

# Les fonctions en JS

# Table des matières

<b>I. Contexte</b>	<b>3</b>
<b>II. Utilisation et définition de fonctions</b>	<b>3</b>
A. Utilisation et définition de fonctions .....	3
B. Exercice : Quiz .....	5
<b>III. Paramètres, valeur de retour et expressions de fonctions</b>	<b>6</b>
A. Paramètres, valeur de retour et expressions de fonctions .....	6
B. Exercice : Quiz .....	10
<b>IV. Essentiel</b>	<b>11</b>
<b>V. Auto-évaluation</b>	<b>12</b>
A. Exercice .....	12
B. Test .....	12
<b>Solutions des exercices</b>	<b>13</b>

## I. Contexte

### Contexte

Les fonctions sont des éléments fondamentaux en programmation. Ce sont des blocs de codes contenant une ou plusieurs instructions qui remplissent un rôle dans le programme. Une fonction permet de réaliser une tâche précise. Les fonctions vont permettre d'organiser le code d'une manière plus cohérente et de réduire les répétitions inutiles d'instructions identiques.

Il est possible d'appeler des fonctions prédéfinies en JavaScript, c'est-à-dire des blocs de codes déjà écrits que l'on peut exécuter sans avoir à les écrire. Mais, s'il est possible d'appeler des fonctions prédéfinies accessibles par défaut ou importables via des bibliothèques, il est aussi possible de construire ses propres fonctions, des blocs de codes personnalisés que l'on pourra appeler autant de fois que nécessaire.

Les fonctions en programmation peuvent être assimilées aux fonctions mathématiques. En effet, nous constaterons que lorsqu'on parle de valeur de retour, de paramètres et d'arguments, le fonctionnement est très similaire. Nous verrons comment définir une fonction et l'utiliser à travers de nombreux exemples de codes.

Dans un premier lieu, nous parlerons de la définition basique de fonctions après avoir abordé l'utilisation de fonctions prédéfinies. Puis nous verrons comment passer des paramètres et définir une valeur de retour dans une fonction. N'hésitez pas à tester les nombreux exemples de code depuis votre IDE et à les modifier pour tester le potentiel des fonctions avec JS.

## II. Utilisation et définition de fonctions

### A. Utilisation et définition de fonctions

#### Fonctions prédéfinies

En JavaScript et en général dans tous les langages de programmation, il existe des fonctions prédéfinies. Il existe plusieurs fonctions natives, que l'on peut appeler n'importe où dans un code JS. Par exemple la fonction **eval()**. Elle permet d'évaluer un script JavaScript contenu dans une chaîne de caractères. Par exemple, si nous écrivons un script dans une chaîne :

```
1 "const nombre = 15; console.log(nombre * 2) ;"
```

Nous pouvons appeler la fonction **eval()** en indiquant dans les parenthèses la chaîne :

```
1 eval("const nombre = 15; console.log(nombre * 2);")
```

Nous pouvons voir qu'avec cette fonction, le code inscrit dans la chaîne est évalué et donc exécuté. La console affiche 30. C'est un exemple tout simple de fonction native. On trouve une liste des fonctions natives sur la documentation Mozilla à l'adresse : [developer.mozilla.org](https://developer.mozilla.org)<sup>1</sup> dans la partie « *Fonctions Prédéfinies* ». Mais intéressons-nous à la définition de « *fonctions personnalisées* ».

#### Syntaxe de la définition de fonctions

Pour définir des fonctions en JavaScript, il faut respecter cette syntaxe :

```
1 function nomDeLaFonction(paramètres) {  
2   instructions;  
3 }
```

On utilise le mot clé **function** pour définir une fonction. Les paramètres (dans le cas où il y en a) sont déclarés dans les parenthèses, et les instructions sont définies dans les accolades. Mais comment appeler notre fonction, afin d'exécuter les instructions qu'elle contient ?

---

1 [https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide/Functions#fonctions\\_pr%C3%A9d%C3%A9finies](https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide/Functions#fonctions_pr%C3%A9d%C3%A9finies)

## Syntaxe de l'appel d'une fonction

Pour appeler une fonction en JavaScript, on peut utiliser la syntaxe :

```
1 nomDeLaFonction (paramètres);
```

Prenons un exemple simple.

### Exemple Première définition d'une fonction

Nous souhaitons créer une fonction qui calcule le double d'un nombre et qui l'affiche dans une chaîne de caractères. Nous définissons une fonction que nous appellerons **calcDouble()** :

```
1 const nombre = 100;
2
3 function calcDouble() {
4   let double = nombre * 2;
5   console.log("Le double du nombre " + nombre + " est " + double);
6 }
```

Nous définissons donc la fonction **calcDouble**. Mais si nous exécutons ce code, il ne se passe pas grand-chose. Il faut que nous appelions notre fonction pour que ses instructions soient exécutées :

```
1 const nombre = 100;
2
3 function calcDouble() {
4   let double = nombre * 2;
5   console.log("Le double du nombre " + nombre + " est " + double);
6 }
7
8 calcDouble();
```

Nous pouvons voir que ça fonctionne, la console affiche : *"Le double du nombre 100 est 200"*. Maintenant, admettons que nous changions la valeur de la variable **nombre** (nous changeons le **const** en **let**) et que nous la définissons sur 40 par exemple. Nous voulons à nouveau calculer le double de **nombre**. Mais est-ce que nous avons besoin de réécrire les instructions permettant de calculer le double de **nombre** ? Non, nous n'avons qu'à appeler à nouveau la fonction **calcDouble()** :

```
1 let nombre = 100;
2
3 function calcDouble() {
4   let double = nombre * 2;
5   console.log("Le double du nombre " + nombre + " est " + double);
6 }
7
8 calcDouble();
9
10 nombre = 40;
11
12 calcDouble();
```

Nous pouvons voir que ça fonctionne, la console affiche :

```
1 Le double du nombre 100 est 200
2 Le double du nombre 40 est 80
```

Nous pouvons donc appeler autant de fois que nécessaire la fonction **calcDouble()**, sans avoir à écrire à chaque fois les mêmes instructions.

Voilà, vous savez maintenant comment définir une fonction de base en JavaScript.

**B. Exercice : Quiz**

[solution n°1 p.15]

## Question 1

Voici un code :

```
1 const cote = 4;
2
3 function aireCarre () {
4   console.log (cote ** 2);
5 }
```

Que va afficher la console ?

- ☐ 16
- ☐ Une erreur
- ☐ Rien

## Question 2

Voici un code :

```
1 const cote = 4;
2
3 aireCarre () {
4   console.log (cote ** 2);
5 }
6
7 aireCarre () ;
```

Que va afficher la console ?

- ☐ 16
- ☐ Une erreur
- ☐ Rien

## Question 3

Voici un code :

```
1 const marque = "Apple";
2
3 function verif() {
4   switch (marque) {
5     case "Apple":
6       console.log("C'est un Iphone");
7       break;
8     case "Samsung":
9       console.log("C'est un Android");
10      break;
11     case "Redmi":
12       console.log("C'est aussi un Android");
13       break;
14     default:
15       console.log("Marque non référencée");
16       break;
17   }
18 }
19
```

```
20 verif() ;
```

Que va afficher la console ?

- ☐ « C'est un Iphone »
- ☐ Une erreur
- ☐ Rien

#### Question 4

Voici un code :

```
1 const rayon = 4;
2
3 function aireSphere {
4   let aire = 4 * Math.PI * (rayon ** 2);
5   console.log (aire);
6 }
7
8 aireSphere();
```

Que va afficher la console ?

- ☐ Un nombre
- ☐ Une erreur
- ☐ Rien

#### Question 5

Voici un code :

```
1 const rayon = 4;
2
3 function aireSphere() {
4   let aire = 4 * Math.PI * (rayon ** 2);
5   console.log (aire);
6 }
7
8 aireSphere();
```

Que va afficher la console ?

- ☐ Un nombre
- ☐ Une erreur
- ☐ Rien

### III. Paramètres, valeur de retour et expressions de fonctions

#### A. Paramètres, valeur de retour et expressions de fonctions

Maintenant que nous savons définir une fonction de base, voyons comment passer des paramètres et quel en est l'intérêt.

## Paramètres

Qu'est-ce qu'un paramètre ? C'est un nom, on peut dire une variable qui est indiquée dans les parenthèses de la fonction. L'intérêt d'un paramètre est qu'il permet d'importer un argument dans une fonction, c'est-à-dire une valeur précise qui sera définie lors de l'appel de la fonction. Pour faire simple, lorsqu'on a un paramètre d'une fonction, on va pouvoir définir sa valeur lors de l'appel de la fonction. On pourra donc définir une valeur différente à chaque appel. Reprenons l'exemple que nous avons pris précédemment :

```
1 let nombre = 100;
2
3 function calcDouble() {
4   let double = nombre * 2;
5   console.log("Le double du nombre " + nombre + " est " + double);
6 }
7
8 calcDouble();
9
10 nombre = 40;
11
12 calcDouble();
```

Admettons que dans cet exemple, on veuille calculer le double d'un nombre passé en paramètre. Ça nous évitera d'avoir à déclarer hors de la fonction une variable **nombre** et d'avoir à changer sa valeur à chaque fois. Donc si on enlève cette variable, on obtient :

```
1 function calcDouble(?) {
2   let double = ? * 2;
3   console.log("Le double du nombre " + ? + " est " + double);
4 }
5
6 calcDouble(?);
7
8 calcDouble(?);
```

On a placé des points d'interrogation pour indiquer les emplacements où il manque une valeur. Comment faire pour initialiser un paramètre qui stockera la valeur d'un nombre ? Prenons un exemple mathématique :

$$f(x) = x * 2$$

Ici, nous avons une fonction mathématique. On peut voir que  $x$  est un paramètre de la fonction. Si on calcule  $f(4)$ , alors tous les  $x$  prennent la valeur 4 :

$$f(4) = 4 * 2$$

C'est le même principe pour les fonctions en programmation. On va initialiser un paramètre qu'on va par exemple appeler **nb**. On pourrait l'appeler **x** ou **a**, bref, lui donner n'importe quel nom. Puis lors de l'appel, on spécifie l'argument, c'est-à-dire la valeur que va prendre le paramètre **nb** :

```
1 function calcDouble(nb) {
2   let double = nb * 2;
3   console.log("Le double du nombre " + nb + " est " + double);
4 }
5
6 calcDouble(100); //Le double du nombre 100 est 200
7
8 calcDouble(40); //Le double du nombre 40 est 80
```

Nous pouvons voir que ça fonctionne et que nous pouvons appeler la fonction un nombre de fois illimité en passant en paramètre le nombre pour lequel le double sera calculé. Lorsque nous définissons une fonction, nous pouvons passer plusieurs paramètres. Par exemple, définissons une fonction permettant de calculer le périmètre d'un rectangle :

```
1 function perimetreRectangle(largeur, longueur) {
2   let perimetre = 2 * largeur + 2 * longueur;
3   console.log("Le perimetre du rectangle est de " + perimetre + " cm2");
4 }
```

Quand nous appelons notre fonction, il suffit de spécifier les arguments, donc les valeurs correspondant à la largeur et à la longueur, dans l'ordre correspondant au passage des paramètres :

```
1 function perimetreRectangle(largeur, longueur) {
2   let perimetre = 2 * largeur + 2 * longueur;
3   console.log("Le perimetre du rectangle est de " + perimetre + " cm2");
4 }
5
6 perimetreRectangle(4, 5);
7
8 perimetreRectangle(2, 4);
9
10 perimetreRectangle(1, 2);
11
12 perimetreRectangle(18, 20);
```

Nous pouvons voir que les différents appels de la fonction permettent d'afficher le périmètre d'un rectangle de largeur 4 et longueur 5, de largeur 2 et longueur 4, etc.

## Valeur de retour

Qu'est-ce qu'une valeur de retour ? Vous vous en rappelez, tout à l'heure nous avons pris un exemple d'une fonction mathématique :

$$f(x) = x * 2$$

Dans ce cas, il y a un paramètre spécifié qui est x. Mais à quoi peut correspondre la valeur de retour ? On peut dire que la valeur de retour de cette fonction est f(x). Dans le cas où x = 4, on a :

$$f(4) = 4 * 2 = 8$$

Dans le cas où x = 4, la valeur de retour de la fonction sera donc 8.

La valeur de retour de la fonction est donc x \* 2.

Jusque-là, nous n'avons pas défini de valeur de retour. Mais on pourrait se demander : est-ce que les fonctions que nous avons définies ont une valeur de retour ? La valeur de retour d'une fonction est renvoyée par l'expression permettant d'appeler une fonction. Par exemple, la valeur de retour de **perimetreRectangle(4, 5)** est renvoyée par l'expression **perimetreRectangle(4, 5)**.

Voyons donc si notre fonction a une valeur de retour en faisant un **console.log** de l'expression **perimetreRectangle(4, 5)**.

```
1 function perimetreRectangle(largeur, longueur) {
2   let perimetre = 2 * largeur + 2 * longueur;
3   console.log("Le perimetre du rectangle est de " + perimetre + " cm2");
4 }
5
6 console.log (perimetreRectangle(4, 5));
```

On peut voir que la fonction est appelée, mais qu'elle renvoie (ou « que sa valeur de retour est ») **undefined**. Notre fonction n'a donc pas de valeur de retour explicite, c'est une fonction que l'on peut qualifier « de procédure ». Mais imaginons qu'on veuille que la valeur de retour de notre fonction soit justement le périmètre de notre rectangle. Ce



serait plus logique puisque c'est la valeur que l'on cherche à obtenir avec la fonction **perimetreRectangle()**. On peut (bien que ce ne soit pas obligatoire) retirer le **console.log** affichant la chaîne de caractères avec le périmètre. Pour définir la valeur de retour d'une fonction, on utilise le mot clé **return** :

```
1 function perimetreRectangle(largeur, longueur) {
2   let perimetre = 2 * largeur + 2 * longueur;
3   return perimetre;
4 }
5
6 console.log (perimetreRectangle(4, 5));
```

Dans ce dernier exemple, la fonction **perimetreRectangle()** a pour valeur de retour la valeur de la variable **perimetre**.

L'expression **perimetreRectangle(4, 5)** va donc renvoyer  $2 * 4 + 2 * 5$ , donc 18. Comme on affiche **perimetreRectangle(4, 5)** dans la console, la console affiche 18. On peut tester la fonction avec de nombreux arguments comme suit :

```
1 function perimetreRectangle(largeur, longueur) {
2   let perimetre = 2 * largeur + 2 * longueur;
3   return perimetre;
4 }
5
6 console.log (perimetreRectangle(4, 5)); //18
7
8 console.log (perimetreRectangle(8, 5)); //26
9
10 console.log (perimetreRectangle(4.4, 1.67)); //12.14
11
12 console.log (perimetreRectangle(64, 45)); //218
```

### Fondamental Utilisation du mot clé return

Le mot clé **return** met fin à l'exécution de la fonction et définit la valeur de retour.

Voilà, vous savez maintenant utiliser le mot clé **return** pour renvoyer une valeur. À noter qu'en règle générale, il est préconisé de définir dans une fonction, une valeur de retour. Tous les exemples que nous avons pu voir dans la première partie ne sont donc pas forcément préconisés. Maintenant que vous savez définir une valeur de retour, prenez donc l'habitude d'utiliser le mot clé **return** dans vos fonctions.

### Complément Créer une fonction grâce à une expression de fonction

Dans tous les exemples de ce cours, nous avons déclaré des fonctions en utilisant une instruction (d'un point de vue syntaxique). Mais il est aussi possible de créer une fonction en passant par une expression. Une expression est une unité de code qui renvoie une valeur. Par exemple, l'expression suivante renvoie 4.

```
1 2 * 2
```

On peut affecter la valeur renvoyée par cette expression à une variable par exemple :

```
1 let a = 2 * 2 ;
```

Mais comment créer une fonction via une expression ? Voici un exemple tout simple :

```
1 let racine = function racineCarree (nombre) {return Math.sqrt(nombre)};
```

Nous pouvons voir que nous affectons à la variable **racine** l'expression de notre fonction **racineCarree()**. Cette fonction a simplement pour objectif de renvoyer la racine du nombre passé en paramètre, grâce au **Math.sqrt()**. Dans ce cas, nous créons la fonction grâce à une expression de fonction. Mais nous pouvons aussi grâce à ce système créer une fonction anonyme, c'est-à-dire une fonction qui n'a pas de nom :

```
1 let racine = function (nombre) {return Math.sqrt(nombre)};
```

Dans les deux cas, nous pouvons appeler notre fonction en passant par la variable **racine**. Par exemple :

```
1 console.log(racine(9));
```

La console affiche donc 3.

Voilà, vous savez maintenant déclarer une fonction en passant des paramètres, en retournant une valeur et en utilisant une expression de fonction.

## B. Exercice : Quiz

[solution n°2 p.17]

### Question 1

Voici un code :

```
1 const cote = 2;
2
3 function perimetreCarre(c) {
4   return c * 4;
5 }
6
7 perimetreCarre(cote);
```

Que va afficher la console ?

- ☐ 8
- ☐ Une erreur
- ☐ Rien

### Question 2

Voici un code :

```
1 const cote = 2;
2
3 function perimetreCarre(c) {
4   c * 4;
5 }
6
7 console.log(perimetreCarre(cote));
```

Que va afficher la console ?

- ☐ 8
- ☐ Une erreur
- ☐ Undefined

### Question 3

Voici un code :

```
1 function calcHypotenuse(a , b) {
2   let hypotenuse = Math.sqrt((a ** 2) + (b ** 2));
3   return hypotenuse;
4 }
5
6 console.log("Hypotenuse : " + calcHypotenuse(10, 11));
```

Que va afficher la console ?

- ☐ "Hypotenuse : " + un nombre
- ☐ Une erreur
- ☐ Rien

#### Question 4

Voici un code différent :

```
1 let maFonction = function (a , b) {  
2   let hypotenuse = Math.sqrt((a ** 2) + (b ** 2));  
3   return hypotenuse;  
4 }  
5  
6 console.log("Hypotenuse : " + maFonction(10, 11));
```

Que va afficher la console ?

- ☐ "Hypotenuse : " + un nombre
- ☐ Une erreur
- ☐ Rien

#### Question 5

Voici un code différent :

```
1 let maFonction = function calcHypotenuse (a , b) {  
2   let hypotenuse = Math.sqrt((a ** 2) + (b ** 2));  
3   return hypotenuse;  
4 }  
5  
6 console.log("Hypotenuse : " + maFonction(10, 11));
```

Le code est-il correct ?

- ☐ Oui
- ☐ Non

## IV. Essentiel

Dans ce cours, nous avons parlé des fonctions en JavaScript. Les fonctions sont des blocs de codes, des ensembles d'instructions que l'on peut appeler. Il existe des fonctions prédéfinies, et il arrive que des librairies ou modules externes fournissent des fonctions toutes préparées.

Il est possible de définir ses propres fonctions, c'est-à-dire des blocs de codes que nous écrivons nous-mêmes et que nous pouvons appeler dans le code, autant de fois que nécessaire. Un avantage considérable des fonctions est qu'elles permettent de réduire les répétitions inutiles d'instructions identiques. Par ailleurs, elles permettent de mieux organiser son code et de centraliser les modifications.

Lorsqu'on définit une fonction en JavaScript, on utilise le mot clé **function**. Puis on donne un nom à notre fonction, on insère des parenthèses permettant de stocker les paramètres et on peut ouvrir des accolades pour y insérer les instructions de la fonction. Le mot clé **return** permet de définir la valeur de retour d'une fonction. Pour appeler une fonction, il suffit d'écrire son nom suivi de parenthèses contenant les arguments passés (dans le cas où des arguments sont attendus).

Nous avons vu qu'il est possible de créer une fonction par le biais d'expressions de fonctions. Par ailleurs, ce système peut permettre de créer des fonctions anonymes.

## V. Auto-évaluation

### A. Exercice

Vous tenez un magasin de coques pour téléphones. Vous cherchez à automatiser certaines tâches concernant la gestion de votre stock et de vos ventes. Voici votre script de base :

```
1 let stock = 4029;
2
3 let solde = 0;
4
5 //code
```

#### Question 1

[solution n°3 p.19]

Vous cherchez à créer une première fonction vous permettant de vendre un certain nombre d'articles, en spécifiant le prix unitaire des articles. Cette fonction permettra de retirer du stock le nombre d'articles vendus, et d'ajouter au solde le total des gains (nombre de produits multiplié par le prix unitaire). L'objectif est de passer en paramètres de la fonction le nombre d'articles vendus, et le prix unitaire. La fonction pourra opérer ces différentes instructions et finalement retourner le total des gains des ventes. Une fois la fonction définie, vous pouvez l'appeler afin de vendre 104 produits avec un prix unitaire de 10,50. Vous pouvez afficher dans la console la valeur de retour de cette expression, puis le stock et enfin le solde.

#### Question 2

[solution n°4 p.19]

Écrire une seconde fonction cette fois-ci permettant d'afficher dans une chaîne le stock et le solde du magasin. Cette chaîne pourra ressembler à :

"Stock du magasin : {stock} produits

Solde total : {solde} euros"

La fonction devra retourner les informations du magasin (stock et solde) dans une chaîne de caractère.

### B. Test

#### Exercice 1 : Quiz

[solution n°5 p.19]

##### Question 1

Quelle est la syntaxe correcte en JS pour définir une fonction ?

- ☐ function nomDeLaFonction (paramètres) {instructions;}
- ☐ def nomDeLaFonction (paramètres): instructions
- ☐ nomDeLaFonction (paramètres) {instructions;}

##### Question 2

Quelle est la syntaxe correcte pour appeler une fonction en JS ?

- ☐ nomDeLaFonction(paramètres);
- ☐ nomDeLaFonction paramètres;
- ☐ nomDeLaFonction;

##### Question 3

Quel mot clé permet de définir la valeur de retour d'une fonction en JS ?

- ☐ return
- ☐ print
- ☐ break

#### Question 4

Voici un code :

```
1 function carre (a) {  
2   return a ** 2  
3   console.log ("Le carré de " + a + " est " + a ** 2)  
4 }  
5  
6 carre (4);
```

Que va afficher la console ?

- ☐ Rien
- ☐ 16
- ☐ "Le carré de 4 est 16"

#### Question 5

Parmi les exemples de code suivants, lequel est correct pour définir une fonction anonyme ?

- ☐ let variable = function (a) {return a};
- ☐ let variable = (a) {return a};
- ☐ let variable = function {return a};

## Solutions des exercices



## Exercice p. 5 Solution n°1

## Question 1

Voici un code :

```
1 const cote = 4;
2
3 function aireCarre () {
4   console.log (cote ** 2);
5 }
```

Que va afficher la console ?

- ☐ 16
- ☐ Une erreur
- ☒ Rien



La fonction **aireCarre()** est définie avec aucune erreur de syntaxe. Cependant, elle n'est jamais appelée. Donc la console n'affichera rien.

## Question 2

Voici un code :

```
1 const cote = 4;
2
3 aireCarre () {
4   console.log (cote ** 2);
5 }
6
7 aireCarre () ;
```

Que va afficher la console ?

- ☐ 16
- ☒ Une erreur
- ☐ Rien



Le mot clé **function** manque pour déclarer la fonction **aireCarree()**. La console va donc afficher une erreur.

## Question 3

Voici un code :

```
1 const marque = "Apple";
2
3 function verif() {
4   switch (marque) {
5     case "Apple":
6       console.log("C'est un Iphone");
7       break;
8     case "Samsung":
9       console.log("C'est un Android");
10      break;
11     case "Redmi":
12       console.log("C'est aussi un Android");
```


```

13     break;
14     default:
15         console.log("Marque non référencée");
16         break;
17     }
18 }
19
20 verif() ;

```

Que va afficher la console ?

- ☒ « C'est un Iphone »
- ☐ Une erreur
- ☐ Rien

 Dans cet exemple, la fonction **verif()** est déclarée et définie sans erreur de syntaxe. Elle contient un **switch** qui est aussi défini sans erreur de syntaxe et qui affiche dans la console "*C'est un Iphone*" dans le cas où **marque** est égale à "*Apple*". La fonction est bien appelée à la fin du code et donc exécutée.

#### Question 4

Voici un code :


```

1 const rayon = 4;
2
3 function aireSphere {
4     let aire = 4 * Math.PI * (rayon ** 2);
5     console.log (aire);
6 }
7
8 aireSphere();

```

Que va afficher la console ?

- ☐ Un nombre
- ☒ Une erreur
- ☐ Rien

 Pour définir la fonction **aireSphere()**, il ne faut pas oublier les parenthèses qui ne sont pas indiquées dans sa déclaration. La console affiche donc une erreur.

#### Question 5

Voici un code :


```

1 const rayon = 4;
2
3 function aireSphere() {
4     let aire = 4 * Math.PI * (rayon ** 2);
5     console.log (aire);
6 }
7
8 aireSphere();

```

Que va afficher la console ?




- ☒ Un nombre
- ☐ Une erreur
- ☐ Rien
-  Dans ce cas, la fonction **aireSphere()** est définie sans erreur de syntaxe. Elle est correctement appelée et va donc afficher un nombre correspondant à l'aire d'une sphère de rayon 4.

**Exercice p. 10 Solution n°2****Question 1**

Voici un code :

```
1 const cote = 2;
2
3 function perimetreCarre(c) {
4   return c * 4;
5 }
6
7 perimetreCarre(cote);
```

Que va afficher la console ?


- ☐ 8
- ☐ Une erreur
- ☒ Rien
-  La fonction **perimetreCarre()** est définie et appelée avec aucune erreur de syntaxe. Un argument doit être passé, car le paramètre **c** est spécifié. On passe **cote**, comme argument lors de l'appel, qui a pour valeur 2. Dans la définition de la fonction, on définit la valeur de retour sur **c \* 4**. Donc l'expression **perimetreCarre(cote)** va renvoyer 8. Mais il n'y a aucun **console.log** permettant d'afficher la valeur renvoyée par **perimetreCarre(cote)**. Donc la console n'affiche rien.

**Question 2**

Voici un code :

```
1 const cote = 2;
2
3 function perimetreCarre(c) {
4   c * 4;
5 }
6
7 console.log(perimetreCarre(cote));
```

Que va afficher la console ?

- ☐ 8
- ☐ Une erreur
- ☒ Undefined
-  Dans ce cas, l'instruction **return** manque, la valeur de retour de la fonction n'est donc pas définie. L'expression **perimetreCarre(cote)** renvoie donc **undefined**, qui est affichée dans la console.


### Question 3

Voici un code :

```
1 function calcHypotenuse(a , b) {
2   let hypotenuse = Math.sqrt((a ** 2) + (b ** 2));
3   return hypotenuse;
4 }
5
6 console.log("Hypotenuse : " + calcHypotenuse(10, 11));
```

Que va afficher la console ?

- ☒ "Hypotenuse : " + un nombre
- ☐ Une erreur
- ☐ Rien

 La fonction **calcHypotenuse()** est correctement définie. Elle permet de calculer l'hypoténuse d'un triangle rectangle à partir des côtés **a** et **b** et en utilisant le théorème de Pythagore, et de retourner sa valeur. Elle est appelée en passant les arguments 10 (pour **a**) et 11 (pour **b**). Dans la console, on affiche une chaîne contenant "Hypotenuse : " et la valeur renvoyée par **calcHypotenuse(10, 11)** qui est un nombre correspondant à l'hypoténuse.


### Question 4

Voici un code différent :

```
1 let maFonction = function (a , b) {
2   let hypotenuse = Math.sqrt((a ** 2) + (b ** 2));
3   return hypotenuse;
4 }
5
6 console.log("Hypotenuse : " + maFonction(10, 11));
```

Que va afficher la console ?

- ☒ "Hypotenuse : " + un nombre
- ☐ Une erreur
- ☐ Rien

 Dans cet exemple, nous créons une fonction anonyme en utilisant une expression de fonction. Il n'y a aucune erreur de syntaxe, et on peut appeler la fonction en passant par la variable **maFonction** qui est définie sur cette expression de fonction.

### Question 5

Voici un code différent :

```
1 let maFonction = function calcHypotenuse (a , b) {
2   let hypotenuse = Math.sqrt((a ** 2) + (b ** 2));
3   return hypotenuse;
4 }
5
6 console.log("Hypotenuse : " + maFonction(10, 11));
```

Le code est-il correct ?

- ☒ Oui
- ☐ Non

Q Le code est correct, car il n'y a aucune erreur de syntaxe. On définit la fonction via une expression de fonction. On peut très bien lui donner un nom (comme dans cet exemple), ou bien créer une fonction anonyme (comme dans l'exemple précédent).

#### p. 12 Solution n°3

Voici un code qui fonctionne :

```
1 let stock = 4029;
2
3 let solde = 0;
4
5 function vendre(nbProduits, priceUnitaire) {
6   stock -= nbProduits;
7   let total = (nbProduits * priceUnitaire);
8   solde += total;
9   return total;
10 }
11
12 console.log(vendre(104, 10.50)); //1092
13
14 console.log(stock); //3925
15
16 console.log(solde); //1092
```

#### p. 12 Solution n°4

Voici un code qui fonctionne :


```
1 let stock = 4029;
2
3 let solde = 0;
4
5 function vendre(nbProduits, priceUnitaire) {
6   stock -= nbProduits;
7   let total = (nbProduits * priceUnitaire);
8   solde += total;
9   return total;
10 }
11
12 function infosMagasin() {
13   let message = "Stock du magasin : " + stock + " produits\nSolde total : " + solde + "
14   euros";
15   console.log (message);
16   return message;
17 }
18 console.log(vendre(104, 10.50)); //1092
19
20 console.log(stock); //3925
21
22 console.log(solde); //1092
23
24 infosMagasin();
```

#### Exercice p. 12 Solution n°5

### Question 1

Quelle est la syntaxe correcte en JS pour définir une fonction ?


- ☒ `function nomDeLaFonction (paramètres) {instructions;}`
- ☐ `def nomDeLaFonction (paramètres): instructions`
- ☐ `nomDeLaFonction (paramètres) {instructions;}`

 La syntaxe correcte pour définir une fonction en JS est **function nomDeLaFonction (paramètres) {instructions;}**.

### Question 2

Quelle est la syntaxe correcte pour appeler une fonction en JS ?


- ☒ `nomDeLaFonction(paramètres);`
- ☐ `nomDeLaFonction paramètres;`
- ☐ `nomDeLaFonction;`

 La syntaxe correcte pour appeler une fonction en JS est **nomDeLaFonction(paramètres);**.

### Question 3

Quel mot clé permet de définir la valeur de retour d'une fonction en JS ?

- ☒ `return`
- ☐ `print`
- ☐ `break`

 Le mot clé **return** permet de définir la valeur de retour d'une fonction. Il met fin à la fonction.


### Question 4

Voici un code :

```
1 function carre (a) {
2   return a ** 2
3   console.log ("Le carré de " + a + " est " + a ** 2)
4 }
5
6 carre (4);
```

Que va afficher la console ?

- ☒ Rien
- ☐ 16
- ☐ "Le carré de 4 est 16"

 Dans cet exemple, nous définissons la fonction **carre()** en définissant la valeur de retour avant d'afficher la chaîne de caractères dans la console. L'instruction **return** définit la valeur de retour de la fonction et met fin à la fonction. Donc le **console.log** n'est pas exécuté. Lorsqu'on appelle la fonction, **carre (4)** renvoie 16, mais cette valeur n'est pas affichée dans la console, donc rien n'est affiché dans la console.

**Question 5**


---

Parmi les exemples de code suivants, lequel est correct pour définir une fonction anonyme ?

☒ `let variable = function (a) {return a};`

☐ `let variable = (a) {return a};`

☐ `let variable = function {return a};`

 Seul l'exemple de code **let variable = function (a) {return a};** est correct. On définit la variable **variable** sur l'expression de fonction **function (a) {return a}**. Il est nécessaire d'utiliser le mot clé **function** ainsi que les parenthèses.