L’entrainement de Monseigneur Node – Partie 2

version 4.5





1. Introduction

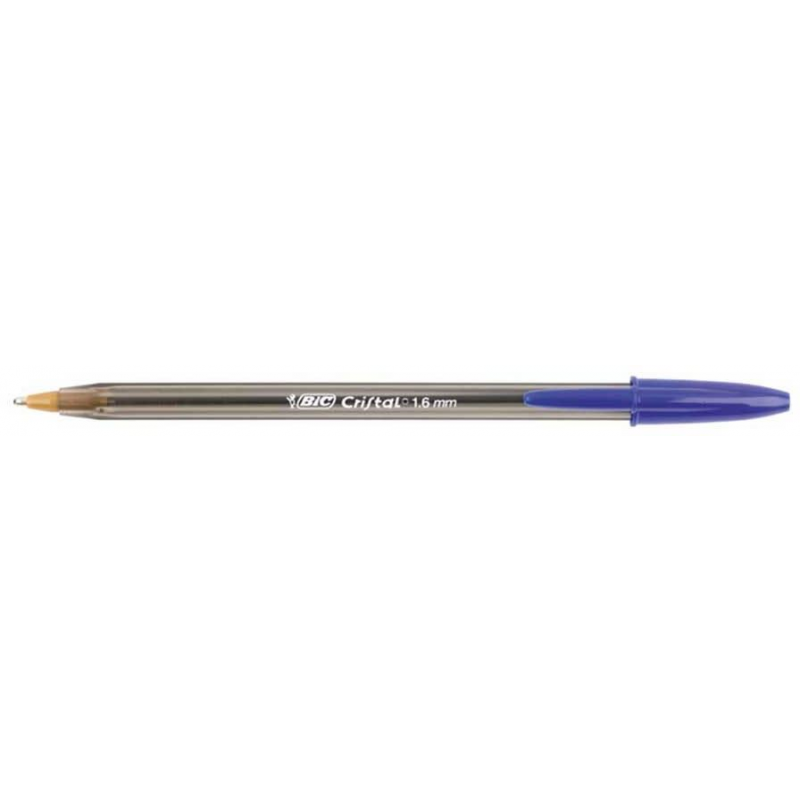
Monseigneur Node est fier de votre dynamisme et de votre détermination. Il est désormais sûr que vous êtes prêts à découvrir cette deuxième partie. Préparez-vous, de nouvelles aventures s’offrent à vous !

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. À la découverte des objets

Monseigneur Node prend un exemple très concret : un stylo. Il vous demande d’étudier ce stylo : quelles sont ses caractéristiques ? Sa couleur d'écriture, son type, son fabricant... Notre stylo possède un certain nombre de propriétés qui le caractérisent.



Le stylo à étudier

Qu'est-ce qu'un objet en JavaScript ? Tout comme ce stylo, un objet en JavaScript est une entité qui possède des propriétés. Chacune définit une caractéristique de l'objet. Une propriété peut être une information, une action ou un autre objet. Par exemple un stylo 4 couleurs est composé de 4 stylos 1 couleur.

Mais comment est-ce qu’on déclare un objet ? Pour garder l’exemple du stylo, Monseigneur Node vous montre cela :

const stylo = {

type: "bille",

couleur: "bleu",

marque: "Bic",

};

En JavaScript, comme dans certains autres langages, on peut créer un objet en définissant ses propriétés à l'intérieur d'une paire d'accolades.

Vous avez maintenant un objet avec plusieurs caractéristiques, mais comment accéder à ces dernières ?

Pour afficher quelque chose, il faut utiliser "console.log()". Consoleest un objet dans lequel il y a une méthode, c’est comme ça qu’on nomme une fonction qui appartient à un objet, qui est "log()". Pour accéder aux valeurs, il faut appeler "nomObjet.caractéristique" ou "nomObjet.méthode()".

console.log(stylo.type);  
console.log(stylo.couleur);  
console.log(stylo.marque);

Une fois un objet créé, on peut modifier les valeurs de ses propriétés avec la syntaxe "nomObjet.caractéristique = nouvelleValeur;"

JavaScript offre même la possibilité d'ajouter dynamiquement de nouvelles propriétés à un objet déjà créé.

const stylo = {

type: "bille",

couleur: "bleu",

marque: "Bic"

};

// Ajout de la propriété "prix"

stylo.prix = "2.5";

console.log("Mon stylo coûte " + stylo.prix + " euros");

Maintenant que vous savez créer un objet avec des caractéristiques, vous allez lui ajouter une méthode ! En gardant l’exemple du stylo, Monseigneur Node souhaite que vous ajoutiez une méthode qui permettra à l’objet de se décrire.

const stylo = {

type: "bille",

couleur: "bleu",

marque: "Bic",

description() {

return ("Je suis un stylo " + this.type + " qui écrit en " + this.couleur + " de la marque " + this.marque + ".");  
 }  
};

Pour afficher le retour de cette méthode, il faudra comme s’en suit :

console.log(stylo.description());

Monseigneur Node vient de se rendre compte qu’il ne vous a pas expliqué ce qu’est le mot-clé "this"! Ce mot clé est défini automatiquement par JavaScript à l'intérieur d'une méthode et représente l'objet en lui-même lorsqu’il veut faire des choses à lui-même ou utiliser ses propriétés.

Monseigneur Node voudrait que vous créiez un objet "personnage" qui aura comme caractéristiques : son nom et prénom, sa tenue et une liste de ce que contient son sac ! En méthodes, il faut qu’il puisse se présenter, qu’il nous fasse un inventaire de ce qu’il y a dans son sac à dos. Bonne chance !

Monseigneur Node attend un résultat similaire au suivant :

Je suis Alice. Je porte une chemise hawaïenne, claquettes chaussettes et j'ai un sac à dos !

Dans mon sac à dos, il y a :

1 cahier

2 stylos  
1 gourde  
1 chargeur type C



Alice et son sac à dos

Quelque chose qui pourrait vous aider c’est "forEach()".

Monseigneur Node a une solution différente aux boucles "while" et "for" qu’il vous a montré précédemment. Cette solution est la méthode "forEach()". Celle-ci permet d'appliquer une fonction sur chaque élément du tableau.

const films = ["Le loup de Wall Street", "Vice-Versa", "Star Wars épisode III"];   
   
films.forEach(film => {   
 console.log(film);   
});

Monseigneur Node voudrait que vous ajoutiez une méthode dans votre objet qui servira à trier le sac à dos de votre *personnage* par ordre alphabétique. Il vous invite à chercher sur internet ce qu’est [la table ASCII](https://fr.wikipedia.org/wiki/American_Standard_Code_for_Information_Interchange).

La chose à faire sera une fonction qui vous permettra d’échanger les éléments à des positions bien définies. Elle prend en paramètre la liste d’éléments, la position du premier élément et la position du deuxième élément qu’on veut échanger.

mySwap = (liste, position1, position2) => { ... }

Cette fonction peut être mise directement dans votre objet afin d’être sûr qu’on ait toujours accès à cette fonction.

Dans un premier temps, il faudra regarder si le caractère actuel est un chiffre grâce [à la fonction "isNaN(...)"](https://www.w3schools.com/jsref/jsref_isnan.asp) (NaN = not a number, pas un nombre). Pensez à mettre un "!" devant pour mettre au négatif la condition.

!isNaN(123); // true

!isNaN(1B3); // false

Si le début de la string est un nombre ou un espace, il faut passer au caractère suivant jusqu’à ce qu’on arrive à un caractère alphabétique. En faisant attention de ne pas aller plus loin que le dernier caractère présent. Répétez la même démarche pour les autres strings une par une.

Il va falloir vérifier si le caractère à la position de la première string est supérieur au caractère à la position de la seconde string pour cela, on regarde leur valeur en ASCII. Si la condition est vérifiée, on échange les éléments. Dans le cas où les deux caractères auraient la même valeur ASCII, donc que ce serait le même caractère, on passe au caractère suivant de la string.

1. Rencontre avec les classes

Monseigneur Node va vous apprendre ce qu’est une classe ou “class” dans la religion du JavaScript. Le rôle est de faciliter la création d'objets ayant le même modèle et partager ainsi les données et leur comportement.

* 1. Comment crée-t-on une classe ?

Pour créer un objet issu d’une classe, il faut appeler son constructeur. Celui-ci, comme son nom l’indique, va permettre de construire un objet issu de la classe souhaitée. Souvent, on se sert du constructeur pour définir certaines variables à la construction plutôt que de s’amuser à les définir une par une.

Prenons l’exemple d’un personnage de jeu vidéo, vous pouvez définir la classe de la manière suivante :

class Personnage {   
 constructor(nom, sante, force, armure) {   
 this.nom = nom;   
 this.sante = sante;   
 this.force = force;   
 this.armure = armure;   
 this.xp = 0; // Toujours à 0 au début   
 this.niveau = 1; // Toujours à 1 au début   
 this.peutCombattre = true;   
 }   
 // Renvoie la description du personnage sous forme de string   
 description() {   
 return `${this.nom} a ${this.sante} points de vie, ${this.force} en force et ${this.armure} en points d'armure. ${this.nom} est niveau ${this.niveau}, a ${this.xp} points d'expérience et ${(this.peutCombattre ? "peut combattre" : "ne peut pas combattre")}.`;   
 }   
}

Vous pouvez voir que dans le constructeur sont initialisées les valeurs des variables du Personnage en fonction de ce qu’il recevra en paramètres. Et ici aussi on utilisera le mot-clé “this” quand on voudra accéder à un élément qui appartient à *cet* objet issu de *cette* classe.

Vous vous demandez peut-être par ailleurs, ce que cette ligne :

${(this.peutCombattre ? "peut combattre" : "ne peut pas combattre")}

Monseigneur Node vous explique que premièrement il y a *${}* pour ajouter une valeur. Ensuite il utilise ce qu’il appelle une **ternaire**. D’après la sainte documentation du JavaScript, ça fonctionne de la manière suivante : **condition ? résultatRetournéSiVrai : résultatRetournéSiFaux**. Dans cette exemple Monseigneur Node vous explique que si la variable peutCombattre de l’objet est à true la ternaire renverra *"peut combattre"* sinon elle renverra *"ne peut pas combattre"*.

* 1. Comment utilise-t-on une classe ?

Une fois la classe définie, vous pouvez l'utiliser pour créer des objets. L'opération de création d'un objet à partir d'une classe porte le nom d'instanciation.

Voici le reste de votre programme d'exemple :

let JeanKevin = new Personnage("Jean Kévin", 25, 5, 5);

console.log(JeanKevin.description());   
//Jean Kévin a 25 points de vie, 5 en force et 5 en points d'armure. Jean Kévin est niveau 1, a 0 points d'expérience et peut combattre.

Vous devez utiliser le mot-clé new. Le mot est assez explicite, car exprime que l’on instancie ici un nouveau Personnage.

* 1. Le modèle objet de JavaScript

Pour créer des relations entre objets, JavaScript utilise les prototypes. En plus de ses propriétés particulières, tout objet JavaScript possède une propriété interne appelée prototype. Il s'agit d'un lien qu’on appelle référence vers un autre objet. Lorsqu'on essaie d'accéder à une propriété qui n'existe pas dans un objet, JavaScript essaie de trouver cette propriété dans le prototype de cet objet.

const pomme = { a: 2 };

// Créer un objet poire avec l’objet pomme comme prototype   
const poire = Object.create(pomme);   
   
console.log(poire.a); // 2

Dans cet exemple, l'instruction JavaScript Object.create() est utilisée pour créer l'objet poire en lui donnant comme prototype l'objet pomme. Lors de l'appel à poire.a. C'est la propriété a de l’objet pomme qui est utilisée puisque la propriété a n'existe pas autrement dans l’objet poire.

Si le prototype d'un objet ne possède pas une propriété recherchée, alors c'est dans son propre prototype que la recherche continue, jusqu'à arriver à la fin de chaîne des prototypes. Si la propriété n'a été trouvée dans aucun objet, son accès renvoie la valeur undefined.

const pomme = { a: 2 };   
   
// Crée l’objet poire avec l’objet pomme comme prototype   
const poire = Object.create(pomme);   
   
console.log(poire.a); // 2   
   
// Crée l’objet banane avec l’objet poire comme prototype   
const banane = Object.create(poire);   
   
console.log(banane.a); // 2   
console.log(banane.b); // undefined

Ce mode de relation entre les objets JavaScript, Monseigneur Node appelle ça délégation : un objet délègue une partie de son fonctionnement à son prototype.

1. Conclusion

Voilà c’est la fin, Monseigneur Node est fier de votre travail et est heureux de voir 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.