Les fonctions en Dart

Table des matières

I. Présentation générale des fonctions en Dart

Contexte

Maintenant que vous savez créer une fonction `main`, le point d'entrée d'exécution de votre programme, il est temps d'y ajouter vos propres fonctions !

Il arrive très souvent que, dans un programme, nous soyons amenés à répéter le même type de tâche dans différents endroits de notre code.

Est-ce que cela signifie qu'il faut écrire plusieurs fois exactement les mêmes instructions ? Heureusement, non ! Les fonctions sont là pour vous faciliter la vie et rendre votre code plus lisible.

Ensemble, nous allons voir les notions de base concernant les fonctions dans Flutter, avant d'en étudier les différents paramètres.

Définition **Fonction**

Une fonction (ou une méthode) est un ensemble de codes qui exécute une tâche spécifique.

En Dart, nous pouvons distinguer les fonctions intégrées à la bibliothèque standard et celles définies par l'utilisateur.

Exemple

`print()` est une fonction intégrée, tandis que **`mafonction()`** est définie par nous ci-dessous :

```
1 void mafonction() {
2   print('Ceci est ma fonction');
3 }
```

Pour appeler une fonction depuis une autre fonction, il suffit alors d'écrire :

```
nom_de_la_fonction(paramètres);
```

Conseil

L'utilisation de fonctions, et donc d'un ensemble d'instructions répondant à une tâche spécifique, permet de mieux organiser son code. En créant des fonctions, vous pourrez les réutiliser plusieurs fois dans votre programme sans avoir à réécrire plusieurs fois les mêmes lignes !

Fondamental **Les propriétés des fonctions**

Une fonction a deux propriétés principales :

- Les **arguments** (aussi appelés paramètres) : ce sont des informations transmises à la fonction qui seront utilisées dans l'exécution du code de celle-ci.
- Le **résultat** : une fonction renvoie (presque toujours) un résultat issu de l'exécution du code.

Exemple

```
1 int mafonction(int x) {
2   return x;
3 }
```

Ici, nous passons le paramètre `x`, qui sera ensuite renvoyé comme le résultat de la fonction.

Attention **La syntaxe des fonctions**

Les fonctions ont une syntaxe bien précise. Tout d'abord, il ne faut pas oublier de **créer un nom** à la fonction, afin que nous puissions l'appeler plus tard.

Il y a un nom de fonction qu'on ne doit utiliser qu'une fois, et c'est **le nom `main`.** La fonction `main` correspond au point d'entrée d'une application Dart. En d'autres termes, il s'agit de la toute première fonction qui sera exécutée.

Méthode

Il faut également renseigner les paramètres requis à l'exécution de notre fonction (une fonction n'est pas obligée d'avoir des paramètres).

Le type de retour est requis, il permet de comprendre le type de la réponse qui est attendue en résultat (un `int` ou une `String` par exemple).

Enfin, il faut rajouter un ensemble d'instructions dans le corps de la fonction.

La syntaxe est comme suit :

```
type_de_resultat nom_de_fonction(type_de_paramètre nom_de_paramètre) {
instructions;
}
```

(La définition de type de paramètre n'est cependant pas obligatoire).

Exemple

```
1 int mafonction(int x) {
2   int y;
3   y = x * 4;
4   return y;
5 }
```

La fonction a pris l'`int « x »` en paramètre, y a appliqué ses instructions et a retourné l'`int « y »` en résultat.

Remarque

Si une fonction peut renvoyer des types de résultats de différentes sortes, ou alors ne pas renvoyer de résultat du tout, il est possible de remplacer le type par `void`.

Exemple

```
1 void main() {
2   print('Cette fonction ne renvoie pas de résultat !');
3 }
```

Fondamental **La syntaxe arrow**

Il existe une syntaxe simplifiée pour une fonction n'ayant qu'une seule instruction dans son corps. Ce type de fonction est l'équivalent d'une fonction `lambda` dans certains langages.

La syntaxe est comme suit :

```
1 type_de_retour nom_de_la_fonction(paramètre) => instruction
```

Exemple

Au lieu d'avoir :

```
void jaffiche(String mastring) {
  print(mastring);
}
```

Nous aurons :

```
void jaffiche(mastring) => print(mastring);
```

C'est plus rapide et souvent plus lisible !

Exercice : Quiz

[solution n°1 p.13]

Question 1

Une fonction est :

- ○ Un ensemble de tâches spécifiques qui exécutent un code
- ○ Un ensemble de codes exécutant une tâche spécifique
- ○ Un ensemble de tâches spécifiques exécutant un ensemble de code

Question 2

La fonction `print()` est :

- ○ Une fonction intégrée à la bibliothèque standard
- ○ Une fonction intégrée d'une bibliothèque spécifique
- ○ Une fonction créée par le programmeur

Question 3

Pourquoi dois-je créer des fonctions ?

- ☐ Cela rend mon code plus rapide
- ☐ Cela rend mon code plus lisible
- ☐ Cela m'évite d'écrire plusieurs fois la même suite d'instructions

Question 4

Quelles sont les deux propriétés principales d'une fonction ?

- ○ Le résultat et les paramètres
- ○ Le type et les paramètres
- ○ Le résultat et le type

Question 5

Lorsque je fais appel à une fonction, à quoi dois-je faire attention ?

- ☐ Aux types des paramètres
- ☐ Au type du résultat
- ☐ À l'ordre des paramètres

Question 6

Il est obligatoire d'indiquer un type pour un paramètre.

- Vrai
- Faux

Question 7

La syntaxe `arrow` permet :

- Une écriture simplifiée pour une fonction n'ayant qu'une seule instruction dans son corps
- Une écriture simplifiée pour une fonction ayant plusieurs autres appels de fonctions dans son corps
- Une écriture plus complexe offrant plus de possibilités

Question 8

```
1 int bonjour(String i) {
2   return i
3 }
```

Cette fonction peut être utilisée.

- Vrai
- Faux

Question 9

```
1 void aurevoir(int i) {
2   return i
3 }
```

Cette fonction peut être utilisée.

- Vrai
- Faux

Question 10

```
1 String jemapelle(String prenom) {
2   print(prenom);
3 }
```

Comment puis-je écrire cette fonction en syntaxe `arrow` ?

- `jemapelle(prenom) => print(prenom);`
- `String jemapelle(prenom) => print(prenom);`
- `void jemapelle(preynom) => print(prénom);`

III. Paramètres

Définition **Les paramètres obligatoires**

Les paramètres obligatoires sont des paramètres qui sont **essentiels à l'exécution d'une fonction**. Si un argument manque à l'appel, la fonction ne pourra pas et même refusera de se lancer !

Pour le moment, tous les paramètres que nous avons utilisés dans nos exemples sont des paramètres obligatoires.

Nous allons donc voir dans la suite de ce cours comment rendre des paramètres optionnels et à quoi cela peut servir.

Remarque

Il est important de faire attention à l'ordre des paramètres lorsque nous faisons appel à une fonction.

Exemple

```
void mafonction(String string1, String string2) {
  print(string1 + string 2)
}
```

Si nous appelons la fonction avec **`mafonction('le Dart', 'j'aime'),`** notre programme affichera `le Dartj'aime`. Il faut donc mettre les arguments dans l'ordre au moment de l'appel de la fonction, pour voir s'afficher `J'aime le Dart`.

Attention

Ceci peut être source d'erreurs, et c'est pourquoi il est nécessaire d'être vigilant. Si plusieurs paramètres sont de types différents, une des façons de se protéger de ce type de coquille est d'indiquer le type du paramètre. Ainsi, si vous inversez deux paramètres et qu'un `int` se retrouve à la place d'une `String`, Dart vous avertira qu'il y a une erreur. Une autre possibilité est de nommer ses paramètres, ce que nous allons voir un peu plus tard dans ce cours.

Méthode **Les paramètres optionnels de position**

Une première méthode pour rendre un paramètre optionnel consiste à la mettre entre crochets **`[ ].`**

Exemple

```
void main() {
  livre('Le livre de la Jungle', 568); //Ceci affichera « Je lis Le livre de la Jungle et il a 568 pages ! »
livre('Le livre de la Jungle'); // « Je lis Le livre de la Jungle. »
}

void livre(String livre, [nombredepages]) {
  if (nombredepages != null) {
    print("Je lis " +
        livre +
        " et il a " +
        nombredepages.toString() +
        " pages !");
  } else {
    print("Je lis " + livre + ".");
  }
}
```

Comme nous pouvons le voir, l'absence du second argument n'empêche pas le lancement de la fonction. Nous prenons cependant bien soin de vérifier si le second argument est `null` ou non avant d'essayer de l'utiliser.

Définition **Les paramètres optionnels nommés**

Les paramètres optionnels nommés permettent, comme leur nom l'indique, de nommer les paramètres d'une fonction. Cela a l'avantage de nous permettre de ne pas respecter l'ordre de ces paramètres sans risquer de faire d'erreur.

Méthode

Ces paramètres doivent être placés entre **{ }** dans la déclaration de la fonction (par exemple : `livre({nombredepages})`. Ensuite, dans l'appel de la fonction, chaque paramètre doit être nommé (par exemple : `livre(nombredepages : 568)`).

Exemple

Si nous reprenons l'exemple précédent, nous aurons alors :

```
void main() {
  livre(livre: 'Le livre de la Jungle', nombredepages: 568); //Ceci affichera « Je lis Le livre de la Jungle et il a 568 pages ! »
}

void livre({String livre, double nombredepages}) {
    print("Je lis " +
        livre +
        " et il a " +
        nombredepages.toString() +
        " pages !");
}

```

Tous les paramètres ne doivent pas nécessairement être nommés ou non, par exemple il est tout à fait possible d'avoir : `livre(String monavis, {String livre, double nombredepages})`. Ainsi, seuls les deux derniers paramètres sont des paramètres optionnels nommés.

Conseil **Les paramètres optionnels nommés avec valeur par défaut**

Il peut parfois être nécessaire de donner des valeurs par défaut à des paramètres.

Si nous prenons l'exemple de Flutter, un kit de développement logiciel en Dart permettant de créer des applications mobiles rapidement, de nombreuses fonctions ont des valeurs par défaut. Ainsi, si nous souhaitons simplement créer une barre d'application, nous pouvons le faire avec **`AppBar()`**. Nous aurons alors tous les paramètres par défaut, à savoir une couleur bleue par exemple. Si nous souhaitons changer cette couleur, il faudra alors appeler : `AppBar(backgroundColor: Colors.yellow)`. Le paramètre que nous venons de renseigner remplacera alors la valeur par défaut, et nous aurons une barre d'application jaune !

Pour assigner une valeur par défaut à des paramètres, rien de plus simple. Il suffit de le rajouter juste après la déclaration du paramètre.

Exemple

```
void main() {
  film(nomdufilm: 'Titanic'); // Affichera « Je regarde Titanic »
film(); // Affichera « Je regarde Matrix », Matrix étant la valeur par défaut
}

void film({String nomdufilm = 'Matrix'}) {
  print('Je regarde ' + nomdufilm);
}
```

Exercice : Quiz

[solution n°2 p.15]

Question 1

```
1 void mafonction([String nomdulivre]);
```

Le paramètre `nomdulivre` :

- O Est optionnel
- O Est optionnel nommé
- O Possède une valeur par défaut

Question 2

```
1 void mafonction({String nomdulivre});
```

Le paramètre `nomdulivre` :

- O Est optionnel
- O Est optionnel nommé
- O Possède une valeur par défaut

Question 3

```
1 void mafonction({String nomdulivre = ‘Shoe dog'});
```

Le paramètre `nomdulivre` :

- ☐ Est optionnel
- ☐ Est optionnel nommé
- ☐ Possède une valeur par défaut

Question 4

```
1 void mafonction(String nomdulivre);
```

Le paramètre *`nomdulivre`* :

- O Est optionnel
- O Est optionnel nommé
- O Possède une valeur par défaut

Question 5

Quel est l'intérêt de donner des valeurs par défaut à des paramètres ?

- O Cela permet de ne jamais avoir d'erreur
- O Cela permet d'avoir des fonctions qui marchent même si nous ne souhaitons pas nous prononcer sur certains paramètres
- O Cela permet une meilleure lisibilité

Question 6

Je peux avoir plusieurs types de paramètres dans une même fonction.

- ○ Vrai
- ○ Faux

Question 7

Si un paramètre est optionnel et ne possède pas de valeur par défaut, à quoi dois-je faire attention ?

- ○ Je dois vérifier avant l'utilisation de ce paramètre qu'il n'est pas `null`.
- ○ Je dois vérifier que le paramètre est bien du bon type.
- ○ Je dois vérifier que le type du paramètre correspond au type du résultat.

V. Essentiel

Les fonctions en Dart :

- Une fonction ou une méthode est un ensemble de codes qui exécute une tâche spécifique.
- Il faut distinguer les fonctions intégrées à la bibliothèque standard et celles définies par l'utilisateur.
- L'utilisation de fonctions, et donc d'un ensemble d'instructions répondant à une tâche spécifique, permet de mieux organiser son code.
- Les arguments (aussi appelés paramètres) sont des informations transmises à la fonction qui seront utilisées dans l'exécution du code de celle-ci.
- Une fonction renvoie (presque toujours) un résultat issu de l'exécution du code.
- Une fonction possède un corps, des paramètres, un résultat et un nom.
- Il existe une syntaxe simplifiée pour une fonction n'ayant qu'une seule instruction dans son corps. C'est la syntaxe `arrow`.

Les paramètres :

- Les paramètres obligatoires sont des paramètres qui sont essentiels à l'exécution d'une fonction.
- Il est important de faire attention à l'ordre des paramètres.
- Une première méthode pour rendre un paramètre optionnel consiste à la mettre entre crochets `[]`. C'est un paramètre optionnel de position.
- Les paramètres optionnels nommés permettent, comme leur nom l'indique, de nommer les paramètres d'une fonction. Ces paramètres doivent être placés entre `{}`.
- Il peut parfois être nécessaire de donner des valeurs par défaut à des paramètres. Pour cela, il suffit de rajouter la valeur juste après la déclaration du paramètre.

VI. Auto-évaluation

A. Exercice

Question 1

[solution n°3 p.17]

Créez une fonction prenant en paramètre une `String` « `chanson` ». Ce paramètre peut ne pas être renseigné, mais la fonction s'exécutera quand même sans erreur : elle affichera alors « `Je n'ai pas de chanson préférée` ». Si le paramètre est renseigné, ma fonction affichera « `Ma chanson préférée est` » suivi de mon paramètre.

Question 2

[solution n°4 p.17]

Créez une fonction avec la syntaxe `arrow`, qui affiche simplement son paramètre. Il peut être de tout type (en le convertissant en `String`).

Question 3

[solution n°5 p.17]

Créez une fonction prenant en paramètre un paramètre obligatoire et un paramètre optionnel nommé avec valeur par défaut. Le paramètre obligatoire sera le nom d'un film, et le paramètre optionnel sera la durée du film (sous forme de `String`, exemple : `1 h 24`). La valeur par défaut de ce paramètre sera `durée inconnue`.

Le programme devra afficher : `je regarde` + `nomdufilm` + `de` + durée.

Exemple : `Je regarde Titanic de durée 3 h`, `Je regarde Titanic de durée inconnnue`.

B. Test

Exercice 1 : Quiz

[solution n°6 p.17]

Question 1

Pourquoi dois-je créer des fonctions ?

- ☐ Cela rend mon code plus rapide
- ☐ Cela rend mon code plus lisible
- ☐ Cela m'évite d'écrire plusieurs fois la même suite d'instructions

Question 2

```
1 String aurevoir(int i) {
2   return i
3 }
```

Cette fonction peut être utilisée.

- ○ Vrai
- ○ Faux

Question 3

Quel est l'intérêt de donner des valeurs par défaut à des paramètres ?

- ○ Cela permet de ne jamais avoir d'erreur
- ○ Cela permet d'avoir des fonctions qui marchent même si nous ne souhaitons pas nous prononcer sur certains paramètres
- ○ Cela permet une meilleure lisibilité

Question 4

Je peux avoir plusieurs types de paramètres dans une même fonction.

- ○ Vrai
- ○ Faux

Question 5

Dans `void mafonction({String unnom = 'Isabelle'})`, quel est le type du paramètre ?

- ○ Optionnel nommé avec valeur par défaut
- ○ Optionnel nommé
- ○ Obligatoire
- ○ Optionnel de position

Solutions des exercices

Exercice p. 5 **Solution n°1**

Question 1

Une fonction est :

- ○ Un ensemble de tâches spécifiques qui exécutent un code
- ◉ Un ensemble de codes exécutant une tâche spécifique
- ○ Un ensemble de tâches spécifiques exécutant un ensemble de code

Une fonction est un ensemble de codes exécutant une tâche spécifique.

Question 2

La fonction `print()` est :

- ◉ Une fonction intégrée à la bibliothèque standard
- ○ Une fonction intégrée d'une bibliothèque spécifique
- ○ Une fonction créée par le programmeur

La fonction `print()` est une fonction intégrée à la bibliothèque standard, c'est-à-dire qu'il est possible d'y faire appel directement sans avoir à la créer nous-même.

Question 3

Pourquoi dois-je créer des fonctions ?

- ☐ Cela rend mon code plus rapide
- ☑ Cela rend mon code plus lisible
- ☑ Cela m'évite d'écrire plusieurs fois la même suite d'instructions

Malheureusement, l'utilisation de fonctions ne rendra pas votre code plus rapide ! Cependant, ce sera un vrai gain de temps car vous n'aurez pas à réécrire plusieurs fois un même ensemble de code, et cela facilitera la relecture.

Question 4

Quelles sont les deux propriétés principales d'une fonction ?

- ◉ Le résultat et les paramètres
- ○ Le type et les paramètres
- ○ Le résultat et le type

Les deux propriétés principales d'une fonction sont le résultat et les paramètres.

Question 5

Lorsque je fais appel à une fonction, à quoi dois-je faire attention ?

- [x] Aux types des paramètres
- [x] Au type du résultat
- [x] À l'ordre des paramètres

Il est important de faire attention aux trois éléments, ou cela pourrait mener à des erreurs. Si le type de paramètres d'une fonction n'est pas respecté, celle-ci ne pourra pas fonctionner correctement. L'ordre des paramètres ainsi que le type du résultat sont également importants.

Question 6

Il est obligatoire d'indiquer un type pour un paramètre.

- ○ Vrai
- ◉ Faux

Il n'est pas obligatoire d'indiquer un type pour un paramètre. C'est cependant recommandé pour plus de lisibilité, et éviter quelques erreurs.

Question 7

La syntaxe `arrow` permet :

- ◉ Une écriture simplifiée pour une fonction n'ayant qu'une seule instruction dans son corps
- ○ Une écriture simplifiée pour une fonction ayant plusieurs autres appels de fonctions dans son corps
- ○ Une écriture plus complexe offrant plus de possibilités

La syntaxe `arrow` est plus simple et est utilisée uniquement pour des fonctions n'ayant qu'une seule instruction dans leur corps.

Question 8

```
1 int bonjour(String i) {
2   return i
3 }
```

Cette fonction peut être utilisée.

- ○ Vrai
- ◉ Faux

Cette fonction ne peut pas être utilisée, car elle prend en paramètre une `String`, et retourne donc une `String`. Or, il est indiqué que le type de retour doit être un `int`.

Question 9

```
1 void aurevoir(int i) {
2   return i
3 }
```

Cette fonction peut être utilisée.

- ◉ Vrai
- ○ Faux

Cette fois-ci, c'est tout bon ! `void` permet de retourner tous les types, ou de ne retourner aucun résultat.

Question 10

```
String jemapelle(String prenom) {
  print(prenom);
}
```

Comment puis-je écrire cette fonction en syntaxe `arrow` ?

- ○ `jemapelle(prenom) => print(prenom);`
- ◉ `String jemapelle(prenom) => print(prenom);`
- ○ `void jemapelle(preynom) => print(prénom);`

La bonne syntaxe est la deuxième option. La première est fausse car le type de résultat n'est pas indiqué. Dans la dernière, le nom de variable en paramètre ne correspond pas au nom de la variable utilisée.

Exercice p. 9 Solution n°2

Question 1

```
void mafonction([String nomdulivre]);
```

Le paramètre `nomdulivre` :

- ◉ Est optionnel
- ○ Est optionnel nommé
- ○ Possède une valeur par défaut

Un paramètre entre crochets est un paramètre optionnel. Il n'est pas nécessaire de le renseigner dans l'appel à une fonction. Nous pouvons faire appel à la fonction comme suit : `mafonction()`.

Question 2

```
void mafonction({String nomdulivre});
```

Le paramètre `nomdulivre` :

- ○ Est optionnel
- ◉ Est optionnel nommé
- ○ Possède une valeur par défaut

Un paramètre entre `{}` est optionnel et nommé. Nous pouvons faire appel à la fonction comme suit : `mafonction()` ou `mafonction(nomdulivre: 'Shoe dog')`.

Question 3

```
void mafonction({String nomdulivre = 'Shoe dog'});
```

Le paramètre `nomdulivre` :

- ☐ Est optionnel
- ☑ Est optionnel nommé
- ☑ Possède une valeur par défaut

Un paramètre suivi d'une valeur possède une valeur par défaut. Il est également nommé. Nous pouvons faire appel à cette fonction comme suit : `mafonction()` ou `mafonction(nomdulivre: 'Shoe dog')`. Dans le second cas, le paramètre indiqué lors de l'appel l'emporte sur la valeur par défaut.

Question 4

```
1 void mafonction(String nomdulivre);
```

Le paramètre *nomdulivre* :

- ◉ Est optionnel
- ○ Est optionnel nommé
- ○ Possède une valeur par défaut

Le paramètre ne se situe ni entre `[]` ni entre `{}`, il est donc un paramètre obligatoire. Il ne possède pas non plus de nom.

Question 5

Quel est l'intérêt de donner des valeurs par défaut à des paramètres ?

- ○ Cela permet de ne jamais avoir d'erreur
- ◉ Cela permet d'avoir des fonctions qui marchent même si nous ne souhaitons pas nous prononcer sur certains paramètres
- ○ Cela permet une meilleure lisibilité

Avec des fonctions ayant des paramètres par défaut, nous n'avons pas à nous prononcer sur certains paramètres si nous ne souhaitons pas un résultat personnalisé.

Question 6

Je peux avoir plusieurs types de paramètres dans une même fonction.

- ◉ Vrai
- ○ Faux

Il est tout à fait possible d'avoir plusieurs types de paramètres dans une fonction. Par exemple : `void mafonction(String s, {int i, int j}, [int x])`.

Question 7

Si un paramètre est optionnel et ne possède pas de valeur par défaut, à quoi dois-je faire attention ?

- ◉ Je dois vérifier avant l'utilisation de ce paramètre qu'il n'est pas `null`.
- ○ Je dois vérifier que le paramètre est bien du bon type.
- ○ Je dois vérifier que le type du paramètre correspond au type du résultat.

Si un paramètre est optionnel, même si celui-ci n'est pas renseigné dans l'appel de la fonction, la fonction pourra s'exécuter. Mais attention au bon usage de la valeur de ce paramètre, qui peut donc être égale à `null` !

p. 10 Solution n°3

```
void chanson([String chanson]) {
  If (chanson != Null)
  {
    print("Ma chanson préférée est " + chanson)
  }
  Else {
    print("Je n'ai pas de chanson préférée");
   }
}
```

p. 10 Solution n°4

void afficher(cequejeveuxafficher) => print(cequejeveuxafficher.toString());

p. 11 Solution n°5

```
void jeregarde(String nomdufilm, {String durée = 'durée inconnue' }) {
  print("Je regarde " + nomdufilm + " de " + duree);
 }
```

Exercice p. 11 Solution n°6

Question 1

Pourquoi dois-je créer des fonctions ?

- ☐ Cela rend mon code plus rapide
- ☑ Cela rend mon code plus lisible
- ☑ Cela m'évite d'écrire plusieurs fois la même suite d'instructions

Malheureusement, l'utilisation de fonctions ne rendra pas votre code plus rapide ! Cependant, ce sera un vrai gain de temps car vous n'aurez pas à réécrire plusieurs fois un même ensemble de code, et cela facilitera la relecture.

Question 2

```
String aurevoir(int i) {
  return i
}
```

Cette fonction peut être utilisée.

- ○ Vrai
- ◉ Faux

Cette fonction ne peut pas être utilisée, car elle prend en paramètre un `int`, et retourne donc un `int`. Or, il est indiqué que le type de retour doit être une `String`.

Question 3

Quel est l'intérêt de donner des valeurs par défaut à des paramètres ?

- Cela permet de ne jamais avoir d'erreur
- Cela permet d'avoir des fonctions qui marchent même si nous ne souhaitons pas nous prononcer sur certains paramètres
- Cela permet une meilleure lisibilité

Avec des fonctions ayant des paramètres par défaut, cela nous permet de ne pas avoir à nous prononcer sur certains paramètres si nous ne souhaitons pas un résultat personnalisé.

Question 4

Je peux avoir plusieurs types de paramètres dans une même fonction.

- Vrai
- Faux

Il est tout à fait possible d'avoir plusieurs types de paramètres dans une fonction. Par exemple : `void mafonction(String s, {int i, int j}, [int x])`.

Question 5

Dans `void mafonction({String unnom = 'Isabelle'})`, quel est le type du paramètre ?

- Optionnel nommé avec valeur par défaut
- Optionnel nommé
- Obligatoire
- Optionnel de position

Cette fonction possède un paramètre optionnel avec une valeur par défaut. Cette valeur par défaut est `Isabelle`.