CLOSURES

Fonction qui enferme des variables provenant d'un scope parent

Exemple:

```
function compteur() {
    let count = 0;

    return function() {
          return count++;
    };
}

let plusUn = compteur();

plusUn( )
plusUn( )
```

Exemple:

```
fonction multiplyBy(number) {
     const closedVar = number ;
     return function(otherNumber) {
          return closedVar * otherNumber ;
     }
}

let mult_deux = multiplyBy(2) ;

let mult_trois = multiplyBy(3) ;



mult_deux(3)

// 6

mult_trois(3)

// 9
```

exercice :

Créer une fonction calcul2(number, operation) qui renvoi une fonction qui effectue une operation (addition , multiplication , division, modulo) sur le même principe de closure.

```
var f = calcul2(5,"+") {
     …
}
```

à compléter…

```
function closure(){
    function first() { console.log('I was declared first')}
    function second() { console.log('I was declared second')}
    function third() { console.log('I was declared third')}
    return [first, second, third]
}

let f = closure()

let one = f[0]
let two = f[1]
let three = f[2]

one() // logs I was declared first
two() // logs I was declared second
three() // logs I was declared third
```

Les closures nous permettent d'utiliser des fonctions pour créer d'autres fonctions qui ajoutent une valeur spécifique à leur argument. Dans ce cas, la fonction parent permettant ce comportement est connue comme une function factory (fonction usine) car elle crée essentiellement d'autres fonctions.

Avant l'introduction des classes dans l'ES6, les closures permettent de créer un moyen de créer une confidentialité de classe similaire à celle utilisée dans la programmation orientée objet, ce qui nous permettait d'émuler des méthodes privées.

Les closures permettent de faire du Currying (programmation fonctionnelle)
… on verra plus tard

```
> let greeting = (a) => (b) => a + ' ' + b

  greeting('Hello There')('General Kenobi')
  //returns Hello There General Kenobi
<· 'Hello There General Kenobi'
> let toto = function(a) { return function(b) { return a+" "+b }}
<· undefined
> toto('coin')('couain')
<· 'coin couain'
>
```