ES6/7

const let var

```
DIFFERENCE ENTRE LET VAR ET CONST

Une fois déclaré et défini une variable étant déclarée et définie étant const ne peut pas être modifiée
ATTENTION les attributs d'un objet déclaré en const peuvent être modifiées.

// 2. Déclaration dans un bloc {}
if (true) {            // début du bloc
    var maVariable1;   // déclaration de la variable
    let maVariable2;   // déclaration de la variable
    const maVariable3; // déclaration de la variable
}                      // fin du bloc mais pas de la portée de maVariable1

alert(maVariable1);    // ne soulève pas d'erreur
alert(maVariable2);    // erreur : la variable est hors de sa portée
alert(maVariable3);    // erreur : la variable est hors de sa portée

// 2. Déclaration dans une fonction

function maFunction() { // début de la fonction
    var maVariable4;    // déclaration de la variable
    let maVariable5;    // déclaration de la variable
    const maVariable6;  // déclaration de la variable
}                       // fin de la fonction et de la portée des variables

alert(maVariable4);     // erreur : la variable est hors de sa portée
alert(maVariable5);     // erreur : la variable est hors de sa portée
alert(maVariable6);     // erreur : la variable est hors de sa portée
```

```
let x = 5 ;
console.log (x) => 5
——-
let x ;
x=7 ;
console.log(x) => 7
———-
```

```
const x;
```
=> donne une erreur
Il faut assigner une valeur à x dès le début

const = on donne une valeur qu'une seule fois (pas de nouvelle valeur)
et il est immutable (on ne peut pas le changer)

```
const x = {
     name : "JB Cavarec",
}
```

x pointe sur un objet , on peut changer sa propriété

```
x.name = "pas toi un autre" ….
console.log(x);
```

fonctions fléchées

argument => retour

( ) => val équivaut à function( ) { return val ; }

```
(arg1,arg2) =>  return val
```

équivaut à

```
function( arg1,arg2) { return val ; }
```

paramètres par défaut

```
     function nom(prénom, nom="toto") { … }

function multiply(a, b = 1) {
     return a * b;
}

console.log(multiply(5, 2));
// Expected output: 10

console.log(multiply(5));
// Expected output: 5
```

rest

Prend les arguments dans des valeurs libres pour les transformer en Array

exemple:

```
fonction sommeTableau(…nombres) {
    // va créer un tableau nombres et y insérer les valeurs libres
}
```

spread

```
// l' operateur spread ... déconstruit un tableau
const arr = [ "pierre","paul","jacques"]
//console.log(arr)
//console.log(...arr)

// permet de copier un tableau ainsi
const arr2 = [...arr]
//console.log(arr2)

// permet de joindre 2 tableaux
const arr3 = ["henri","emile"]
const jointure = [...arr,...arr3]
console.log(jointure)
```

Différents for

```
// on accede au tableau par les indice (ici i)
    for (let i = 0; i < arr.length; ++i) {
        console.log(arr[i]);
    }

    for (let i in arr) {
        console.log(arr[i]);
    }


    arr.forEach((v, i) => console.log(v,i));
```

backtick

évite la concatenation +

`texte... ${variable}...texte`

destructuring Array

soit un tableau

```
nombres = [1,2,3]
```

l'action

```
[a,b,c] = nombres
```

donnera:

```
a=1
b=2
c=3
```

exemple de swap:

```
let a = 1 ;
let b = 2 ;
[b,a] = [a,b] ;
```

// le destructuring permet d'inverser 2 valeurs par exemple.

destructuring Object

comme tableau mais ainsi

```
const me = { name: "Jean", age: 12 }

const { name , age } = me

console.log(age,name)
```

pas d'ordre dans les objets, ici le nom des propriétés doit correspondre