Programmation Objet

I. Exemple introductive

Le JavaScript est un langage qui possède un fort potentiel pour la programmation orientée objet.

Un objet en JavaScript est un conteneur qui va pouvoir stocker plusieurs variables qu'on va appeler ici des propriétés. Lorsqu'une propriété contient une fonction en valeur, on appelle alors la propriété une méthode. Un objet est donc un conteneur qui va posséder un ensemble de propriétés et de méthodes qu'il est cohérent de regrouper.

On considère l'exemple :

```
<!DOCTYPE html>
<html>
   <head>
      <title>Cours JavaScript</title>
      <meta charset="utf-8">
      <meta name="viewport"
        content="width=device-width, initial-scale=1, user-scalable=no">
      <link rel="stylesheet" href="cours.css">
      <script src='cours.js' async></script>
   </head>

   <body>
      <h1>Titre principal</h1>
      <p>Un paragraphe</p>
      <p id='p1'></p>
   </body>
</html>
```

Et soit le fichier cours.js

```
/*Notre variable "utilisateur" est ici une variable-objet */

let utilisateur = {
   /*nom, age et mail sont des propriétés de l'objet utilisateur
    *La valeur de la propriété "nom" est un tableau*/
   nom : ['Pierre', 'Toto'],
   age : 29,
   mail : 'pierre.toto@gmail.com',

   //Bonjour est une méthode de l'objet utilisateur
```

```
  bonjour: function(){
     alert('Bonjour, je suis ' + this.nom[0] + ', j\'ai ' + this.age + ' ans');
  }
};


alert(typeof utilisateur);
```

Pour créer un objet, on commence par définir et initialiser une variable.

La variable **let utilisateur** stocke notre objet.
Cette variable stocke une valeur de type objet.

On remarque qu'on utilise dans le cas de la création d'un objet littéral une paire d'accolades qui indiquent au JavaScript qu'on souhaite créer un objet :

let utilisateur = { };

Ce qui nous intéresse particulièrement ici sont les membres de notre objet.

Un « membre » est un couple « nom : valeur », et peut être une propriété ou une méthode.

Notre objet est ici composé de différents membres : 3 propriétés et 1 méthode.

La première propriété nom de notre objet est particulière puisque sa valeur associée est un tableau.

Le membre nommé bonjour de notre objet est une méthode puisqu'une fonction anonyme lui est associée en valeur.

Vous pouvez également remarquer l'usage du mot clef **this** et de l'opérateur **.** dans notre méthode.

this.nom[0]

this.age

Chaque membre d'un objet est toujours composé d'un nom et d'une valeur qui sont séparées par **:**. Les différents membres d'un objet sont quant-à-eux séparés les uns des autres par des virgules.

```
nom : ['Pierre', 'Toto'],
age : 29,
mail : 'pierre.toto@gmail.com',
```

II. Création d'un objet littéral et sa manipulation

Nous pouvons créer des objets de 4 manières différentes en JavaScript. On va pouvoir :

- Créer un objet littéral ;
- Utiliser le constructeur Object() ;
- Utiliser une fonction constructeur personnalisée ;
- Utiliser la méthode create().

Ces différents moyens de procéder vont être utilisés dans des contextes différents, selon ce que l'on souhaite réaliser.

Création d'un objet littéral

Nous venons de créer un premier objet nommé **utilisateur**. Pour être tout à fait précis, nous avons créé un objet littéral :

```
/*"personne" est une variable qui contient un objet. Par abus de langage,
 *on dira que notre variable EST un objet*/
let personne = {
    //nom, age et mail sont des propriétés de l'objet "pierre"
    nom : ['Pierre', toto],
    age : 29,
    mail : 'pierre.toto@gmail.com',

    //Bonjour est une méthode de l'objet pierre
    bonjour: function(){
        alert('Bonjour, je suis ' + this.nom[0] + ', j\'ai ' + this.age + ' ans');
    }
};
```

On parle ici d'objet « littéral » car nous avons défini chacune de ses propriétés et de ses méthodes lors de la création, c'est-à-dire littéralement.

Pour créer un objet littéral, on utilise une syntaxe utilisant une paire d'accolades **{ … }** qui indique au JavaScript que nous créons un objet.

Nos objets vont généralement être stockés dans des variables. Par abus de langage, on confondra alors souvent la variable et l'objet et on parlera donc « d'objet » pour faire référence à notre variable stockant une valeur de type objet. Dans l'exemple ci-dessus, on dira donc qu'on a créé un objet nommé « **utilisateur** ».

Un objet est composé de différents couples de « nom : valeur » qu'on appelle membres. Chaque nom d'un membre doit être séparé de sa valeur par un caractère deux-points : et les différents membres d'un objet doivent être séparés les uns des autres par une virgule.

La partie « nom » de chaque membre suit les mêmes règles que le nommage d'une variable. La partie valeur d'un membre peut être n'importe quel type de valeur : une chaine de caractère, un nombre, une fonction, un tableau ou même un autre objet littéral.

Utiliser le point pour accéder aux membres d'un objet, les modifier ou en définir de nouveaux

Pour accéder aux propriétés et aux méthodes d'un objet, on utilise le caractère point . qu'on appelle également un accesseur. On va ici commencer par préciser le nom de l'objet puis l'accesseur puis enfin le membre auquel on souhaite accéder.

Cet accesseur va nous permettre non seulement d'accéder aux valeurs de nos différents membres mais également de modifier ces valeurs.

Exemple

Objet.html

```
<!DOCTYPE html>
<html>
   <head>
      <title>Cours JavaScript</title>
      <meta charset="utf-8">
      <meta name="viewport"
        content="width=device-width, initial-scale=1, user-scalable=no">
      <link rel="stylesheet" href="cours.css">
      <script src='cours.js' async></script>
   </head>

   <body>
      <h1>Titre principal</h1>
      <p>Un paragraphe</p>
      <p id='p1'></p>
      <p id='p2'></p>
      <p id='p3'></p>
   </body>
</html>
```

Cours.js

```
let personne = {
    nom : ['Pierre', 'toto'],
    age : 29,
    mail : 'pierre.toto@gmail.com',

    //Bonjour est une méthode de l'objet personne
    bonjour: function(){
        alert('Bonjour, je suis ' + this.nom[0] + ', j\'ai ' + this.age + ' ans');



/*On accède aux propriétés "nom" et "age" de "personne" et on affiche leur valeur
 *dans nos deux paragraphes p id='p1' et p id='p2'.
 *A noter : "document" est en fait aussi un objet, getElementById() une méthode
 *et innerHTML une propriété de l'API "DOM"!*/


document.getElementById('p1').innerHTML = 'Nom : ' + personne.nom;
document.getElementById('p2').innerHTML = 'Age : ' + personne.age;

//On modifie la valeur de la propriété "age" de "personne"
personne.age = 30;

document.getElementById('p3').innerHTML = 'Nouvel âge : ' + personne.age;

/*On accède à la méthode "bonjour" de l'objet "personne" qu'on exécute de la même
 *même façon qu'une fonction anonyme stockée dans une variable*/
```

Ici, on commence par accéder aux propriétés nom et age de notre objet personne en utilisant les notations personne.nom et personne.age.

Cela nous permet de récupérer les valeurs des propriétés.
En dessous, on utilise notre accesseur avec l'opérateur d'affectation = pour cette fois-ci modifier la valeur de la propriété age de notre objet pierre, et on affiche ensuite la nouvelle valeur pour bien montrer que la propriété a été modifiée.
Finalement, on utilise notre accesseur pour exécuter la méthode bonjour() de l'objet pierre. Pour faire cela, on procède de la même façon que pour exécuter une fonction anonyme placée dans une variable.

Enfin, on va encore pouvoir utiliser notre accesseur pour créer de nouveaux membres pour notre objet. Pour cela, il suffit de définir un nouveau nom de membre et de lui passer une valeur comme cela :

```
let personne = {

    //nom, age et mail sont des propriétés de l'objet "pesonne"
```

```
    nom : ['Pierre', 'toto'],
    age : 29,
    mail : 'pierre.toto@gmail.com',
    bonjour: function(){
        alert('Bonjour, je suis ' + this.nom[0] + ', j\'ai ' + this.age + ' ans');
    }
};
personne.taille = 170;
personne.prez = function(){
    alert('Bonjour, je suis ' + this.nom[0] +
        ', j\'ai ' + this.age + ' ans et je mesure ' + this.taille + 'cm');}
personne.prez();
```

Ici, on ajoute une propriété taille et une méthode prez() à notre objet personne. On invoque ensuite notre nouvelle méthode pour s'assurer qu'elle fonctionne bien.

Utiliser les crochets pour accéder aux propriétés d'un objet, les modifier ou en définir de nouvelles

On va également pouvoir utiliser des crochets plutôt que le point pour accéder aux propriétés des objets, mettre à jour leur valeur ou en définir de nouvelles. Cela ne va en revanche pas fonctionner pour les méthodes.

Les crochets vont être particulièrement utiles avec les valeurs de type tableau (qui sont des objets particuliers qu'on étudiera plus tard dans ce cours) puisqu'ils vont nous permettre d'accéder à une valeur en particulier dans notre tableau.

```
let personne = {
    //nom, age et mail sont des propriétés de l'objet "personne"
    nom : ['Pierre', 'toto'],
    age : 29,
    mail : 'pierre.toto@gmail.com',

    //Bonjour est une méthode de l'objet personne
    bonjour: function(){
        alert('Bonjour, je suis ' + this.nom[0] + ', j\'ai ' + this.age + ' ans');
    }
};
```

```
document.getElementById('p1').innerHTML = 'Nom complet : ' + personne['nom'];
document.getElementById('p2').innerHTML = 'Prénom : ' + personne['nom'][0];
personne['age'] = 30;
document.getElementById('p3').innerHTML = 'Age : ' + personne['age'];
```

L'utilisation du mot clef this

Le mot clef this est un mot clef qui apparait fréquemment dans les langages orientés objets. Dans le cas présent, il sert à faire référence à l'objet qui est couramment manipulé.

En l'occurrence, lorsqu'on écrit personne.bonjour(), le mot clef this va être remplacé par personne.

III. Définition et création d'un constructeur d'objets en JavaScript

Plutôt que de créer les objets un à un de manière littérale, il serait pratique de créer une sorte de plan ou de schéma à partir duquel on pourrait créer des objets similaires à la chaine.

Nous allons pouvoir faire cela en JavaScript en utilisant ce qu'on appelle un constructeur d'objets qui n'est autre qu'une fonction constructrice.

La fonction construction d'objets : définition et création d'un constructeur

Une fonction constructeur d'objets est une fonction qui va nous permettre de créer des objets semblables. En JavaScript, n'importe quelle fonction va pouvoir faire office de constructeur d'objets.

Pour construire des objets à partir d'une fonction constructeur, nous allons devoir suivre deux étapes :

- définir la fonction constructeur
- appeler ce constructeur avec le mot clefs **new**.

Dans une fonction constructrice, on va pouvoir définir un ensemble de propriétés et de méthodes.

Les fonctions sont un type particulier d'objets en JavaScript ! Comme tout autre objet, une fonction peut donc contenir des propriétés et des méthodes.

Exemple :

L'objectif ici va être de créer une fonction qui va nous permettre de créer des objets possédant les mêmes propriétés nom, age, mail et méthode bonjour() que notre objet littéral.
On va donc modifier notre script comme cela :

```
/Utilisateur() est une fonction constructeur
function Utilisateur(n, a, m){
   this.nom = n;
   this.age = a;
   this.mail = m;

   this.bonjour = function(){
      alert('Bonjour, je suis ' + this.nom[0] + ', j\'ai ' + this.age + ' ans');
   }
}
```

On définit ici une fonction Utilisateur() qu'on va utiliser comme constructeur d'objets.
Notez que lorsqu'on définit un constructeur, on utilise par convention une majuscule au début du nom de la fonction afin de bien discerner nos constructeurs des fonctions classiques dans un script.

Le code de la fonction de la fonction constructeur utilise le mot clef this qui va permettre de définir et d'initialiser les propriétés ainsi que les méthodes de chaque objet créé.

Le constructeur possède trois paramètres qu'on a ici nommé n, a et m qui vont nous permettre de transmettre les valeurs liées aux différentes propriétés pour chaque objet.

En effet, l'idée d'un constructeur en JavaScript est de définir un plan de création d'objets. Comme ce plan va potentiellement nous servir à créer de nombreux objets par la suite, on ne peut pas initialiser les différentes propriétés en leur donnant des valeurs effectives, puisque les valeurs de ces propriétés vont dépendre des différents objets créés.

Créer des objets à partir d'une fonction constructeur

Pour créer ensuite de manière effective des objets à partir de notre constructeur, nous allons simplement appeler le constructeur en utilisant le mot clef new. On dit également qu'on crée une nouvelle instance.

```
//Utilisateur() est une fonction constructeur
function Utilisateur(n, a, m){
    this.nom = n;
    this.age = a;
    this.mail = m;

    this.bonjour = function(){
        alert('Bonjour, je suis ' + this.nom[0] + ', j\'ai ' + this.age + ' ans');
    }
}
let perso = new Utilisateur(['Pierre', toto], 29, 'pierre.toto@gmail.com');
perso.bonjour();
/*On accède aux valeurs des propriétés de l'objet crée qu'on affiche dans
 *les paragraphes de notre fichier HTML*/
document.getElementById('p1').innerHTML = 'Nom complet : ' + perso['nom'];
document.getElementById('p2').innerHTML = 'Prénom : ' + perso['nom'][0];
document.getElementById('p3').innerHTML = 'Age : ' + perso['age'];
```

La page objet2.html

```
<!DOCTYPE html>

<html>

    <head>

        <title>Cours JavaScript</title>

        <meta charset="utf-8">

        <meta name="viewport"

          content="width=device-width, initial-scale=1, user-scalable=no">

        <link rel="stylesheet" href="cours.css">

        <script src='cours.js' async></script>

    </head>


    <body>

        <h1>Titre principal</h1>

        <p>Un paragraphe</p>

        <p id='p1'></p>

        <p id='p2'></p>

        <p id='p3'></p>
```

```
  </body>
</html>
```

<script src='cours.js' **async**> : async signifie que le script pourra être exécuté de façon **asynchrone**, dès qu'il sera disponible (téléchargé).

Comme le constructeur est une fonction, on va pouvoir l'appeler autant de fois qu'on le veut et donc créer autant d'objets que souhaité à partir de celui-ci et c'est d'ailleurs tout l'intérêt d'utiliser un constructeur. Chaque objet créé à partir de ce constructeur partagera les propriétés et méthodes de celui-ci.

(objet3.html, cours3.js)

```
//Utilisateur() est une fonction constructeur
function Utilisateur(n, a, m){
   this.nom = n;
   this.age = a;
   this.mail = m;
      this.bonjour = function(){
      alert('Bonjour, je suis ' + this.nom[0] + ', j\'ai ' + this.age + ' ans');
   }
}
var pierre = new Utilisateur(['Pierre', 'toto'], 29, 'pierre.toto@gmail.com');
var mathilde = new Utilisateur(['Math', 'Ml'], 27, 'math@edhec.com');
var florian = new Utilisateur(['Flo', 'Dchd'], 29, 'flo.dchd@gmail.com');
document.getElementById('p1').innerHTML = "Prénom de Pierre : " + pierre['nom'][0];
document.getElementById('p2').innerHTML = 'Age de Mathilde : ' + mathilde['age'];
document.getElementById('p3').innerHTML = 'Mail de Florian : ' + florian['mail'];
florian.bonjour();
```

Constructeur et différenciation des objets

En JavaScript, une fois un objet créé et à n'importe quel moment de sa vie, on peut modifier les valeurs de ses propriétés et ses méthodes ou lui en attribuer de nouvelles.

La fonction constructeur doit vraiment être vue en JavaScript comme un plan de base pour la création d'objets similaires et comme un moyen de gagner du temps et de la clarté dans son code. On ne va définir dans cette fonction que les caractéristiques communes de nos objets et on pourra ensuite rajouter à la main les propriétés particulières à un objet.

Exemple : rajouter une propriété taille à notre objet mathilde après sa création.(cours4.js)

```
function Utilisateur(n, a, m){
   this.nom = n;
   this.age = a;
   this.mail = m;

   this.bonjour = function(){
      alert('Bonjour, je suis ' + this.nom[0] + ', j\'ai ' + this.age + ' ans');
   }
}

let pierre = new Utilisateur(['Pierre', 'Giraud'], 29, 'pierre.giraud@edhec.com');
let mathilde = new Utilisateur(['Math', 'Ml'], 27, 'math@edhec.com');
let florian = new Utilisateur(['Flo', 'Dchd'], 29, 'flo.dchd@gmail.com');

mathilde.taille = 170;

document.getElementById('p1').innerHTML = 'Taille de Pierre : ' + pierre['taille'];
document.getElementById('p2').innerHTML = 'Taille de Math : ' + mathilde['taille'];
document.getElementById('p3').innerHTML = 'Mail de Florian : ' + florian['mail'];
```

IV. Méthode privée

Il suffit de déclarer la méthode privée en tant que fonction, *à l'intérieur* de la fonction de base. Par exemple :

```
function Personne (fiche)
  {
    var personnePrenom = typeof fiche.prenom !== 'undefined' ? fiche.prenom : "Nino" ;
    var personneGenre = typeof fiche.genre !== 'undefined' ? fiche.genre : "homme" ;
    var personneDateNaissance = typeof fiche.dateNaissance !== 'undefined' ?
fiche.dateNaissance : "1934" ;
    var personneCaractere = typeof fiche.caractere !== 'undefined' ? fiche.caractere :
"musicien" ;

    function vieillit()
      {
        personneDateNaissance-- ;
      }

    this.getPrenom = function()
      {
```

```
      return personnePrenom ;
    } ;

  this.getDateNaissance = function()
    {
      return personneDateNaissance ;
    } ;

  this.setPrenom = function(prenom)
    {
      personnePrenom=prenom ;
    } ;
}

var bernadette = new Personne ( {prenom:"Bernadette", genre:"femme", caractere:"très
chouette"}) ;
bernadette.vieillit() ;
alert(bernadette.getDateNaissance());//renvoie 1934
```

V. Constructeur Object, prototype et héritage en JavaScript

Notre code n'est pas optimal puisqu'en utilisant notre constructeur plusieurs fois on va copier à chaque fois la méthode bonjour() qui est identique pour chaque objet.

Ici, l'idéal serait de ne définir notre méthode qu'une seule fois et que chaque objet puisse l'utiliser lorsqu'il le souhaite. Pour cela, nous allons recourir à ce qu'on appelle des prototypes.

Les fonctions en JavaScript sont des objets. Lorsqu'on créé une fonction, le JavaScript va automatiquement lui ajouter une propriété prototype qui ne va être utile que lorsque la fonction est utilisée comme constructeur, c'est-à-dire lorsqu'on l'utilise avec la syntaxe new.

Cette propriété prototype possède une valeur qui est elle-même un objet. On parlera donc de « prototype objet » ou « d'objet prototype » pour parler de la propriété prototype.

Par défaut, la propriété prototype d'un constructeur ne contient que deux propriétés : une propriété constructor qui renvoie vers le constructeur contenant le prototype et une propriété __proto__ qui contient elle-même de nombreuses propriétés et méthodes.

A quoi servent la propriété prototype d'un constructeur et la propriété __proto__ dont disposent à la fois le constructeur mais également tous les objets créés à partir de celui-ci ?

En fait, le contenu de la propriété prototype d'un constructeur va être partagé par tous les objets créés à partir de ce constructeur. Comme cette propriété est un objet, on va pouvoir lui ajouter des propriétés et des méthodes que tous les objets créés à partir du constructeur vont partager.

Cela permet l'héritage en orienté objet JavaScript. On dit qu'un objet « hérite » des membres d'un autre objet lorsqu'il peut accéder à ces membres définis dans l'autre objet.

Exemple : (cours5.js)

```
function Utilisateur(n, a, m){
   this.nom = n;
   this.age = a;
   this.mail = m;
 }
   //on ajoute taille te bonjour à prototype
Utilisateur.prototype.taille = 170;
Utilisateur.prototype.bonjour = function(){
      alert('Bonjour, je suis ' + this.nom[0] + ', j\'ai ' + this.age + ' ans');
   }
let pierre = new Utilisateur(['Pierre', 'toto'], 29, 'pierre.toto@gmail.com');
let mathilde = new Utilisateur(['Math', 'Ml'], 27, 'math@edhec.com');
document.getElementById('p1').innerHTML = 'Taille de Pierre : ' + pierre['taille'];
document.getElementById('p2').innerHTML = 'Taille de Math : ' + mathilde['taille'];
document.getElementById('p3').innerHTML = 'Mail de Florian : ' + mathilde['mail'];
mathilde.bonjour();
```

Ici, on ajoute une propriété taille et une méthode bonjour() à la propriété prototype du constructeur Utilisateur(). Chaque objet créé à partir de ce constructeur va avoir accès à cette propriété et à cette méthode.

Définir des propriétés et des méthodes dans le prototype d'un constructeur nous permet ainsi de les rendre accessible à tous les objets créés à partir de ce constructeur sans que ces objets aient à les redéfinir.

Pour avoir le code le plus clair et le plus performant possible, nous définirons donc généralement les propriétés des objets (dont les valeurs doivent être spécifiques à l'objet) au sein du constructeur et les méthodes (que tous les objets vont pouvoir appeler de la même façon) dans le prototype du constructeur.