Partie 2 - Les bases du langage

Table des matières

1. LES FONCTIONS

Dans un programme, le besoin de répéter une même action à différents endroits du code arrive rapidement …

Pour cela, on peut évidement copier/coller le code à répéter. La maintenance deviendra rapidement complexe si on doit modifier ce code répété !

Heureusement, nous avons les fonctions 😊

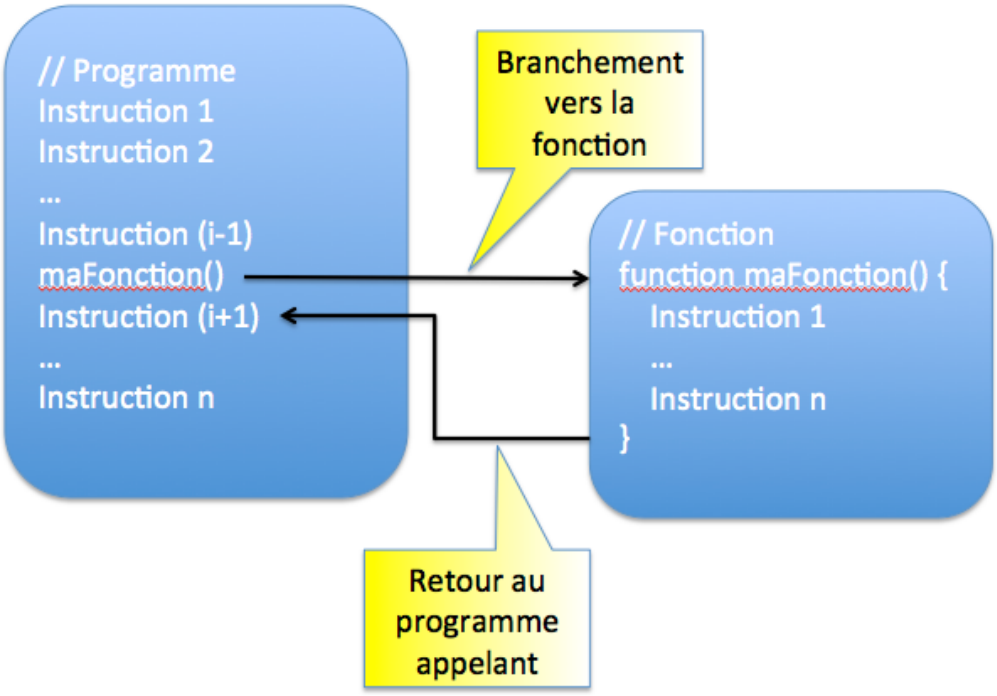
Une fonction est un regroupement d'instructions qui réalise une tâche donnée.

```
//Déclaration de la fonction nomée direBonjour
function direBonjour() {
  console.log("Bonjour !");
}
console.log("Début du programme");
direBonjour(); //appel de la fonction direBonjour = exécution
console.log("Fin du programme");
```

L'opération de création d'une fonction s'appelle la **déclaration.**

L'appel d'une fonction s'effectue en écrivant le nom de la fonction suivi de parenthèses :

L'appel d'une fonction déclenche l'exécution des instructions qui la constituent, puis l'exécution reprend à l'endroit où la fonction a été appelée.



Mécanisme d'appel d'une fonction

Valeur de retour

L'utilisation du mot-clé « ***return*** » dans une fonction permet de lui donner une **valeur de retour**.

Son appel produit un résultat qui correspond à la valeur placée juste après le return :

```
//avec return
function direBonjour() {
  return "Bonjour !";
}
console.log("Début du programme");
const resultat = direBonjour();
console.log(resultat); // "Bonjour !"
console.log("Fin du programme");
```

Cette valeur de retour peut être de n'importe quel type (nombre, chaîne, etc.).
En revanche, une fonction ne peut renvoyer qu'une seule valeur.

Rien n'oblige à récupérer la valeur de retour d'une fonction, mais dans ce cas, cette valeur est "oubliée" par le programme qui appelle la fonction !

Si on essaie de récupérer la valeur de retour d'une fonction qui n'inclut pas d'instruction `return`, on obtient la valeur JavaScript `undefined`.

javascript

```
function maFonction() {
  // Pas d'instruction return
}

let resultat = maFonction();
console.log(resultat); // undefined
```

Une fonction qui ne renvoie pas de valeur est parfois appelée une **procédure**.

L'exécution de l'instruction `return` renvoie immédiatement vers le programme appelant. Il ne faut jamais ajouter d'instructions après un `return` dans une fonction : elles ne seraient jamais exécutées.

Variables locales

Il est possible de déclarer des variables à l'intérieur d'une fonction, elles sont appelées des **variables locales.**

Attention à la **portée des variables** : elles ne sont utilisables qu'à l'intérieur de la fonction !

```
function direBonjour() {
  const message = "Bonjour !"; //variable locale message
  return message;
}
console.log(direBonjour()); // "Bonjour !"
console.log(message); // Erreur : message n'existe pas ici
```

A chaque appel d'une fonction qui déclare des variables locales, ces variables sont recréées.

On peut donc appeler plusieurs fois la même fonction, et chaque appel sera parfaitement indépendant des autres.

Ne pas pouvoir utiliser de variables locales en dehors des fonctions où elles sont déclarées peut sembler une limitation. C'est au contraire un double avantage :

- Une fonction peut être conçue comme une entité autonome et réutilisable.
- Un programme peut déclarer ses propres variables et utiliser autant de fonctions que nécessaire, sans se préoccuper des variables locales qui y sont déclarées.

Passage de paramètres

Les paramètres d'une fonction sont définis entre parenthèses juste après le nom de la fonction. On peut ensuite utiliser leur valeur dans le corps de la fonction.

```
function direBonjour(prenom) {
  const message = "Bonjour, " + prenom + " !";
  return message;
}
console.log(direBonjour("Nadine")); // "Bonjour, Nadine !"
console.log(direBonjour("Pierre")); // "Bonjour, Pierre !"
```

Voici la syntaxe générale de la déclaration d'une fonction acceptant des paramètres.

Leur nombre n'est pas limité, mais il est rarement nécessaire de dépasser 3 ou 4 paramètres.

```
1 // Déclaration de la fonction maFonction
2 function maFonction(param1, param2, ...) {
3   // Instructions pouvant utiliser param1, param2, ...
4 }
5
6 // Appel de la fonction maFonction
7 // param1 reçoit la valeur de arg1, param2 la valeur de arg2, ...
8 maFonction(arg1, arg2, ...);
```

Créer des fonctions à bon escient

Une fonction peut utiliser les mêmes éléments qu'un programme classique : variables, conditions, boucles, etc. Une fonction peut même faire appel à une autre fonction, ce qui ouvre des possibilités infinies pour construire nos programmes.

Il convient toutefois de rester raisonnable et de ne pas multiplier artificiellement le nombre de fonctions d'un programme, sous peine de compliquer sérieusement sa compréhension (c'est le syndrome du code spaghetti). Il vaut mieux essayer de créer des fonctions ayant chacune un rôle bien défini et minimiser les interdépendances entre fonctions.

Utiliser les fonctions prédéfinies de JavaScript

Nous avons déjà utilisé des fonctions prédéfinies de JavaScript comme ***prompt***() ou ***alert***().

Le langage vous propose un nombre important de fonctions qui répondent à des besoins variés.

```
// Exemples utilisant les fonctions mathématiques de javascript
console.log(Math.min(4.5, 5)); // 4.5
console.log(Math.min(19, 9));  // 9
console.log(Math.min(1, 1));   // 1

console.log(Math.random()); // Un nombre aléatoire
console.log(Math.floor(Math.random() * 100 +1));
```

A voir : https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference

2. LES TABLEAUX

Il est fréquent de devoir manipuler des listes de données.

Pour cela, dispose d'un nouveau type de données : **les tableaux**.

Définition d'un tableau

```
const films = ["Retour vers le futur", "Star Wars",
               "Les gardiens de la galaxie"];
```

On crée un tableau à l'aide d'une paire de crochets **[]**

Tout ce qui se trouve entre les crochets correspond au contenu du tableau.

Les différents éléments stockés sont séparés par des virgules.

Avec JavaScript, on peut stocker dans un tableau des éléments de différents types :

```
const tableau = ["Bonjour", 7, 13.5, true];
```

Pour connaitre le nombre d'éléments dans le tableau : **length**

```
console.log("Nombre de films :" + films.length); //3
```

Chaque élément présent dans un tableau est identifié par un numéro, appelé son **indice** (**index** en anglais).

```
//accéder à un élément
const films = ["Retour vers le futur", "Star Wars", "Les gardiens de
 la galaxie"];
console.log(films[0]); // "Retour vers le futur"
console.log(films[1]); // "Star Wars"
console.log(films[2]); // "Les gardiens de la galaxie"
console.log(films[3]); // undefined : le dernier indice valide est 2
```

Attention : L'indice du premier élément d'un tableau est **0 !**

Le plus grand indice utilisable est donc égal à la **taille du tableau - 1**.

Utiliser un indice invalide pour accéder à un élément d'un tableau JavaScript renvoie la valeur spéciale **undefined**

Parcourir un tableau

Il existe plusieurs solutions pour parcourir un tableau élément par élément.
La première consiste à utiliser la boucle for :

```
// boucle for
for (let i = 0; i < films.length; i++) {
  console.log(films[i]);
}
```

Une autre solution consiste à utiliser la méthode forEach() sur le tableau.
Celle-ci permet d'appliquer une fonction sur chaque élément du tableau.

```
//boucle forEach
films.forEach(function(film) {
  console.log(film);
});
```

Une troisième solution pour parcourir un tableau est d'utiliser la boucle for-of, introduite dans la dernière version majeure du langage. Elle a l'avantage de ne pas nécessiter la gestion d'un compteur de boucle.

```
for(const film of films) {
  console.log(film);
};
```

Ajouter un élément dans un tableau

L'ajout d'un nouvel élément dans un tableau se fait avec les méthodes :

push() : pour ajouter l'élément à la fin du tableau ;

unshift() : pour l'ajouter au début du tableau.

```
films.push("Tron"); // Ajoute le film à la fin du tableau
console.log(films[3]); // "Tron"

films.unshift("Johnny Mnemonic"); // Ajoute le film au début du tableau
console.log(films[0]); // "Johnny Mnemonic"
```

Supprimer un élément dans un tableau

La suppression d'un élément dans un tableau se fait avec les méthodes :

pop() : pour supprimer le dernier élément du tableau ;

shift() : pour supprimer le premier élément du tableau.

```
films.pop(); // Supprime le dernier élément
console.log(films.length); // 2
console.log(films[2]); // undefined

films.shift(); // Supprime le 1er element
console.log(films.length); // 1
console.log(films[0]); // "Star Wars"
console.log(films[1]); // "Les gardiens de la galaxie"
```

Autre méthode intéressante, la méthode qui fait : **splice()** *Documentation MDN*

Cette modifie le contenu d'un tableau en retirant des éléments et/ou en ajoutant de nouveaux éléments dans le même.

```
const months = ['Jan', 'March', 'April', 'June'];
months.splice(1, 0, 'Feb'); //ajoute Feb en 2ème position

months.splice( months.length, 0, 'July' ); // = push : ajoute à la fin
months.splice( 0, 0, 'December' ); // = unshift : ajoute au debut

months.splice( months.length-1, 1 ); // = pop : supp le dernier
months.splice( 0, 1 ); // = shift : supp le premier

console.log(months);
```

Cette méthode fait tout, mais les méthode `push()`, `unshift()`, `pop()` et `shift()` sont plus concises, c'est pourquoi on les utilise toujours …

Pour tout connaitre sur la manipulation des tableaux en JS :
https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Array

3. MANIPULATIONS DES CHAINES DE CARACTERES

Récapitulons ce que nous savons déjà au sujet des chaînes de caractères :

- Une valeur de type chaîne (string) permet de représenter un texte.
- En JavaScript, on définit une chaîne en plaçant un texte entre guillemets simples ('je suis une chaîne'), entre guillemets doubles ("je suis une chaîne") ou entre côtes anglaises (`je suis une chaîne`)
- On peut insérer dans une chaîne des caractères spéciaux en utilisant le caractère \ ("antislash" ou "backslash") suivi d'un autre caractère.
  Par exemple, \n définit un retour à la ligne.
- Appliqué à des chaînes de caractères, l'opérateur + déclenche la concaténation (jointure) des deux valeurs.

Longueur d'une chaine

```
// longueur
console.log("ABC".length); // 3
console.log("Je suis une chaîne".length); // 18
const mot = "Kangourou";
const longueurMot = mot.length; // longueurMot contient la valeur 9
console.log(longueurMot); // 9
```

Convertir une chaine en minuscules ou en majuscules

```
//majuscules minuscules
const motInitial = "Bora-Bora";
console.log(motInitial.toLowerCase());  // "bora-bora"
console.log(motInitial.toUpperCase());  // "BORA-BORA"
console.log(motInitial);  // "Bora-Bora"
```

Il est essentiel de comprendre que la chaîne initiale n'est pas modifiée par ces méthodes, ni par aucune autre méthode appelée sur une chaîne de caractères. Toutes les opérations applicables aux chaînes de caractères ne modifient JAMAIS la chaîne initiale, mais renvoient de nouvelles chaînes. On dit que la chaine créée est **immuable** (en anglais : immutable).

Comparer deux chaines

La comparaison entre deux chaînes s'effectue avec l'opérateur ===. Cette opération renvoie une valeur booléenne : `true` si les deux chaînes sont égales, `false` sinon.

```
//comparaison
const chaine = "azerty";
console.log(chaine === "azerty"); // true
console.log(chaine === "qwerty"); // false
```

Attention cependant : la comparaison entre chaînes est sensible à la casse (case sensitive).

```
console.log("Azerty" === "azerty"); // false (à cause du A)
```

La conversion en minuscules ou en majuscules est souvent utilisée pour comparer une valeur saisie par l'utilisateur (donc comportant potentiellement des minuscules et/ou des majuscules) à une valeur prédéfinie.

```
const valeurSaisie = "Quitter";
console.log(valeurSaisie === "quitter");  // false (à cause du Q)
console.log(valeurSaisie.toLowerCase() === "quitter");  // true
```

Les chaines comme ensemble de caractères

Une chaîne de caractères peut être considérée comme un ensemble de caractères.
Chaque caractère est identifié par un numéro, appelé son indice, de base 0 comme pour les tableaux.

```
//Les chaines comme ensemble de caractères
const sport = "Tennis-ballon"; // 13 caractères
console.log(sport[0]); // "T"
console.log(sport[6]); // "-"
console.log(sport[13]); // undefined (longueur dépassée)
```

Nous pouvons utiliser les boucles pour accéder à tous les caractères. Par exemple, un `FOR` car le nombre de tours est connu (c'est la longueur de la chaine) ou un `FOR-OF` :

```
//Boucle FOR
const prenom = "Nadine";
for (let i = 0; i < prenom.length; i++) {
  console.log(prenom[i]);
}

//Boucle FOR-OF
const prenom = "Nadine";
for (const lettre of prenom) {
  console.log(lettre);
}
```

Transformer une chaîne en tableau

La méthode `Array.from()` permet de transformer une chaîne en un véritable tableau :

```
const prenom = "Nadine";
const tabPrenom = Array.from(prenom);
tabPrenom.forEach(lettre => {
  console.log(lettre);
});
```

Rechercher dans une chaîne

La méthode `indexOf()` permet de rechercher une chaine dans une autre. Elle retourne la position (l'indice) de la chaine trouvée.

```
const chanson = "Honky Tonk Women";
console.log(chanson.indexOf("onk")); // 1
console.log(chanson.indexOf("Onk")); // -1 (à cause du O)
```

Les méthodes `startsWith()` et `endsWith()` permettent de rechercher une valeur au début ou à la fin de la chaîne.

```
console.log(chanson.startsWith("Honk")); // true
console.log(chanson.startsWith("honk")); // false
console.log(chanson.startsWith("Tonk")); // false

console.log(chanson.endsWith("men")); // true
console.log(chanson.endsWith("Men")); // false
console.log(chanson.endsWith("Tonk")); // false
```

Décomposer une chaîne en sous-parties

La méthode `split()` permet d'obtenir un tableau composé de toutes les sous-parties d'une chaine qui sont séparées par un caractère particulier (point, tiret, point-virgule, etc).

```
const mois = "Jan,Fev,Mar,Avr,Mai,Jun,Jul,Aou,Sep,Oct,Nov,Dec";
const tabloMois = mois.split(",");
console.log(tabloMois[0]); // "Jan"
console.log(tabloMois[11]); // "Dec"
```