Cours de Javascript V

Partie V

1. Comprendre et utiliser des fonctions, tableaux et objets
    - Introduction
    - Les fonctions
    - Les tableaux
    - Les objets

1.1 Introduction

Les fonctions font partie des briques fondamentales de JavaScript.

Une fonction est une procédure JavaScript, un ensemble d'instructions effectuant une tâche ou calculant une valeur. Afin d'utiliser une fonction, il est nécessaire de l'avoir auparavant définie au sein de la portée dans laquelle on souhaite l'appeler.

1.2 Les fonctions

Il y a les fonctions ou variables natives (déjà existantes), mais on peut aussi en créer de nouvelles, selon la structure suivante :

```
// Le terme "function" est obligatoire pour déclarer une fonction
function myFunction(arguments) {
    // Le code que la fonction va devoir exécuter
};
```

```
function multiplierParDeux() {
    var result = parseInt(prompt('Entrez le nombre à multiplier par 2 :'));
    alert(result * 2);
};
// On appelle la fonction créée
multiplierParDeux();

// Puis un message intermédiaire
alert('Entrez un autre nombre !');

// Et on appelle de nouveau la fonction
multiplierParDeux();
```

1.2 Les fonctions

- La portée des variables: variables locales et globales

Attention: toute variable déclarée dans une fonction n'est utilisable que dans cette même fonction. Ces variables spécifiques à une seule fonction ont un nom: les variables locales. Déclarées en dehors des fonction, on parle de variables globales.

```
24  // Déclaration deux variables globales
25  let x = 5;
26  var y = 10;
27
28  // Fonction qui utilise les variables globales
29  function showValeur1() {
30      document.write('<p>Variable x = ' + x +
31                     '<br>Variable y = ' + y + '</p>');
32  };
33
34  // Fonction qui utilise des variables locales
35  function showValeur2() {
36      let a = 1;
37      var b = 2;
38      document.write('<p>Variable a = ' + a +
39          '<br>Variable b = ' + b + '</p>');
40  };
41
42  // Fonction qui utilise aussi des variables locales
43  function showValeur3() {
44      let x = 20;
45      var y = 40;
46      document.write('<p>Variable x = ' + x +
47          '<br>Variable y = ' + y + '</p>');
48  };
```

```
49
50  // On oublie pas l'appel des fonctions !
51  showValeur1();
52  showValeur2();
53  showValeur3();
54
55  // Affichage des variables globales
56  document.write('<p>Voyons voir la variable globale x = ' + x +
57                 '<br> et y = ' + y + '</p>');
58
59  // Affichage des variables locales
60  document.write('<p>Voyons voir la variable locale a = ' + a +
61                 '<br> et b = ' + b + '</p>');
62                 console.log(a);
63                 console.log(b);
```

1.2 Les fonctions

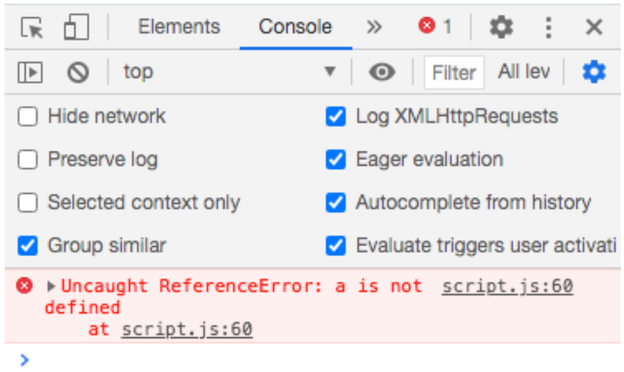
Les fonctions

Variable x = 5
Variable y = 10

Variable a = 1
Variable b = 2

Variable x = 20
Variable y = 40

Voyons voir la variable globale x = 5
et y = 10



Mais on évite de créer des variables locales et globales qui portent le même nom.

En règle générale, on préfère utiliser des variables locales (pour éviter les confusions).

1.2 Les fonctions

- Les arguments

Les arguments d'une fonction sont des valeurs qu'on va passer à notre fonction afin qu'elle fonctionne normalement ou pour préciser le comportement qu'elle doit adopter. Certaines fonctions ne vont pas nécessiter d'arguments, d'autres vont avoir besoin d'un argument, d'autres de deux, etc. De plus, certains arguments vont être obligatoires tandis que d'autres vont être facultatifs.

```
// Notre argument est la variable « arg »
function myFunction(arg) {
    alert('Votre argument : ' + arg);
};
myFunction('En voilà un beau test !');
```

```
function myFunction(arg) {
    alert('Votre argument : ' + arg);
};
myFunction(prompt('Que souhaitez-vous passer en argument à la fonction ?'));
```

1.2 Les fonctions

```
function exemple(first, second) {
    // On peut maintenant utiliser les variables « first » et « second » comme on
    // le souhaite:
        alert('Votre premier argument : ' + first);
    alert('Votre deuxième argument : ' + second);
};
exemple(
    prompt('Entrez votre premier argument :'),
    prompt('Entrez votre deuxième argument :')
);
```

- Les valeurs de retour

Une valeur de retour est une valeur unique qui va être renvoyée par la fonction après son exécution et qu'on va pouvoir récupérer pour la manipuler dans notre script.
Certaines fonctions prédéfinies vont renvoyer une valeur de retour tandis que d'autres ne vont pas en renvoyer.

1.2 Les fonctions

Attention: l'instruction return met fin à l'exécution d'une fonction, ce qui signifie que toutes les autres opérations qui suivent une instruction return dans une fonction seront ignorées.
Pour cette raison, on fera toujours bien attention à placer l'instruction return en fin de fonction, après que toutes les opérations aient été réalisées.

```
function sayHello() {
    // L'instruction « return » suivie d'une valeur.
    // Cette dernière est donc renvoyée par la fonction (il ne peut pas y en avoir d'autres)
    return 'Bonjour !';
};
 alert(sayHello());
```

- Les fonctions anonymes

Les fonctions anonymes sont, comme leur nom l'indique, des fonctions qui ne vont pas posséder de nom. Lorsqu'on crée une fonction, nous ne sommes pas obligés de lui donner un nom.

1.2 Les fonctions

Généralement, on utilise les fonctions anonymes lorsque le code de notre fonction n'est appelé qu'à un endroit dans notre script et n'est pas réutilisé.

```
// On peut enfermer la fonction dans une variable pour s'en servir
let message = function() {
    // Le code de votre fonction anonyme
    alert('Voici la fonction anonyme');
};
```

- Les fonctions prédéfinies

Le langage JavaScript dispose de nombreuses fonctions que nous pouvons utiliser pour effectuer différentes tâches. Les fonctions définies dans le langage sont appelées fonctions prédéfinies ou fonctions prêtes à l'emploi car il nous suffit de les appeler pour nous en servir.

1.2 Les fonctions

Par exemple, le JavaScript dispose d'une fonction nommée random() (qui appartient à l'objet Math) et qui va générer un nombre décimal aléatoire entre 0 et 1 ou encore d'une fonction replace() (qui appartient cette fois-ci à l'objet String) qui va nous permettre de chercher et de remplacer une expression par une autre dans une chaine de caractères.

```html
<!DOCTYPE html>
<html lang="fr">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Les fonctions prédéfinies</title>
</head>

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

    <script type="text/javascript" src='cours.js'></script>
</body>

</html>
```

```javascript
let phrase = 'Bonjour, je suis Alexandre';

/*Math.random() génère un nombre décimal aléatoire entre 0 et 1 qu'on
  place ici au sein de notre paragraphe  p id='p1'*/
document.getElementById('p1').innerHTML = Math.random();

/*String.replace() chercher une expression dans une chaine de caractères
  et la remplace par une autre. Ici, on cherche "Pierre" dans let prez et
  on remplace par "Mathilde" avant d'afficher le résultat dans p id='p2'*/
let phrase2 = phrase.replace('Alexandre', 'Natacha');
document.getElementById('p2').innerHTML = phrase2;
```

1.2 Les fonctions

L'intérêt principal des fonction prédéfinies est de nous permettre de réaliser des opérations complexes de manière très simple : en les appelant, tout simplement.

En effet, vous devez bien comprendre que derrière ces noms de fonctions se cachent des codes parfois longs et complexes qui vont être exécutés lorsqu'on appelle la fonction et qui vont permettre de réaliser une opération précise (générer un nombre aléatoire, etc.).

En plus de cela, le code d'une fonction est réutilisable : cela veut dire qu'on va pouvoir appeler une même fonction autant de fois qu'on le souhaite afin qu'elle accomplisse plusieurs fois la même opération.

Pour exécuter le code caché derrière la fonction, il suffit de l'appeler ou de « l'invoquer ». Pour faire cela, on n'a qu'à écrire le nom de la fonction suivi d'un couple de parenthèses et éventuellement préciser des arguments entre les parenthèses.

1.2 Les fonctions

- Quelques exemples de fonctions prédéfinies dites aussi natives:

  - isNaN() : La fonction isNaN() détermine si une valeur est NaN ou non. (Not-A-Number)
  - parseFloat() : La fonction parseFloat() convertit une chaîne de caractères en un nombre décimal.
  - parseInt() : La fonction parseInt() convertit une chaîne de caractères et renvoie un entier dans la base donnée.
  - sort() : Trie les éléments par ordre alphabétique.
  - replace() : La fonction replace () qui remplace un bout de chaîne par un autre.

1.3 Les tableaux

Un tableau ou plutôt un array en anglais, est une variable qui contient plusieurs valeurs, appelées items. Chaque item est accessible au moyen d'un indice (index en anglais) et dont la numérotation commence à partir de 0.

```
var myArray = ['Rafael', 'Mathilde', 'Ahmed', 'Jérôme', 'Guillaume'];
// Le contenu se définit entre crochets, avec une virgule entre chaque valeur.
// La chaîne 'Rafael' correspond à l'indice 0, 'Mathilde' à l'indice 1...
console.log(myArray[2]); // Affiche : « Ahmed »
```

```
var myArray = ['Rafael', 'Mathilde', 'Ahmed', 'Jérôme', 'Guillaume'];
myArray[1] = 'Paul';
console.log(myArray[1]); // Affiche : « Paul »
console.log(myArray); // Affiche : ["Rafael", "Paul", "Ahmed", "Jérôme", "Guillaume"]
```

- Opérations sur les tableaux

On peut ajouter des items avec la méthode push() :

1.3 Les tableaux

```
var myArray = ['Rafael', 'Mathilde'];
// Ajoute « Ahmed » à la fin du tableau
myArray.push('Ahmed');
console.log(myArray); // Affiche : ["Rafael", "Mathilde", "Ahmed"]
// Ajoute « Jérôme » et « Guillaume » à la fin du tableau
myArray.push('Jérôme', 'Guillaume');
console.log(myArray); // Affiche : ["Rafael", "Mathilde", "Ahmed", "Jérôme", "Guillaume"]
```

La méthode unshift() fonctionne comme push(), excepté que les items sont ajoutés au début du tableau. Les méthodes shift() et pop() retirent respectivement le premier et le dernier élément du tableau.

```
var myArray = ['Rafael', 'Mathilde', 'Ahmed', 'Jérôme', 'Guillaume'];
myArray.shift(); // Retire « Rafael »
console.log(myArray); //Affiche : ["Mathilde", "Ahmed", "Jérôme", "Guillaume"]
myArray.pop(); // Retire « Guillaume »
console.log(myArray); // Affiche : ["Mathilde", "Ahmed", "Jérôme"]
```

On peut découper une chaîne de caractères en tableau avec split() :

1.3 Les tableaux

```
var family = 'Jérôme Guillaume Paul',
    familyArray = family.split(' '); // Avec les espaces, on a trois items
console.log(family);
console.log(familyArray);
```

On fait l'inverse avec join() :

```
var family2 = familyArray.join('-');
console.log(family2); // Affiche : Jérôme-Guillaume-Paul
```

- Parcourir un tableau

On peut parcourir un tableau avec for :

```
var myArray = ['Rafael', 'Mathilde', 'Ahmed', 'Jérôme', 'Guillaume'];
console.log(myArray.length); // La length est de 5
// On crée la variable c pour que la condition ne soit pas trop lourde en caractères
for (var i = 0, c = myArray.length; i < c; i++) {
    // On affiche chaque item, l'un après l'autre, jusqu'à la longueur totale du tableau
    console.log(myArray[i]);
};
```

1.4 Les objets

- Les objets littéraux

On peut remplacer l'indice par un identifiant. Dans ce cas on crée un objet (dans l'exemple familyMember). Les identifiants créés (self, sister...) sont des propriétés, avec plusieurs possibilités d'affichage. On peut ajouter des données (avec une méthode différente que pour un tableau).

```
let familyMember = {
    self: 'Rafael',
    sister: 'Mathilde',
    brother: 'Ahmed',
    cousin_1: 'Jérôme',
    cousin_2: 'Guillaume'
};
let id = 'sister';
console.log(familyMember[id]); // Affiche : « Mathilde »
console.log(familyMember.brother); // Affiche : « Ahmed »
console.log(familyMember['self']); // Affiche : « Rafael »
familyMember['uncle'] = 'Karim'; // On ajoute une donnée, avec un identifiant.
familyMember.aunt = 'Pauline'; // On peut ajouter aussi de cette manière.
console.log(familyMember.uncle); // Affiche : « Ahmed »
```

1.4 Les objets

- Parcourir un objet avec for in

On ne peut pas parcourir l'objet avec for, parce que for s'occupe d'incrémenter des variables numériques. Là on fournit une variable-clé pour le parcours

```
let familyMember2 = {
    self: 'Rafael',
    sister: 'Mathilde',
    brother: 'Ahmed',
    cousin_1: 'Jérôme',
    cousin_2: 'Guillaume'
};
// On stocke l'identifiant dans « id » pour parcourir l 'objet « familyMember2 »
for (let index in familyMember2) {
    console.log(familyMember2[index]);
};
```

Liens externes pour aller plus loin

- https://www.pierre-giraud.com/javascript-apprendre-coder-cours/presentation-fonction/
- https://www.devenir-webmaster.com/V2/TUTO/CHAPITRE/JAVASCRIPT/17-les-fonctions/
- https://developer.mozilla.org/fr/docs/Learn/JavaScript/First_steps/tableaux
- https://www.pierre-giraud.com/javascript-apprendre-coder-cours/creation-objet-litteral/