Partie 3 - Les objets : conception et utilisation

Les objets

Code : JavaScript

```
let myArray = ['Sébastien', 'Laurence', 'Ludovic', 'Pauline', 'Guillaume'];
```

Code : JavaScript

```
let myArray = [
	{
		nick: 'Sébastien',
		age: 23,
		sex: 'm',
		parent: 'aîné',
		work: 'Développeur'
	}, {
	},
	// et ainsi de suite...
];
```

Objet constructeur

Un objet représente quelque chose, une idée ou un concept. Ici, suite à l'exemple de la famille, nous allons créer un objet appelé Person qui contiendra des données, à savoir le prénom, l'âge, le sexe, le lien de parenté, le travail et la liste des amis (qui sera un tableau).

L'utilisation de tels objets se fait en deux temps :
1. On définit l'objet via un constructeur, cette étape permet de définir un objet qui pourra être réutilisé par la suite. Cet objet ne sera pas directement utilisé car nous en utiliserons une « copie » : on parle alors d'instance.
2. À chaque fois que l'on a besoin d'utiliser notre objet, on crée une instance de celui-ci, c'est-à-dire qu'on le « copie ».

Définition via un constructeur

Code : JavaScript

```
function Person() {
// Code du constructeur
}
```

Le code du constructeur va contenir une petite particularité : le mot-clé **this**. Ce mot-clé fait référence à l'objet dans lequel il est exécuté, c'est-à-dire ici le constructeur Person. Si on utilise **this** au sein du constructeur Person, **this** pointe vers Person. Grâce à **this**, nous allons pouvoir définir les propriétés de l'objet Person :

Code : JavaScript
```
function Person(nick, age, sex, parent, work, friends) {
	this.nick = nick;
	this.age = age;
	this.sex = sex;
	this.parent = parent;
	this.work = work;
	this.friends = friends;
}
```

Utilisation de l'objet

L'objet Person a été défini grâce au constructeur qu'il ne nous reste plus qu'à utiliser :

Code : JavaScript
```
// Définition de l'objet Person via un constructeur
function Person(nick, age, sex, parent, work, friends) {
	this.nick = nick;
	this.age = age;
	this.sex = sex;
	this.parent = parent;
	this.work = work;
	this.friends = friends;
}
// On crée des variables qui vont contenir une instance de l'objet Person :
let seb = new Person('Sébastien', 23, 'm', 'aîné', 'Développeur', []);
let lau = new Person('Laurence', 19, 'f', 'soeur', 'Sous-officier', []);

alert(seb.nick); // Affiche : « Sébastien »
alert(lau.nick); // Affiche : « Laurence »
```

Que s'est-il passé ici ? L'objet Person a été défini comme nous l'avons vu plus haut. Pour pouvoir utiliser cet objet, on définit une variable qui va contenir une instance de l'objet Person, c'est-à-dire une copie. Pour indiquer au Javascript qu'il faut utiliser une instance, on utilise le mot-clé **new**.

Modifier les données

Une fois la variable définie, on peut modifier les propriétés, exactement comme s'il s'agissait d'un simple objet littéral :

Code : JavaScript

```
let seb = new Person('Sébastien', 23, 'm', 'aîné', 'Développeur', []);
seb.nick = 'Bastien'; // On change le prénom
seb.age = 18; // On change l'âge
alert(seb.nick + ' a ' + seb.age + 'ans'); // Affiche : « Bastien a 18 ans »
```

Au final, on peut réécrire myArray comme contenant des éléments de type Person :

Code : JavaScript

```
let myArray = [
        new Person('Sébastien', 23, 'm', 'aîné', 'Développeur', []),
        new Person('Laurence', 19, 'f', 'soeur', 'Sous-officier', []),
        new Person('Ludovic', 9, 'm', 'frère', 'Etudiant', []),
        new Person('Pauline', 16, 'f', 'cousine', 'Etudiante', []),
        new Person('Guillaume', 16, 'm', 'cousin', 'Dessinateur', []),
];
```

Ajouter des méthodes

Si nous reprenons l'exemple précédent et que l'on souhaite ajouter un ami, il faut faire comme ceci :

Code : JavaScript

```
let seb = new Person('Sébastien', 23, 'm', 'aîné', 'Développeur', []);
// On ajoute un ami dans le tableau « friends »
seb.friends.push(new Person('Johann', 19, 'm', 'aîné', 'Développeur', []));

alert(seb.friends[0].nick); // Affiche : « Johann »
```

On voudrait ajouter une méthode addFriend() à l'objet Person de manière à pouvoir ajouter un ami comme ceci :

Code : JavaScript

```
seb.addFriend('Johann', 19, 'm', 'aîné', 'Développeur', []);
```

Il y a deux manières de définir une méthode pour un objet : dans le constructeur ou via prototype.

Définir une méthode dans le constructeur

Pour cela, rien de plus simple, on procède comme pour les propriétés, sauf qu'il s'agit d'une fonction :

Code : JavaScript

```
function Person(nick, age, sex, parent, work, friends) {
	this.nick = nick;
	this.age = age;
	this.sex = sex;
	this.parent = parent;
	this.work = work;
	this.friends = friends;

	this.addFriend = function(nick, age, sex, parent, work, friends) {
		this.friends.push(new Person(nick, age, sex, parent, work, friends));
	};
}
```

Le code de cette méthode est simple : il ajoute un objet Person dans le tableau « friends ».

Définir une méthode via prototype

Lorsque vous définissez un objet, il possède automatiquement un sous-objet appelé prototype.

Cet objet prototype va nous permettre d'ajouter des méthodes à un objet. Voici comment ajouter une méthode addFriend() à notre objet Person :

Code : JavaScript

```
Person.prototype.addFriend = function(nick, age, sex, parent, work, friends) {
	this.friends.push(new Person(nick, age, sex, parent, work, friends));
}
```

Le **this** fait ici aussi référence à l'objet dans lequel il s'exécute, c'est-à-dire l'objet Person.
L'ajout de méthodes par prototype a l'avantage d'être indépendant de l'objet.

En résumé

- Outre les objets natifs, le Javascript nous offre la possibilité de créer des objets personnalisés.
- Le constructeur contient la structure de base de l'objet. On définit un constructeur de la même manière qu'une fonction.
- Le mot-clé **this** fait référence à l'objet dans lequel il est utilisé, ce qui nous permet de créer les propriétés et les méthodes de l'objet.
- Une fois le constructeur défini, il est conseillé d'ajouter les méthodes via l'objet prototype.

Les chaînes de caractères
Les types primitifs

Pour créer une chaîne de caractères, on utilise généralement cette syntaxe :
Code : JavaScript
```
let myString = "Chaîne de caractères primitive";
```

Pour instancier un objet String, il faut faire comme ceci :
Code : JavaScript
```
let myRealString = new String("Chaîne");
```

Cela est valable pour les autres objets :
```
let myArray = []; // Tableau primitif
let myRealArray = new Array();

let myObject = {}; // Objet primitif
let myRealObject = new Object();

let myBoolean = true; // Booléen primitif
let myRealBoolean = new Boolean("true");

let myNumber = 42; // Nombre primitif
let myRealNumber = new Number("42");
```

L'objet String
L'objet String est l'objet qui gère les chaînes de caractères.

Propriétés
String ne possède qu'une seule propriété, length, qui retourne le nombre de caractères contenus dans une chaîne.
Code : JavaScript
```
alert('Ceci est une chaîne !'.length);
```

Méthodes

La casse et les caractères : toLowerCase() et toUpperCase()

toLowerCase() et toUpperCase() permettent respectivement de convertir une chaîne en minuscules et en majuscules.

Code : JavaScript

```
let myString = 'le javascript';
myString = myString.toUpperCase(); // Retourne : « LE JAVASCRIPT »
```

Accéder aux caractères avec charAt()

La méthode charAt() permet de récupérer un caractère en fonction de sa position dans la chaîne de caractères.

Code : JavaScript

```
let myString = 'Pauline';
let first = myString.charAt(0); // P
let last = myString.charAt(myString.length - 1); // e
```

Supprimer les espaces avec trim()

La méthode trim() sert à supprimer les espaces avant et après une chaîne de caractères.

Code : JavaScript

```
let myString = "       Hello World!        ";
alert(myString.trim()); // affiche « Hello World! »
```

Rechercher, couper et extraire

Connaître la position avec indexOf() et lastIndexOf()

La méthode indexOf() est utile dans deux cas de figure :

- Savoir si une chaîne de caractères contient un caractère ou un morceau de chaîne ;
- Savoir à quelle position se trouve le premier caractère de la chaîne recherchée.

indexOf() retourne la position du premier caractère trouvé, et s'il n'y en a pas la valeur -1 est retournée.

Code : JavaScript

```
let myString = 'Le JavaScript est plutôt cool';
let result = myString.indexOf('JavaScript');
if (result > -1) {
        alert('La chaîne contient le mot "Javascript" qui débute à la position ' + result);
}
```

Extraire une chaîne avec substring(), substr() et slice()

La méthode substring() retourne un sous-chaîne de la chaîne courante, entre un indice de début (inclus) et un indice de fin (exclus).
Code : JavaScript
```
let date = "23/12/2016";
let jour = date.substring(0,2);
let mois = date.substring(3,5);
let annee = date.substring(6,10);
document.write(jour+"<br />");
document.write(mois+"<br />");
document.write(annee+"<br />");
```

La méthode slice() renvoie un objet tableau, contenant une copie d'une portion du tableau d'origine, la portion est définie par un indice de début (inclus) et un indice de fin (exclus).
Code : JavaScript
```
let fruits = ["Banane", "Orange", "Citron", "Pomme", "Mangue"];
let agrumes = fruits.slice(1, 3);
// fruits vaut --> ["Banane", "Orange", "Citron", "Pomme", "Mangue"]
// agrumes vaut --> ["Orange", "Citron"]
```

Couper une chaîne en un tableau avec split()
La méthode split() permet de couper une chaîne de caractères à chaque fois qu'une sous-chaîne est rencontrée.
Code : JavaScript
```
let myCSV = 'Pauline,Guillaume,Clarisse'; // CSV = Comma Separated Values
let splitted = myCSV.split(','); // On coupe à chaque fois qu'une virgule est rencontrée
alert(splitted.length); // 3
```

En résumé
- Il existe des objets et des types primitifs. Si leur utilisation semble identique, il faut faire attention lors des tests avec les opérateurs instanceof et typeof. Mais heureusement valueOf() sera d'une aide précieuse.
- Il est préférable d'utiliser les types primitifs, comme nous le faisons depuis le début de ce cours.
- String fournit des méthodes pour manipuler les caractères : mettre en majuscule ou en minuscule, récupérer un caractère grâce à sa position et supprimer les espaces de part et d'autre de la chaîne.
- String fournit aussi des méthodes pour rechercher, couper et extraire.
- L'utilisation du caractère tilde est méconnue, mais peut se révéler très utile en couple avec indexOf() ou lastIndexOf().

Les expressions régulières (1/2)

En résumé

- Les regex constituent une technologie à part, utilisée au sein du Javascript et qui permet de manipuler les chaînes de caractères. La syntaxe de ces regex se base sur celle du langage Perl.
- Plusieurs méthodes de l'objet String peuvent être utilisées avec des regex, à savoir match(), search(), split() et replace().
- L'option i indique à la regex que la casse doit être ignorée.
- Les caractères ^ et $ indiquent respectivement le début et la fin de la chaîne de caractères.
- Les classes et les intervalles de caractères, ainsi que les types génériques, servent à rechercher un caractère possible parmi plusieurs.
- Les différents métacaractères doivent absolument être échappés.
- Les quantificateurs servent à indiquer le nombre de fois qu'un caractère peut être répété.

Les expressions régulières (partie 2/2)

Construire une regex

Code : JavaScript

```
let email = prompt("Entrez votre adresse e-mail :", "javascript@viaformation.fr");
if (/^[a-z0-9._-]+@[a-z0-9._-]+\.[a-z]{2,6}$/.test(email)) {
	alert("Adresse e-mail valide !");
} else {
	alert("Adresse e-mail invalide !");
}
```

Rechercher et remplacer

Un rechercher-remplacer se fait au moyen de la méthode replace() de l'objet String.

Code : JavaScript

```
let sentence = 'Je m\'appelle Sébastien';
let result = sentence.replace(/Sébastien/, 'Johann');
alert(result); // Affiche : « Je m'appelle Johann »
```

Rechercher et capturer

Code : JavaScript

```
let date = '12/23/2016';
let = date.replace(/^(\d{2})\/(\d{2})\/(\d{4})$/, 'Le $2/$1/$3');
alert(date); // Le 23/12/2016
```

Les données numériques

L'objet Number

Code : JavaScript
```
var myNumber = new Number('3'); // La chaîne de caractères « 3 » est convertie en un nombre de valeur 3
```

Cet objet possède des propriétés accessibles directement sans aucune instanciation, la plus utilisée est :

- NaN : vous connaissez déjà cette propriété qui signifie Not A Number et qui permet, généralement, d'identifier l'échec d'une conversion de chaîne de caractères en un nombre.

L'objet Math

Les propriétés

Les propriétés de l'objet Math sont des constantes qui définissent certaines valeurs mathématiques comme le nombre pi (π) ou le nombre d'Euler (e).

Code : JavaScript
```
alert(Math.PI); // Affiche la valeur du nombre pi
alert(Math.E); // Affiche la valeur du nombre d'Euler
```

Les méthodes

L'objet Math comporte de nombreuses méthodes permettant de faire divers calculs un peu plus évolués qu'une simple division.

Arrondir et tronquer

La méthode floor() retourne le plus grand entier inférieur ou égal à la valeur que vous avez passée en paramètre :

Code : JavaScript
```
Math.floor(33.15); // Retourne : 33
Math.floor(33.95); // Retourne : 33
Math.floor(34); // Retourne : 34
```

Calculs de puissance et de racine carrée
Code : JavaScript
```
Math.pow(3, 2); // Le premier paramètre est la base, le deuxième est l'exposant
                // Ce calcul donne donc : 3 * 3 = 9
```

Code : JavaScript
```
Math.sqrt(9); // Retourne : 3
```

Les cosinus et sinus
Code : JavaScript
```
Math.cos(Math.PI); // Retourne : -1
Math.sin(Math.PI); // Retourne : environ 0
```

Choisir un nombre aléatoire
Code : JavaScript
```
alert(Math.random()); // Retourne un nombre compris entre 0 et 1
```

Les fonctions de conversion
parseInt() et parseFloat(), elles permettent de convertir une chaîne de caractères en un nombre.

Les fonctions de contrôle
isNaN() renvoie true si la variable ne contient pas de nombre, elle s'utilise de cette manière :
Code : JavaScript
```
let myNumber = parseInt("test"); // Notre conversion sera un échec et renverra « NaN »
alert(isNaN(myNumber)); // Affiche « true », cette variable ne contient pas de nombre
```

isFinite(), cette fonction renvoie true si le nombre ne tend pas vers l'infini :
Code : JavaScript
```
let myNumber = 1/0; // 1 divisé par 0 tend vers l'infini
alert(isFinite(myNumber)); // Affiche « false », ce nombre tend vers l'infini
```

En résumé

- Un objet possède parfois des propriétés issues du constructeur. C'est le cas de l'objet Number, avec des propriétés comme Number. MAX_VALUE ou Number. **NaN**.
- De même que Number, l'objet Math possède ce genre de propriétés. Utiliser π est alors simple : Math.PI.
- Il faut privilégier parseInt() et parseFloat() pour convertir une chaîne de caractères en un nombre.
- L'usage des propriétés de l'objet Number lors de tests est déconseillé : il vaut mieux utiliser les fonctions globales prévues à cet effet, comme par exemple isNaN() au lieu de **NaN**.

La gestion du temps

En résumé

- Le Javascript se base sur un timestamp exprimé en millisecondes pour calculer les dates. Il faut faire attention, car un timestamp est parfois exprimé en secondes, comme au sein du système Unix ou dans des langages comme le PHP.
- En instanciant, sans paramètres, un objet Date, ce dernier contient la date de son instanciation. Il est ensuite possible de définir une autre date par le biais de méthodes ad hoc.
- Date est couramment utilisé pour déterminer le temps d'exécution d'un script. Il suffit de soustraire le timestamp du début au timestamp de fin.
- Plusieurs fonctions existent pour créer des délais d'exécution et de répétition, ce qui peut être utilisé pour réaliser des animations.

Les tableaux

Parcourir un tableau

Code : JavaScript

```
let myArray = ["C'est", "un", "test"],
    length = myArray.length;
for(var i=0;i<length; i++){
        alert(
        'Index : ' + i
        + '\n' +
        'Valeur : ' + myArray[i]
        );
}
```

Utilisons plutôt une nouvelle méthode nommée **forEach()**, elle reçoit en paramètres trois arguments :

- Le premier contient la valeur contenue à l'index actuel ;
- Le deuxième contient l'index actuel ;
- Le troisième est une référence au tableau actuellement parcouru.

Code : JavaScript

```
let myArray = ["C'est", "un", "test"];
myArray.forEach(function(value, index, array) {
        alert(
        'Index : ' + index + '\n' +
        'Valeur : ' + value
        );
 });
```

Trier un tableau

Deux méthodes peuvent vous servir à trier un tableau.

La méthode reverse()

Code : JavaScript

```
let myArray = [1, 2, 3, 4, 5];
myArray.reverse();
alert(myArray); // Affiche : 5,4,3,2,1
```

La méthode sort()
Code : JavaScript

```
let myArray = [3, 1, 5, 10, 4, 2];
myArray.sort();
alert(myArray); // Affiche : 1,10,2,3,4,5
```

En résumé

- Pour concaténer deux tableaux, il faut utiliser la méthode **concat()**, car l'opérateur + ne fonctionne pas selon le comportement voulu.
- La méthode **forEach()** permet de parcourir un tableau en s'affranchissant d'une boucle for. Mais cette méthode n'est pas supportée par les versions d'Internet Explorer antérieures à la version 9.
- **indexOf()** et **lastIndexOf()** permettent de rechercher un élément qui peut être une chaîne de caractères, un nombre, ou même un tableau. Il faudra toutefois faire attention lors de la comparaison de deuxtableaux.
- L'utilisation d'une fonction pour trier un tableau est possible et se révèle particulièrement utile pour effectuer un tri personnalisé.
- Les piles et les files sont un moyen efficace pour stocker et accéder à de grandes quantités de données.

Les images

En résumé

- L'objet Image est généralement utilisé pour s'assurer qu'une image a été chargée, en utilisant l'événement load().

Les polyfills et les wrappers

En résumé

- Les polyfills sont un moyen de s'assurer de la prise en charge d'une méthode si celle-ci n'est pas supportée par le navigateur, et ce sans intervenir dans le code principal. C'est donc totalement transparent.
- Les wrappers permettent d'ajouter des propriétés ou des méthodes aux objets, en particulier les objets natifs, en créant un objet dérivé de l'objet en question.