Les fonctions anonymes et récursives en JavaScript

Table des matières

I. Contexte **3**

II. Fonctions récursives JavaScript **3**

III. Exercice : Appliquez la notion **5**

IV. Fonctions anonymes JavaScript **5**

V. Exercice : Appliquez la notion **9**

VI. Arrows functions **9**

VII. Exercice : Appliquez la notion **11**

VIII. Auto-évaluation **12**

A. Exercice final 12

B. Exercice : Défi 15

Solutions des exercices **16**

I. Contexte

Durée : 45 min

Environnement de travail : Repl.it

Pré-requis : Avoir suivi le cours sur les fonctions JavaScript

Contexte

Dans ce cours, nous allons explorer des notions avancées sur l'utilisation des fonctions en JavaScript, comme les fonctions s'appelant elles-mêmes (fonctions récursives) et les fonctions anonymes qui, comme leur nom l'indique, sont des fonctions pour lesquelles nous n'allons pas préciser de nom.

II. Fonctions récursives JavaScript

Objectif

- Apprendre ce qu'est une fonction récursive.

Mise en situation

Dans certains cas de développement JavaScript, nous sommes amenés à créer une fonction récursive, c'est-à-dire qui va s'appeler elle-même. Par exemple, pour parcourir une arborescence de fichiers afin de trouver un fichier spécifique, pour réaliser un calcul, etc.

Remarque

Les fonctions récursives peuvent s'apparenter à des boucles.

Exemple **Un exemple simple**

Dans les deux exemples ci-dessous, nous calculons un pourcentage de chargement : le résultat est le même, car la console affiche "Chargement terminé !" dans les deux cas. Seule l'approche est différente.

Dans la fonction `loading`, qui est appelée une seule fois, nous mettons en place une boucle qui incrémente la valeur `percent` tant qu'elle est inférieure à 100, puis nous affichons le message.

Dans la fonction `loadingRecursive`, si la valeur en pourcent (`percent`) est inférieure à 100, alors on l'incrémente, puis on appelle à nouveau la fonction en lui passant la nouvelle valeur, et ainsi de suite jusqu'à ce qu'on ne passe plus dans la condition et que le message de la console soit donc affiché.

```
function loading (percent) {
while (percent < 100) {
percent++
}
console.log('Chargement terminé !')
}

function loadingRecursive (percent) {
if (percent < 100) {
percent++
loadingRecursive(percent)
} else {
console.log('Chargement terminé !')
}
}
```

```
16
17 loading(0)
18 loadingRecursive(0)
```

Exemple **Afficher un compte à rebours**

```
1  function countDown (number) {
2    console.log(number)
3
4    if (number > 0) {
5      number--
6      countDown(number)
7    }
8  }
9
10 countDown(5)
```

Résultats successifs dans la console :

5

4

3

2

1

0

Attention **L'utilisation de return**

Attention, les fonctions récursives sont à utiliser avec beaucoup de précautions. Lorsqu'une fonction retourne une valeur, il faut aussi que sa récursivité soit retournée.

Pour mieux comprendre, prenons un exemple : une fonction qui concatène un nombres aléatoire tant que celui-ci est différent de 0.

La méthode Math.random() * (11) + 0; permet de retourner une valeur entre une valeur min (incluse, dans ce cas 0) et une valeur max (exclue, ici 11). La méthode Mathfloor(x) renvoie le plus grand entier qui est inférieur ou égal à un nombre x. La combinaison des deux méthodes nous retourne aléatoirement un entier entre 0 et 10.

```
1  function concatenation(concat) {
2    let aleatoryNumber = Math.floor(Math.random() * (11)) + 0;
3    if (aleatoryNumber !== 0) {
4      concat += `${aleatoryNumber} `
5      return concatenation(concat); // on retourne la récursivité
6    } else {
7      return concat
8    }
9  }
10
11 console.log(concatenation('')) // un tableau de chiffre aléatoire.
```

Maintenant, cette même fonction sans le `return` sur la récursivité retournera `undefined`.

```
1  function concatenation(concat) {
2    let aleatoryNumber = Math.floor(Math.random() * (11)) + 0;
3    if (aleatoryNumber !== 0) {
4      concat += `${aleatoryNumber} `
5      concatenation(concat); // on retorune la récursivité
6    } else {
```

```
    return concat
  }
}

console.log(concatenation('')) // undefined
```

Syntaxe **À retenir**

- Une fonction récursive est une fonction qui s'appelle elle-même.

Complément

Algorithme récursif (Wikipédia)[1]

La récursivité (MDN Web docs)[2]

III. Exercice : Appliquez la notion

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :

[3]

Question

[solution n°1 p.17]

Pour éditer le bulletin de notes d'une classe, nous avons besoin d'extraire la meilleure note. À l'aide d'une fonction récursive, créez un algorithme qui trie les éléments du tableau et retourne la plus haute.

```
const notes  = [10, 28, 15, 17, 32, 5, 12, 4];
```

La fonction récursive que nous souhaitons faire est une fonction qui compare une variable qui stocke la meilleure note avec la valeur de l'index du tableau :

- Si la valeur de l'index du tableau est supérieure à la meilleure note, alors la meilleure note devient celle-ci.

Pour augmenter l'index du tableau à comparer, vous pouvez créer une variable compteur qui s'incrémentera avant chaque rappel de la fonction.

La condition de fin de la fonction est lorsque la taille du tableau de notes est égale au compteur.

Cette fonction récursive retourne une variable `higher`. N'oubliez pas de retourner la récursivité.

Indice :

Commencez par définir vos variables.

Il y en aura 3 :

- Le tableau de note.
- Le compteur `counter` qui commence à zéro.
- La valeur la plus petite que peut prendre `higher` est zéro.

IV. Fonctions anonymes JavaScript

1 https://fr.wikipedia.org/wiki/Algorithme_r%C3%A9cursif
2 https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide/Fonctions#La_r%C3%A9cursivit%C3%A9
3 https://repl.it/

Objectif

- Apprendre ce qu'est une fonction anonyme, quand l'utiliser et comment la déclarer

Mise en situation

Dans certains cas, nous avons besoin de créer des blocs d'instructions à exécuter suite à un événement (ex : clic sur un bouton) ou pour un usage unique, sans avoir besoin de créer une fonction nommée, et donc d'avoir à l'appeler par son nom. Nous allons pour cela utiliser les fonctions anonymes.

Les fonctions anonymes sont, comme leur nom l'indique, des fonctions qui ne vont pas posséder de nom.

En effet, lorsqu'on crée une fonction, nous ne sommes pas obligés de lui donner un nom à proprement parler.

Généralement, on utilise les fonctions anonymes lorsqu'on n'a pas besoin d'appeler notre fonction par son nom c'est-à-dire lorsque le code de notre fonction n'est appelé qu'à un endroit dans notre script et n'est pas réutilisé.

En d'autres termes, les fonctions anonymes vont très souvent simplement nous permettre de gagner un peu de temps dans l'écriture de notre code et (bien que cela porte à débat) à le rendre plus clair en ne le polluant pas avec des noms inutiles.

Création et exécution ou appel d'une fonction anonyme

On va pouvoir créer une fonction anonyme de la même façon qu'une fonction classique, en utilisant le mot clef **function** mais en omettant le nom de la fonction après.

```
<code>
function(){
  alert('Je suis une fonction anonyme !')
}
</code>
```

Ci-dessus on déclare une fonction anonyme dont le rôle est de déclencher une fonction alert().

On a vu dans le module précédent que pour déclencher une fonction il fallait l'appeler, mais comment appeler une fonction qui ne possède pas de nom ? Pour cela trois méthodes:

- Enfermer le code de notre fonction dans une variable et utiliser la variable comme une fonction
- Utiliser un évènement pour déclencher l'exécution de notre fonction
- Auto-invoquer notre fonction anonyme

Méthode **Stocker la fonction anonyme dans une variable**

Avec cette technique, nous allons créer une variable ayant comme valeur la fonction anonyme.

Ensuite, quand nous voudrons faire s'exécuter notre fonction, nous utiliserons notre variable suivie de parenthèses (qui vont lancer l'exécution).

Exemple

Si nous reprenons notre exemple permettant d'afficher un message de bienvenue, nous obtenons le résultat ci-dessous en utilisant une variable.

La fonction n'est exécutée que lorsque la variable est appelée suivie de parenthèses.

Ici, elle peut donc être appelée plusieurs fois puisqu'il suffit de réutiliser la variable, comme dans l'exemple ci-dessous où elle est déclenchée trois fois.

```
let showHello = function () {
      alert('Bienvenue !')
    }

    showHello()
    showHello()
    showHello()
```

Le même exemple en ajoutant le paramètre `name`, permettant ainsi de personnaliser notre message de bienvenue :

```
      let showHello = function (name) {
      alert('Bienvenue ! ' + name)
      }

     showHello('John')
     showHello('Jane')
```

Complément **Fonction nommée vs fonction anonyme stockée dans une variable**

Il n'y a aucune différence de fonctionnement entre une fonction nommée et une fonction anonyme stockée dans une variable. La différence est uniquement syntaxique.

Méthode **Fonction anonyme exécutée suite au déclenchement d'un événement**

Nous verrons en détails dans un autre cours ce qu'est la programmation événementielle en JavaScript, nous allons juste montrer ici comment se fait l'association d'une fonction anonyme à un événement.

Exemple

Dans le code HTML, nous avons créé un bouton permettant à l'internaute d'indiquer qu'il accepte les conditions générales de vente.

```
<!DOCTYPE html>
<html lang="fr">
<head>
  <meta charset="UTF-8">
  <title>Fonctions anonymes</title>
</head>
<body>
  <button id="cgv">J'accepte les CGV</button>

  <script src="main.js"></script>
</body>
</html>
```

Nous voulons lui afficher un message de confirmation lorsqu'il clique sur celui-ci.

Nous affectons notre élément bouton à la variable `cgvButton`, puis nous associons notre fonction anonyme à l'événement `click` du bouton à l'aide de la méthode *`addEventListnener()`*.

```
cgvButton.addEventListener('click', function () {
```

Cette ligne de code peut être traduite en langage oral par “on vient écouter (Listener) notre bouton en attente d’un évènement (event), cet événement doit être de type ‘click’ pour déclencher notre fonction anonyme function()”.

Ainsi, lors du clic sur celui-ci, notre méthode sera exécutée et le message de confirmation sera affiché à l'internaute. Elle peut donc être exécutée plusieurs fois, puisqu'elle le sera à chaque clic sur le bouton.

```
let cgvButton = document.getElementById('cgv')

cgvButton.addEventListener('click', function () {
  alert('Vous venez d\'accepter nos CGV')
})
```

Attention

Il faut parfois différencier bonnes pratiques et clarté de code. Imaginons que le code de la fonction déclenchée au clic fasse 200 lignes. Il sera alors peut-être plus intéressant d'utiliser une fonction nommée et de rassembler les méthodes d'actions et les fonctions à deux endroits séparés. Il faut trouver le juste milieu entre clarté de code et gain de lignes.

Méthode **Fonction anonyme auto-exécutée**

Dans ce cas, la fonction est exécutée dès sa création. Pour la déclarer, on englobe la fonction entière entre parenthèses (avant `function` et après l'accolade fermante `}`), elles-mêmes suivies de parenthèses (permettant de l'exécuter).

Cela peut être utilisé si l'on souhaite exécuter une seule fois un bloc de code dès sa création et sans avoir à le réutiliser, comme dans l'exemple ci-dessous où nous affichons un message de bienvenue lorsque l'internaute arrive sur le site.

Exemple

La fonction ci-dessous est exécutée dès sa création :

```
(function () {
  alert('Bienvenue !')
})()
```

Vous pouvez noter deux choses à propos des fonctions auto-invoquées. Tout d'abord, vous devez savoir que la notion d'auto-invocation n'est pas réservée qu'aux fonctions anonymes : on va tout à fait pouvoir auto-invoquer une fonction qui possède un nom.

Cependant, en pratique, cela n'aura souvent pas beaucoup d'intérêt (puisque si une fonction possède un nom, on peut tout simplement l'appeler en utilisant ce nom).

Ensuite, vous devez bien comprendre que lorsqu'on auto-invoque une fonction, la fonction s'exécute immédiatement et on n'a donc pas de flexibilité par rapport à cela : une fonction auto-invoquée s'exécutera toujours juste après sa déclaration.

Syntaxe **À retenir**

Une fonction anonyme est une fonction ne possédant pas de nom et ne pouvant donc pas être appelée grâce à celui-ci.

Elle peut :

- Être affectée à une variable, qui sera alors utilisée pour appeler la fonction en ajoutant des parenthèses à la suite,
- Être exécutée suite à un événement (clic sur un bouton, etc.).
- Être auto-exécutée.

Complément

Fonction anonyme (Wikipédia)[1]

Expressions de fonctions immédiatement invoquées[2]

IIFE (Immediately Invoked Function Expression)[3]

Fonctions fléchées[4]

Lambda-calcul (Wikipédia)[5]

V. Exercice : Appliquez la notion

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :

[6]

Question

[solution n°2 p.18]

Écrivez une fonction anonyme qui change les lettres d'un mot en majuscules.

VI. Arrows functions

Objectif

- Apprendre ce qu'est une `arrow function` et savoir l'utiliser

Mise en situation

Le JavaScript est issu du langage ECMAScript, qui évolue tous les ans. Chaque année, de nouvelles spécifications sont apportées, mais en règle générale on se base sur l'Ecmascript 5 ou l'EcmaScript 6. Jusqu'à présent, nous avons utilisé l'ES5, mais nous allons apprendre comment déclarer une fonction en ES6. Il est impératif de connaître cette notation.

Méthode **Méthode : Fonction fléchée (arrow function)**

Une expression de fonction fléchée donne la possibilité d'obtenir une syntaxe plus courte pour nos fonctions .

Fonction nommée :

```
Régulière :
<code>
function sayHello() {
    return "Hello World!"
}

</code>
ES6 :
<code>
sayHello = () => {
```

1 https://fr.wikipedia.org/wiki/Fonction_anonyme
2 https://fr.wikipedia.org/wiki/JavaScript#Expressions_de_fonctions_imm%C3%A9diatement_invoqu%C3%A9es
3 https://developer.mozilla.org/fr/docs/Glossaire/IIFE
4 https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Fonctions/Fonctions_fl%C3%A9ch%C3%A9es
5 https://fr.wikipedia.org/wiki/Lambda-calcul
6 https://repl.it/

```
    "Hello World!"


</code>
```

Méthode **Forme raccourcie de arrow function**

Lorsque le retour d'une fonction tient en une action (comparaison, concaténation, etc.), il est possible d'utiliser la forme raccourcie de l'arrow function. Cette forme permet à la fonction de tenir sur une ligne, supprime les {} et le terme *return.*

```
</code>
ES6 :
<code>
sayHello = () => "Hello World!";

</code>
```

Exemple **Fonction affectée à une variable**

ES5 :

```
const hello = function(name){
  return ('Bienvenue ! ' + name);
}

const name = hello('Jane');
```

ES6 :

```
const hello = (name) => {
  return ('Bienvenue ! ' + name);
}

const name = hello('Jane');
```

Exemple **Fonction associée à un événement**

ES5 :

```
let cgvButton = document.getElementById('cgv')

cgvButton.addEventListener('click', function() {
  alert('Vous venez d\'accepter nos CGV')
});
```

ES6 :

```
let cgvButton = document.getElementById('cgv')

cgvButton.addEventListener('click', () => {
  alert('Vous venez d\'accepter nos CGV')
});
```

Exemple **Exemple : Fonction auto-exécutée**

ES5 :

```
1  (function(){
2    alert('Bienvenue !')
3  })()
```

ES6 :

```
1  (() => {
2    alert('Bienvenue !')
3  })()
```

Exemple **Utilisation à partir de la méthode filter() de l'objet Array()**

La méthode filter() sur tableau crée un nouveau tableau avec des éléments qui répondent à un critère donné à partir d'un tableau existant :

Dans l'exemple suivant chaque valeur de l'array studentAge sera traitée indépendamment en tant que age et sera comparer à > 15

```
1 const studentsAge = [17, 18, 15, 16, 13];
2 const ageSup15 = studentsAge.filter(age => age > 15);
3 console.log(ageSup15) // [17, 18, 16];
```

Syntaxe **À retenir**

- L'`arrow function` est issue de l'ES6.
- Cette syntaxe permet de raccourcir les lignes de code et d'améliorer la lisibilité.
- Il est possible d'utiliser simultanément, dans un même code, l'ES5 et l'ES6.
- Pour que la notation simplifiée de l'`arrow function` fonctionne, il faut que l'instruction de retour tienne sur 1 ligne.

Complément

Fonctions fléchées[1]

Lambda-calcul (Wikipédia)[2]

VII. Exercice : Appliquez la notion

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :

[3]

1 https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Fonctions/Fonctions_fl%C3%A9ch%C3%A9es
2 https://fr.wikipedia.org/wiki/Lambda-calcul
3 https://repl.it/

Question 1

[solution n°3 p.18]

On vous demande de moderniser le code d'une application, à partir de vos connaissances, transformez le code suivant pour qu'il corresponde aux nouvelles spécifications ES6.

```
<code>
function sum(num1, num2){
    return num1 + num2
}

console.log(sum(40,2)) // retourne dans la console 42
</code>
```

Question 2

[solution n°4 p.18]

On vous demande de moderniser le code d'une application, à partir de vos connaissances, transformez le code suivant pour qu'il corresponde aux nouvelles spécifications ES6.

```
<code>
function stringLength(str){
    let length = str.length
    console.log(`the length of "${str}" is:`, length)
    return str.length
}

stringLength("willynilly")

</code
```

Question 3

[solution n°5 p.19]

On vous demande de moderniser le code d'une application, à partir de vos connaissances, transformez le code suivant pour qu'il corresponde aux nouvelles spécifications ES6.

Pour aller plus loin, recherchez dans la doc Array.prototype.forEach()

```
<code>
const array1 = ['a', 'b', 'c'];

for (const element of array1) {
  console.log(element);
}

// expected output: "a"
// expected output: "b"
// expected output: "c"

</code>
```

VIII. Auto-évaluation

A. Exercice final

Exercice 1

[solution n°6 p.19]

Exercice

Une fonction anonyme est une fonction qui s'appelle elle-même.

- ○ Vrai
- ○ Faux

Exercice

Que retournera ce code ?

```
function increment(counter) {
  if (counter === 10){
    return counter;
  } else {
    counter++;
    increment(counter)
  }
}

console.log(increment(0));
```

- 10
- undefined
- null
- 0

Exercice

Pour que ce code retourne 10, qu'aurait-il fallu faire ?

- C'est impossible
- Retourner aussi la fonction récursive

Exercice

Que pourra retourner ce code ?

```
const randomNumber = ()=> {
  return Math.floor(Math.random() * (9 + 1)) + 1
}

console.log(randomNumber());
```

- [ ] 10
- [ ] 0
- [ ] 8
- [ ] 12

Exercice

Une fonction anonyme peut être aussi une fonction récursive.

- Vrai
- Faux

Exercice

Que retournera la console ?

```
(() => {
  const hello = 'Hello world';
  console.log(hello);
})()

```

- La console ne retourna rien, la fonction n'est pas initialisée
- Syntaxe error
- Hello world
- null

Exercice

Les deux fonctions ci-dessous retourneront le même résultat.

```
const fonction1 = () => {
  return 2;
};

function fonction2() {
  return 2;
};
```

- Vrai
- Faux

Exercice

Que se passera-t-il si on exécute ce code ?

```
() => {
  console.log(2);
}
```

- La console retournera 2
- Rien, on ne peut pas appeler cette fonction
- Le navigateur renverra une erreur, le code ne pourra pas s'exécuter
- Le code fera une boucle infinie qui appelle la fonction

Exercice

Quel chiffre apparaîtra en premier dans la console ?

```
(() => {
  let counter = 0
  for (let i = 0; i < 1000001; i++) {
    counter = i
  }
  console.log(counter)
})()

console.log(2)
```

- 1000000
- 2

Exercice

Comment écrit-on la fonction suivante sous forme d'une constante en ES6 ?

```
function empty(){}
```

B. Exercice : Défi

Dans une application de type CMS permettant de créer un site web à partir d'une interface intuitive, un utilisateur peut ajouter des blocs (`div`) pour éditer du contenu. Ces blocs seront stockés dans un objet JavaScript comme suit :

```
const page = {
   blocs: [] // stock une liste de blocs
}

Object Bloc = {
   id : string;
   blocs: [] // stock une liste de blocs
}
```

La page stocke une liste de blocs, et chaque bloc peut stocker lui-même une liste de blocs.

À chaque fois que l'utilisateur ajoute un bloc à un autre bloc, celui-ci se stocke dans la liste des blocs de l'élément. L'objet peut donc évoluer dans le temps.

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :

1

Question

[solution n°7 p.22]

On vous demande de créer un `input select` qui affichera la liste de tous les `id` de tous les blocs de la page. Comme vous ne connaissez pas l'objet à l'avance, une boucle `for` est impossible pour effectuer cette tâche. On pourrait penser que ce problème peut être résolu par une boucle `while`, mais la résolution par ce biais paraît implicitement trop complexe pour que nous puissions retenir cette solution.

La seule solution qu'il vous reste est une fonction récursive, qui sera appelée tant que votre boucle ne sera pas passée dans chaque tableau de chaque bloc.

Vous stockerez les `id` dans un tableau JavaScript.

Pour vous aider, voici un exemple d'objet simple sur lequel vous pouvez tester votre algorithme dans un premier temps :

```
const page = {
  blocs: [
    {
      id: 'block1',
      blocs: [
        {
          id: 'block3',
          blocs: []
        },
        {
          id: 'block4',
          blocs: [
            {
              id: 'block7',
              blocs: []
            }
          ]
        }
      ]
    },
```

1 https://repl.it/

```
    {
      id: 'block2',
      blocs: [
        {
          id: 'block5',
          blocs: [
            {
              id: 'block6',
              blocs: []
            }
          ]
        }
      ]
    }
  ]
}
```

Une fois votre algorithme testé à partir d'un objet simple, copiez/collez le code ci-dessous et testez votre algorithme sur l'objet `page` retourné par la fonction `createPage()`.

`createPage()` crée un objet `page` possédant un nombre de blocs aléatoire.

```
createPage = () => {

  createBlocs = (blocs) => {
    const numberBlocks = Math.floor(Math.random() * 5) + 1
    for (let i = 0; i < numberBlocks; i++) {
      blocs.push({id: compteur, blocs: []})
      compteur++;
    }
  }

  const page = {blocs: []}
  let compteur = 1;
  createBlocs(page.blocs)
  page.blocs.forEach(bloc => {
    createBlocs(bloc.blocs)
    bloc.blocs.forEach(bloc => {
      createBlocs(bloc.blocs)
      bloc.blocs.forEach(bloc => {
        createBlocs(bloc.blocs)
      })
    })
  })
  return page
}


const page = createPage();
console.log(page);

```

Solutions des exercices

p. 5 **Solution n°1**

```
<code>
// Tableau de note
const notes  = [10, 28, 15, 17, 32, 5, 12, 4];

//On démarre à zéro le compteur pour tester toutes les valeurs du tableau notes[counter]
let counter = 0;

// La note la plus basse sur un bulletin est zéro
let higher = 0;


function getNote(counter, notes, higher) {

    // On check si il reste des valeurs à comparer dans le tableau
                            if (counter < notes.length) {
    // si oui on compare notre valeur de higher avec la nouvelle valeur
                           if(higher < notes[counter]){
       higher = notes[counter];
     }
     counter ++
     return getNote(counter, notes, higher)

                           } else {
    //sinon on retourne notre valeur
                           return higher;
                           }
                          }



                          console.log(getNote(counter, notes, higher)) // retourne 32 dans la console.

</code>

Pour allez plus loin :

<code>
// Tableau de note
const notes  = [10, 28, 15, 17, 32, 5, 12, 4];

//On démarre à zéro le compteur pour tester toutes les valeurs du tableau notes[counter]
let counter = 0;

// La note la plus basse sur un bulletin est zéro
let higher = 0;


function getNote(counter, notes, higher) {

   // On check si il reste des valeurs à comparer dans le tableau
                            if (counter < notes.length) {
         // si oui on compare notre valeur de higher avec la nouvelle valeur
                           higher = higher > notes[counter] ? higher : notes[counter];
                            counter++;
```

```
                                return getNote(counter, notes, higher)
                                } else {
          //sinon on retourne notre valeur
                                 return higher;
                               }
                             }



                             console.log(getNote(counter, notes, higher)) // retourne 32 dans la
console.
</code>

```

p. 9 Solution n°2

```
const upper = function (word) {
  return word.toUpperCase();
}

console.log(upper('Hello'))
```

p. 12 Solution n°3

Réponses possibles ;

```
<code>
const sum = (num1, num2) => num1+num2
</code>
```

Vous avez besoin des parenthèses autour des arguments car il y en a plus d'un. On peut se passer des {} et du `return` car il y a une seule ligne de commande.

Valide aussi :

```
<code>
const sum = (num1, num2) => {return num1+num2}
<code>
```

Valide aussi avec `let` tout dépend de si la variable sera redéfinie plus tard ou pas.

p. 12 Solution n°4

```
<code>
const stringLength = str => {
    let length = str.length
    console.log(`the length of "${str}" is:`, length)
    return str.length
}
stringLength("willynilly")
</code>
```

Les parenthèses autour de l'argument *str* ne sont pas obligatoires car argument unique. Les {} et le `return` sont obligatoires car la fonction est une fonction multi-lignes.

p. 12 Solution n°5

```
<code>
const array1 = ['a', 'b', 'c'];

array1.forEach(element => console.log(element));

// expected output: "a"
// expected output: "b"
// expected output: "c"

</code>
```

Exercice p. 12 Solution n°6

Exercice

Une fonction anonyme est une fonction qui s'appelle elle-même.

- ○ Vrai
  Une fonction qui s'appelle elle-même est une fonction récursive.
- ◉ Faux

Exercice

Que retournera ce code ?

```
function increment(counter) {
  if (counter === 10){
    return counter;
  } else {
    counter++;
    increment(counter)
  }
}

console.log(increment(0));
```

- ○ 10
- ◉ undefined
- ○ null
- ○ 0

Exercice

Pour que ce code retourne 10, qu'aurait-il fallu faire ?

- ○ C'est impossible
- ◉ Retourner aussi la fonction récursive

Exercice

Que pourra retourner ce code ?

```
const randomNumber = ()=> {
  return Math.floor(Math.random() * (9 + 1)) + 1
}

console.log(randomNumber());
```

- [x] 10
- [ ] 0
- [x] 8
- [ ] 12

La méthode pour retourner un nombre dans un intervalle donné est :

```
const randomNumber = (min, max){
  return Math.floor(Math.random() * (max - min + 1)) + 1;
}
// donc Math.random() * (9 + 1) <=> Math.random() * (10 - 1 + 1)
// donc randomNumber retournera un nombre aléatoire entre 1 et 10 (1 et 10 inclu)
```

Exercice

Une fonction anonyme peut être aussi une fonction récursive.

- (•) Vrai
- ( ) Faux

  Une fonction anonyme est une fonction, à ce titre, elle peut aussi s'appeler elle-même.

Exercice

Que retournera la console ?

```
(() => {
  const hello = 'Hello world';
  console.log(hello);
})()

```

- ( ) La console ne retourna rien, la fonction n'est pas initialisée

  Une fonction anonyme qui est initialisée comme ceci sera appelée une fois à l'initialisation du code.
- ( ) Syntaxe error

  Il n'y a aucune erreur de syntaxe.
- (•) Hello world
- ( ) null

  Pour que la console retourne `null`, il faudrait qu'on ait déclaré une variable sans l'initialiser :

  `const hello ;`

  `console.log(hello ) // null`

Exercice

Les deux fonctions ci-dessous retourneront le même résultat.

```
const fonction1 = () => {
  return 2;
};

function fonction2() {
  return 2;
};
```

- (•) Vrai
- ( ) Faux

  `fonction2` est écrite en ES5 et `fonction1` en ES6+. Habituez-vous à utiliser la syntaxe ES6.

Exercice

Que se passera-t-il si on exécute ce code ?

```
() => {
  console.log(2);
}
```

- ( ) La console retournera 2
- (•) Rien, on ne peut pas appeler cette fonction
- ( ) Le navigateur renverra une erreur, le code ne pourra pas s'exécuter
- ( ) Le code fera une boucle infinie qui appelle la fonction
- Le code ne renverra rien et cette fonction sera inutilisable, car on ne pourra pas l'initialiser.

Exercice

Quel chiffre apparaîtra en premier dans la console ?

```
(() => {
  let counter = 0
  for (let i = 0; i < 1000001; i++) {
    counter = i
  }
  console.log(counter)
})()

console.log(2)
```

- (•) 1000000
- ( ) 2

  JavaScript lit les informations de haut en bas. Sauf cas particulier, il va résoudre les étapes les unes à la suite des autres. On dit que son fonctionnement est synchrone.

  Dans le cas présent, même si la résolution de la boucle prend plus de temps que l'affichage d'un simple chiffre, il attendra d'avoir résolu la fonction avant de passer à l'étape suivante.

  Attention ! Un code JavaScript peut être rendu très facilement asynchrone, et vous serez souvent confronté à la problématique de la gestion de la synchronicité.

Exercice

Comment écrit-on la fonction suivante sous forme d'une constante en ES6 ?

```
function empty(){}
```

const empty= () => {}

p. 15 **Solution n°7**

```
createPage = () => {

  createBlocs = (blocs) => {
    const numberBlocks = Math.floor(Math.random() * 5) + 1
    for (let i = 0; i < numberBlocks; i++) {
      blocs.push({id: compteur, blocs: []})
      compteur++;
    }
  }

  const page = {blocs: []}
  let compteur = 1;
  createBlocs(page.blocs)
  page.blocs.forEach(bloc => {
    createBlocs(bloc.blocs)
    bloc.blocs.forEach(bloc => {
      createBlocs(bloc.blocs)
      bloc.blocs.forEach(bloc => {
        createBlocs(bloc.blocs)
      })
    })
  })
  return page
}


getBlock = (bloc, blocs) => {
  if (bloc.blocs.length > 0) {
    bloc.blocs.forEach(b => {
      blocs.push(b.id);
      getBlock(b, blocs);
    })
  }
}

const page = createPage(); // création d'un objet page aléatoire
console.log(page);

const idblocs = []; // tableau de stockage des id de chaque bloc
// On boucle sur le tableau de blocs contenu dans page
page.blocs.forEach(bloc => {
  idblocs.push(bloc.id); // stock les id des blocs
  getBlock(bloc, idblocs) // appel de la fonction récursives
})

console.log(idblocs);
```