} = ${x + y}’);

console.log(x + “ + “ + y + “ = “ + (x + y));

Ce qui affichera "3 + 7 = 10”.

* 1. Conversion de types

Similairement aux autres religions, le JavaScript accepte les conversions. Ce que Monseigneur Node souhaite vous expliquer ici, c’est qu’il est possible de transformer une variable d’un certain type en un autre.

Par exemple, si vous définissez une variable “A”qui prendra comme valeur “42”donc un int. Ensuite vous lui assignez le retour de la fonction “String” à laquelle vous donnez “A’ en paramètre. Et que vous affichez le type de ‘A”, ça vous affichera le type de la variable grâce à la fonction “typeof()”.

D’après lui, niveau code vous devriez avoir quelque chose comme ce qui suit :

let A = 42;

A = String(A);

console.log(typeof(A));

Vous pouvez faire la même chose, mais avec la fonction “Number()” qui transformera votre variable en nombre.

⚠ **Attention** : Selon le type d’avant de votre variable, ça peut ne pas fonctionner voire crasher.

1. Instruction, boucles et conditions

Jusqu'à présent, toutes les instructions de vos programmes, à part les commentaires, étaient systématiquement exécutées. Monseigneur Node souhaite désormais vous instruire afin d’enrichir vos programmes en y ajoutant des possibilités d'exécution conditionnelle.

Quand il parlera des instructions “if” et “else” qui veulent dire “si” et “sinon” en français. Lorsqu’il parlera des boucles “while” et “for” soit “tant que” et “pour” en français.

* 1. Les opérateurs

Il y a plusieurs opérateurs qui vous permettent de dire qu’une condition est validée ou non, et que Monseigneur Node aimerait que vous ayez connaissance :

|  |  |
| --- | --- |
| **Opérateurs** | **Signification** |
| === | Strictement égal à |
| == | Égal à |
| !== | Strictement différent de |
| != | Différent à |
| < | Inférieur à |
| <= | Inférieur ou égal à |
| > | Supérieur à |
| >= | Supérieur ou égal à |
| && | Et |
| || | Ou |

Pour plus d’explication sur les différentes égalités, vous pouvez vous renseigner [ici](https://developer.mozilla.org/fr/docs/Web/JavaScript/Les_diff%C3%A9rents_tests_d_%C3%A9galit%C3%A9) d’après Monseigneur Node.

⚠ **Attention** : La confusion entre l'opérateur d'égalité **===** (ou **==**) et l'opérateur d'affectation **=** dans l'écriture d'une condition est une erreur très fréquente.

* 1. L’instruction if

En JavaScript et dans beaucoup d’autres religions, l’instruction “if” se déclare de la manière suivante :

if (condition) { // si la condition est vraie

//alors exécuter le code de ce bloc

}

Bien entendu, il peut y avoir plusieurs conditions à valider pour entrer dans le sous-bloc ! Souvenez-vous des opérateurs “&&” et “||”. Il est d’ailleurs possible d’inverser une condition en mettant un **!** devant la condition.

Maintenant que vous avez pris connaissance de comment créer une condition avec l’instruction if, Monseigneur Node aimerait bien vous faire utiliser le “else” ! Dans le cas du else, nous il n’y a pas de parenthèses pour définir une condition, juste des accolades car la condition a déjà été vérifiée et elle était fausse.

Vous pouvez aussi faire une autre condition après un else. Comme ceci :

if (condition1) {

// code

} else if (condition2) {

// autre code

}

* 1. Définissez les couleurs

Monseigneur Node voudrait que vous définissiez une variable couleur et que vous lui assignez la valeur alphabétique de votre choix (string). Dans le cas où la couleur est *bleu* affichez "La couleur est bleu", si la couleur est *rouge* affichez "La couleur est rouge", et pareil pour *vert* et *jaune*. Et si ce n’est aucun des cas, afficher "La couleur était : " suivi de la couleur en question.

let couleur = /\* mettez une couleur ici \*/;

if (couleur === “bleu”) {

console.log(“La couleur est bleu”);

} else if (couleur === “rouge”) {

console.log(“La couleur est rouge”);

} else if (couleur === “vert”) {

console.log(“La couleur est vert”);

} else if (couleur === “jaune”) {

console.log(“La couleur est jaune”);

} else {

console.log(‘La couleur était: ${couleur}’);

}

Cette succession de if else est appelée "forêt de if". Et il y a une manière plus propre de le faire, mais Monseigneur Node vous en parlera un peu plus loin !

* 1. Les boucles

Comme Monseigneur Node avait introduit plus haut, il y a deux boucles différentes : “while” et “for”. La principale différence entre les deux se trouve dans leur déclaration. Mais la boucle for a un truc en plus : l’optimisation. En effet, sur des petits programmes on ne verra pas la différence mais sur de gros programmes avec des calculs compliqués, Monseigneur Node privilégie d’utiliser des boucles “for”. La boucle “while” s’écrit :

while (condition) {

//code et actualisation de la variable de référence pour la condition d’arrêt

}

Cela peut donner :

let A = 0;

while (A < 10) {

console.log(A);

A++;

} // après ces tours de boucles, A vaudra 9

La boucle for s’écrit :

for (/\* assignation de valeur \*/; /\*condition\*/; /\*incrémentation \*/ ) {

//code

}

Cas concret :

let A = 10;

for (let B = 0; B < A; B++) {

console.log(B);

}

Ici, plutôt que de déclarer B on aurait pu assigner la valeur 0 à “A”, changer la condition par “A < 10” et changer l’incrémentation par “A++” et bien évidement afficher “A”, pas “B”. Par ailleurs, si vous essayez d’afficher “B” hors du for, vous ne pourrez pas car son scope se limite aux accolades du for (car déclaré à l’intérieur).

1. Les fonctions

Monseigneur Node aime beaucoup utiliser des fonctions lorsqu’il édite les saints documents et il va vous apprendre à bien les comprendre et les utiliser.

Tout d’abord, qu’est-ce qu’une fonction ? Comme dans toutes les religions, une fonction c’est un bloc de code qui a pour vocation d’être appelé plusieurs fois ou simplement à simplifier la lecture du code. Par exemple la fonction “log()” de l’objet “console” que vous utilisiez depuis le début dans le but d’afficher quelque chose dans votre terminal.

Comment on déclare une fonction demandez-vous à Monseigneur Node. Il vous répond qu’il existe plusieurs façons. Monseigneur Node va commencer par vous montrer la manière la plus traditionnelle !

* 1. Les fonctions traditionnelles

Pour écrire une fonction de manière traditionnelle, il faut s’y prendre de la façon suivante :

function nomDeLaFonction(/\*les arguments s’il y’en a\*/) {

//instructions de la fonction

}

Lorsque vous donnez des arguments à la fonction, en JavaScript, ne donnez que les noms des variables, pas besoin de mettre “let”.

Monseigneur Node voudrait vous faire savoir une chose importante : une fonction peut renvoyer une valeur. Elle peut faire ça grâce au mot-clé “return”. Pour que la fonction puisse renvoyer quelque chose il faut faire suivre le mot-clé par ce quelque chose. Sinon, s’il n’est suivi de rien il permettra de quitter la fonction car le retour sera implicite par la valeur “undefined”. Exemple :

function myReturn0() {

return (0);

}

function myReturnNothing() {

return;

}

Monseigneur Node vous précise aussi que les parenthèses autour de la valeur de retour ne sont pas toujours nécessaires. Elles le sont lorsque vous renvoyez une opération (concaténation de strings ou opération mathématique par exemple).

Pour afficher le retour d’une fonction, vous pouvez faire comme ci-dessous :

console.log(myReturn0());

Ce qui affichera 0, ici.

Monseigneur Node souhaite être sûr que son disciple ai bien compris, il voudrait que vous fassiez une fonction qui prendra deux paramètres, elle fera une addition de ces deux-là et renverra le résultat. Evidemment, affichez le retour de cette fonction !

Deux réponses possible :

function myAdditionV1(val1, val2) {

return(val1 + val2);

}

function myAdditionV2(val1, val2) {

let res = val1 + val2;

return(res);

}

console.log(myAdditionV1(1, 1));

console.log(myAdditionV2(1, 1));

* 1. Les fonctions anonymes et fléchées

Monseigneur Node a aussi l’habitude d’utiliser des fonctions anonymes lorsqu’il ne souhaite utiliser une fonction qu’une seule fois. Mais qu’est-ce que c’est ? Et bien c’est une fonction comme son nom l'indique qui n’a pas de nom… Elle a cependant quand même le mot function au départ et qui est suivi des parenthèses avec les arguments !

Vous demandez à Monseigneur Node comment vous pouvez déclarer une fonction anonyme et il vous montre de la manière suivante :

// Affectation d’une fonction anonyme à la variable maVariable

const maVariable = function(param1, param2, …) {

// Instructions pouvant utiliser param1, param2, ...

}

// Appel de la fonction anonyme

// param1 reçoit la valeur de arg1, param2 la valeur de arg2, …

maVariable(arg1, arg2, …);

La fonction créée ci-dessus est anonyme et directement affectée à la variable "maVariable". La valeur de cette variable est donc une fonction. Cette manière de créer une fonction est appelée expression de fonction.

Pour vous illustrer un peu plus le fonctionnement, Monseigneur Node vous fait un exemple d’utilisation :

const maVariable = function (param1, param2) {

return (param1 + param2);

}

console.log(maVariable(1, 1));

console.log(maVariable(2, 2));

Ce qui nous affichera 2 puis 4.

Monseigneur Node durant sa vie contribuer à la découverte de fonctionnalités par avant méconnues qui ont introduit une syntaxe plus concise pour créer des fonctions anonymes. L'exemple suivant est strictement équivalent au précédent, mais il s’agit cette fois de fonction fléchée.

// Affectation d’une fonction anonyme à la variable maVariable

const maVariable = (param1, param2, ...) => {

// Instructions pouvant utiliser param1, param2, ...

}

// Appel de la fonction anonyme

// param1 reçoit la valeur de arg1, param2 la valeur de arg2, …

maVariable(arg1, arg2, …);

Et si on reprend l’exemple précédent, ça donnerait cette fois :

const maVariable = (param1, param2) => {

return(aram1 + param2);

}

console.log(maVariable(1, 1));

console.log(maVariable(2,2));

Dans certains cas particuliers, on peut simplifier la syntaxe des fonctions fléchées. C’est-à-dire, lorsque le corps de la fonction se limite à une seule ligne, on peut écrire son résultat sans créer de blocs de code avec des accolades. Dans ce cas, l'instruction “return” est implicite. Mais aussi lorsque la fonction n'a qu'un seul argument, on peut omettre les parenthèses autour de celui-ci.

const bonjour = prenom => `Bonjour, ${prenom} !`;

console.log(bonjour(“Thomas”));

Vous vous souvenez du dernier exercice sur les couleurs ?

Cette fois, Monseigneur Node voudrait que son disciple fasse une fonction qui prenne en paramètre un tableau de couleurs. Et il faut que cette fonction renvoie une string qui dit à quelle couleur correspond à quel index.Petite spécificité, Monseigneur Node demande que ce soit une fonction anonyme (voire fléchée si vous voulez).

Cela devrait donner quelque chose comme ça :

const colorsToIndex = array => {

let res = “”;

for (let i = 0; i < array.lengh; i++) {

res += `Il y a la coulour ${array[I]} à l’index ${I}!`;

if (i !== array.length - 1) { // si notre index est différent de la longueur du tableau –1

res += ‘\n’; // on ajoute le caractère ‘\n’ (retour à la ligne)

}

}

return (res);

}

console.log(colorsToIndex([“bleu”, “rouge”, “jaune”, “vert”, “violet”, “blanc”]));

1. Introduction au “switch case”

Monseigneur Node décide de vous faire éviter les forêts de if. Pour cela, il vous instruit sur le switch case. C’est une instruction qui fonctionne un peu comme if else et qui permet de tester plusieurs valeurs dans un seul bloc.

let i = 7;

switch (I) { // i est la variable don’t on va regarder la valeur

case 0:

console.log(0)

break; // mot-clé qui permet de sortir de ce cas pour tester ensuite les autres

case 1:

console.log(1);

break;

case 2:

console.log(3);

break;

case 3:

console.log(4);

break;

case 4:

console.log(5);

break;

default:

console.log(“(╯°□°)╯︵ ┻━┻");

}

Reprenez l’exercice des couleurs de tout à l’heure et modifiez le pour introduire un switch case. Commencez par créer une variable qui sera un tableau de couleurs et une bouche for qui va itérer sur chaque couleur. Dans celle-ci, introduisez un switch case qui regardera quelle est la couleur à l’index, puis afficher `La couleur à l’index ${index} est : ` suivi de la couleur. Sinon afficher `La couleur [la couleur] n’était pas attendue`

1. Conclusion

Voilà c’est la fin, Monseigneur Node est fier de votre travail et est heureux de voir que la relève est assurée ! Cependant, Monseigneur Node est sûr qu’il reste encore plein d’autres aspects à découvrir ! N’hésitez pas à vous renseigner par vous-même.

Par ailleurs, s’il vous reste du temps, pourquoi n’essayeriez-vous pas de faire la partie suivante, pour cela demandez à un Cobra de s’il/elle peut vous la fournir.



Félicitations, vous pouvez passer à la partie suivante