Fonctions / Fonctions Fléchée / return / param / param par defaut / scope

Dans tous les langages de programmation on retrouve le concept de fonctions, il s'agit d'un bloc de code (un sous-programme dans notre programme si l'on veut) qui va contenir plusieurs instructions, de cette manière plutôt que d'écrire plusieurs fois certaines lignes de code, on va les regrouper au sein d'une fonction, de cette manière nous pourrons exécuter ailleurs dans le programme toutes les lignes de code de la fonction

Function

Syntaxe de base pour déclarer une fonction

```
function maSuperFonction(){
    console.log('Hello World');
    console.log(22+33);
}
```

Ensuite quelque part dans le programme il va falloir exécuter la fonction :

```
//! Détailler la fonction OK, mais ne pas oublier
//! d'exécuter au moins une fois dans le programme cette fonction
maSuperFonction();
```

Paramètre

Les fonctions ont aussi un concept de paramètre, dans le cas où les instructions au sein de la fonction ont besoin d'une variable extérieure. Ci-dessous une fonction qui va afficher en console une variable unNom

```
//! Certaines fonction ont besoin de prendre un paramètre ici num
//! Pas besoin de déclarer le paramètre, il sera défini à l'utilisation de
//! la fonction
function fonctionAvecParametre(num){
    console.log('Hello World');
    console.log(22+num);
}
//! Ici notre paramètre num aura pour valeur 9
fonctionAvecParametre(9);
```

La variable num est définie directement lors de l'utilisation de la fonction.

Auteur : Jean-François Pech
Relu, validé & visé par :
Jérôme CHRETIENNE
Sophie POULAKOS
Mathieu PARIS
Date création : 03/03/2023
Date révision : 10/03/2023

Exercice Fonction 1

```
//! EXO 5 : Function
// TODO : créer une fonction qui prend un nombre en paramètre
// TODO : La f° doit afficher en console: 33 + le nombre reçu en paramètre
// TODO : créer une autre fonction qui prend 2 nombres en paramètre
// TODO : Cette seconde f° doit afficher en console l'ADDITION des 2 nombres reçus en paramètre
```

Solution

```
function fonctionUn(unTruc){
    console.log(33+unTruc);
}
fonctionUn(7);
```

💡 On peut aussi prévoir des fonctions nécessitant plusieurs paramètres comme ceci

```
function fonctionDeux(unTruc,unBidule){
    console.log(unBidule+unTruc);
}
fonctionDeux(10,88);
```

Return

Les fonctions gèrent également le concept de return, c'est-à-dire que nous pouvons écrire des fonctions qui vont retourner quelque chose (une variable, un résultat, etc..). Dans les exemples précédents, si on analyse bien, nos fonctions font deux choses techniquement : un calcul ET l'affichage du résultat de ce calcul, mais admettons, nous voulons garder un code clair et précis, on veut une fonction qui fait Uniquement du calcul, l'affichage du résultat sera géré ailleurs dans le programme.
Il faut donc adapter notre fonction pour qu'elle retourne un résultat que l'on stockera dans une variable.

Exemple avec une fonction qui a pour but de soustraire 2 nombres :

```
//! Dans certains cas une fonction doit pouvoir retourner quelquechose
//! le résultat d'un calcul par exemple
//! Ci-dessous on fait une fonction de calcul, notre fonction ne fait que ca
//! Elle se charge JUSTE de faire un calcul
//! L'affichage du résultat se fera en dehors de la fonction
function calculReturn(unNombre, unAutreNombre){
    return unNombre + unAutreNombre
}
//! Ici le calcul qui est return par la fonction est stocké dans une variable
//! resultat
let resultat = calculReturn(22,99);
console.log(resultat);
// ou executer la fonction quand on a besoin
console.log('Le résultat : ', calculReturn(22,99));
```

Paramètre par défaut

Une bonne pratique lorsque l'on écrit des fonctions (surtout en travail collaboratif), consiste à renseigner un paramètre par défaut dans le cas où on oublie de renseigner un paramètre quand on exécute une fonction.

```
//** Bonne Pratique : paramètre par défaut
function fonctionAvecParametre(num=0){
    console.log(22+num);
}
```

Scope

⚠️ Quand on code des fonctions (quand les scripts se complexifient), il est nécessaire de prendre en compte la notion de scope soit la portée des variables.
Une fonction va pouvoir utiliser des variables déclarées globalement mais le programme global ne peut pas utiliser une variable déclarée dans une fonction.

```
// ? La notion de scope (la portée d'une variable)
// ? Dans l'exemple ci-dessous on a 2 fois la même variable testScope1 qui est déclarée
?????
// ? En fait même si elles ont le même nom ce ne sont pas les même espaces mémoires qui
sont alloués
// ? let testScope1 = 99; est dans le scope global de notre programme
// ? let testScope1 = 12; est dans le scope de la fonction
let testScope1 = 99;
function maFonctionTestScope(){
    let testScope1 = 12;
    console.log('scope de la fonction :',testScope1);
};
maFonctionTestScope();
console.log('scope hors de la fonction :',testScope1);
```

Exercice : Quiz 1 Trouver le bug

```
//TODO : Pourquoi ca beug ?
function buggyFunction() {
    let wtf = 9;
    console.log(wtf);
};
buggyFunction();
console.log(wtf);
```

```
▸Uncaught ReferenceError: wtf is not defined          app.js:71
    at app.js:71:13
```

Exercice : Quiz 2 Trouver le bug

```
//TODO : Pourquoi ca beug / Pourquoi ca marche pas ?
let something = 44;
function functionBugParent() {
    let something = 9;
    console.log(something);
    console.log(lesNews);


    function functionBugEnfant() {
        let lesNews = `il est 99h67`;
    }
};
functionBugParent();
console.log(something);
```

```
▶Uncaught ReferenceError: lesNews is not defined    app.js:79
    at functionBugParent (app.js:79:17)
    at app.js:86:1
```

Exercice : Moyenne

```
//! EXO 5.2 : La moyenne de 2 notes
//TODO: Créer une fonction qui calcule la moyenne de 2 notes
//TODO: Afficher le résultat en console
```

Solution

```
let noteSport = 8;
let notePhilo = 2;
let laMoyenne = moyenne2notes(notePhilo,noteSport);
// On peut executer la f° AVANT de la définir (pas d'ordre pour décrire les fonctions)
function moyenne2notes(a,b){
    return (a+b)/2;
};
console.log('La moyenne des 2 notes : ',laMoyenne);
```

https://github.com/jefff404/cours-js/tree/6-functions

Opérateurs de comparaison / condition ternaire

Opérateurs de comparaison

Autre concept qui sera surtout utilisé pour les conditions, ce sont les opérateurs de comparaison cela renvoi un booléen (true ou false).

```
/**
 * ************************************
 * 7- Les opérateurs
 * ************************************
 */
//! Les booléens : 2 états possibles TRUE ou FALSE (vrai ou faux)
let a = 11;
let b = 99;
console.log("variable a:",a);
console.log("variable b:",b);
//! avec == on demande si a est égal à b
console.log("c'est égal ? :",a == b);
//!pour vérifier si a est différent de b on utilise !=
console.log("C'est inégal ? :",a != b);
//! Ensuite on retrouve les même opérateurs qu'en Mathématique
//! ici on demande si a est strictement suppérieur à b
console.log("Strictement suppérieur ? :",a > b);
//! ici on demande si a est strictement inférieur à b
console.log("Strictement inférieur ? :",a < b);
//! ici on demande si a est inférieur ou égal à b
console.log("Inférieur ou égal ? :",a <= b);
//! ici on demande si a est suppérieur ou égal à b
console.log("suppérieur ou égal ?:",a >= b);
//? Attention : de base JS ne prend pas en compte le typage des variables :
//? ci dessous le nombre 2 est égal au caractère "2" 🤔
console.log("le chiffre 2 = \"2\"?:",2 == "2");
//! Pour prendre en compte le type des donnée que l'on compare, on utilise l'opérateur
===
//! c'est l'égalité stricte
console.log("égalité stricte ?:",2 === "2");
//! il y a aussi l'inégalité stricte avec l'opérateur !==
console.log("inégalité stricte ?:",2 !== "2");
```

à noter la différence pour vérifier une égalité entre l'opérateur == et ===, le triple égale va permettre de vérifier aussi si les variables comparées sont du même type.

Condition / opérateur ternaire

Une syntaxe pour écrire des conditions simples qui renvoient quelque chose ou quelque chose d'autre si une condition est remplie ou non. Grâce à l'opérateur « ? »
Avant le ? on décrit notre condition, après le ? nous avons ce qui est return quand la condition est remplie puis « : » et ce qui est return quand la condition n'est pas remplie

```
//!-------CONDITIONS TERNAIRES-------
// ? on combine un opérateur de comparaison et l'opérateur ? pour établir une condition qui return une chose ou une autre chose
// ? cela permet de faire une condition if (simple) avec une syntaxe racourcie
let whatIsYourAge = 6;
console.log(whatIsYourAge >18 ? '🍹':'👮');
// Astuce pour check si une variable est définie (si ya QQchose dans son espace mémoire)
let userPremium;
// On check si une variable est définie la condition c'est juste uneVariable ?
console.log(userPremium?'OK 👍':'Not OK 👺');
// ↑ c'est l'équivalent de ↓
console.log(userPremium ==true?'OK 👍':'Not OK 👺');
// on doit lui assigner QQCHOSE
userPremium = 'YES';
console.log(userPremium?'OK 👍':'Not OK 👺');
```

On peut aussi combiner plusieurs conditions avec les opérateurs
|| (une condition OU une autre condition), && (une condition ET une autre condition)

```
// ? On peut utiliser des operateur aussi pour combiner des conditions && (pour ET) || (pour OU)
console.log(3==3&&3<4);
let typeUtilisateur = 'Extra';
console.log(typeUtilisateur == 'Extra' || typeUtilisateur == 'Premium');
```