chapitre 4 : Les fonctions

4.1 Déclaration d'une fonction

Si on redéfinit 2 fois la même fonction, seul la 2e sera conservée.

4.1.1 Déclaration Standard

```
function maFonctionQuiSertARien(x){
    return x;
}
```

4.1.2 Déclaration comme une expression fonctionnelle

En javascript les fonctions peuvent être placées en arguments d'autres fonctions.

> attention aussi au hissage et à l'utilisation de var. Car plus de hissage comme mais une TDZ (du au fait de la déclaration avec let)

```
let affiche = fonction (x) {...};
let affiche = fonction NOM_INTERNE(x) {...}; // équivalent à la première. rien ne change car le nom interne est facultatif
```

4.1.3 Constructeur Fonction

Attention, ici la fonction est un objet (pas super ***Clean Code***)

```
let affiche = new Function ("param1","param2",...,"code");
```

4.2 Les Paramètres

4.2.1 Paramètre Formel vs Paramètre Effectif

Un ***paramètre formel*** est le **paramètre générique** utilisé lors de la déclaration de la fonction. Tandis que le ***paramètre effectif*** sera la variable placé en argument de la fonction lors de son appel.

4.2.2 Vérification de type

Le Javascript n'effectue aucune vérification sur les types de paramètres placés en argument

4.2.3 Vérification du nombre

Le Javascript n'effectue aucune vérification du nombre de paramètres d'une fonction.

- Si il y en à **trop** comparé à ce que demande la fonction, ils seront **ignorés**
- Si il n'y en a **pas assez**, les paramètres manquants seront remplacés par undefined

4.2.4 Passage par valeur VS Passage par Référence

Le ***passage par valeur*** est une "**copie**" de la valeur de la variable placé en argument. La modification dans la fonction de la "copie" n'aura pas d'incidence sur son original.

Le ***passage par référence*** est le passage de la référence de la variable placé en paramètre. Toute modification sur l'entité aura pour conséquence de modifier l'entité originale. (ex :objet, fonction, Array , ...)

> Les chaînes de caractères sont des types primitifs et donc passés par valeur

Fonction de base

un code décrit sous la forme d'un objet fonction doit être évalué à chaque appel et est donc peu efficace (Clean Code : a éviter):

```
const maFonction = new function("var1","var2","return var1+var2;");
```

fonctions avec un nom interne (facultatif) :

```
let affiche = function osefNom (var,var){return var +var;}
```

Attention aux déclarations hoistées et donc non initialisées

Arguments

Aucune vérification sur les types n'est effectuée

Surcharge et trop peu de paramètres

Si une fonction est appelée avec plus d'arguments que nécessaire, ils seront ignorés (surcharge)

à l'inverse, si il y en a trop peu, on les remplace par undefined.

Passage des arguments

Les types primitifs effectuent un passage par valeur. càd une copie de la valeur de la variable. tandis que les objets sont passées par référence et donc les originaux sont envoyés en paramètre.

Return avec multiples arguments

```
sommeProduit(x,y){
    let somme = x+y;
    let produit = x*y;
    return {somme,produit}
};//retourne un objet {somme:VAL,produit:VAL}
```