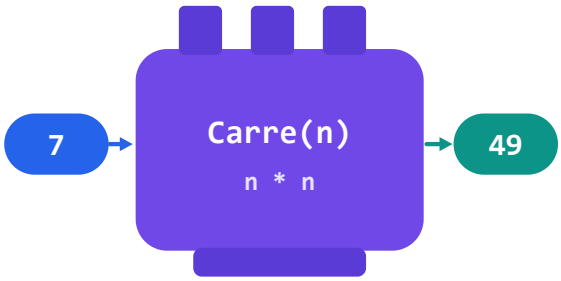
CHAPITRE 7

Les fonctions en C#

Écrire une fois, utiliser partout



AU PROGRAMME

1. Tu utilises déjà des fonctions
2. Les éléments d'une fonction
3. Paramètres et valeur de retour
4. Le passage par copie
5. Appeler les fonctions toutes faites

Nom : ..... Classe : .....

AVANT DE COMMENCER

Ce que tu vas apprendre

Quand un programme grandit, les mêmes morceaux reviennent : calculer un prix, vérifier une saisie, afficher un menu. En C#, une fonction regroupe ces instructions sous un nom ; on l'écrit une fois et on l'appelle autant de fois qu'il le faut.

1. **Expliquer ce qu'est une fonction et reconnaître celles que tu utilises déjà.**
2. **Citer les quatre éléments d'une fonction : type de retour, nom, paramètres, corps.**
3. **Écrire une fonction `static` et l'appeler depuis `Main`.**
4. **Expliquer le passage des paramètres par copie.**
5. **Utiliser `return` pour récupérer le résultat d'une fonction.**
6. **Écrire une fonction `void` qui ne retourne rien.**

PARTIE 1

Tu utilises déjà des fonctions

Les fonctions ne sont pas tout à fait nouvelles pour toi : tu en as déjà rencontré de deux sortes.

Les fonctions toutes faites

`Console.WriteLine()` et `Console.ReadLine()` sont des fonctions. Elles sont déjà écrites dans les bibliothèques du C# : tu les utilises sans savoir comment elles marchent à l'intérieur.

Celles que tu écris

Tout le code que tu places dans `static void Main()` est contenu dans une fonction. `Main` est la fonction **principale** : c'est elle qui est lancée en premier quand tu démarres le programme. C'est le **point d'entrée**.

DÉFINITION

Une fonction

Un ensemble d'instructions **regroupées sous un même nom**, qui réalise une tâche précise. On peut l'**appeler** plusieurs fois dans le programme, sans répéter son code partout.

Pourquoi découper en fonctions ?

AVANTAGES

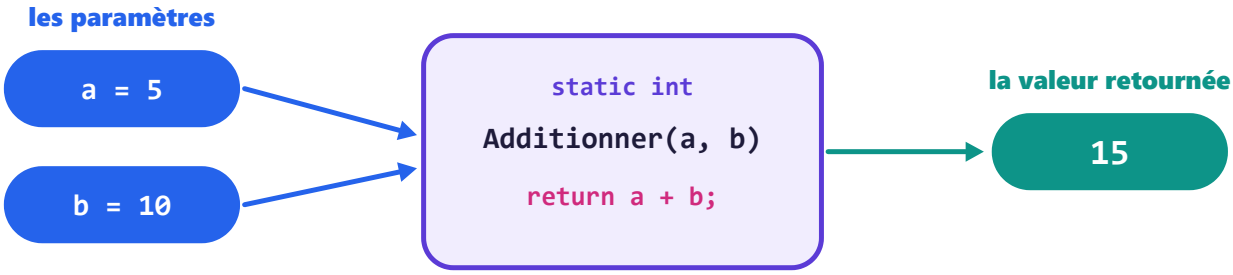
- Pas de copier-coller : le code est écrit une seule fois.
- Une correction se fait à un seul endroit.
- `Main` devient court et lisible.

INCONVÉNIENTS

- Il faut bien choisir le nom et les paramètres.
- Une fonction mal découpée est plus difficile à suivre.

PARTIE 2

Les éléments d'une fonction



On donne des valeurs à la fonction ; elle fait son travail et renvoie un résultat.

Figure 7.1 — Une fonction reçoit des valeurs (les paramètres) et renvoie un résultat.

c# La structure

```
typeDeRetour NomDeLaFonction(parametres)   // l'en-tête
{
    // le corps de la fonction
}
```

Élément	Rôle	Exemple
Le type de retour	le type de la valeur renvoyée : `int`, `string`, `bool`... ou `void` si la fonction ne renvoie rien	`int`
Le nom	un nom explicite qui dit ce que fait la fonction ; c'est un **identifiant** (mêmes règles que pour les variables)	`Additionner`
Les paramètres	facultatifs : les valeurs dont la fonction a besoin, entre parenthèses, chacune avec son type	`int a, int b`
Le corps	les instructions de la fonction, entre accolades `{ }`	`return a + b;`

C# Un premier exemple

```
using System;
class Program
{
    static void Main()
    {
        // appel d'une fonction préexistante
        Console.WriteLine("Bienvenue dans le monde des fonctions !");
        // appel de notre propre fonction
        SalutUtilisateur();
    }
    // définition de la fonction
    static void SalutUtilisateur()
    {
        Console.WriteLine("Bonjour, utilisateur !");
    }
}
```

À L'ÉCRAN

```
Bienvenue dans le monde des fonctions !
Bonjour, utilisateur !
```

`SalutUtilisateur()` est une fonction que **nous** avons définie : elle affiche un message. `Console.WriteLine()` est une fonction **préexistante**. Les deux s'appellent de la même façon : le nom, puis les parenthèses.

À RETENIR

Le mot-clé static

- Pour l'instant, chaque fonction que tu écris commence par `static`, comme `Main`.
- Les raisons techniques viendront plus tard : retiens seulement qu'il est obligatoire.

PARTIE 3

Paramètres et valeur de retour

C#

```
using System;
class Program
{
    static void Main()
    {
        int resultat = Additionner(5, 10);
        Console.WriteLine("Le résultat de l'addition est : " + resultat);
        Console.WriteLine("Et 2 + 3 = " + Additionner(2, 3));
    }
    // fonction qui additionne deux nombres
    static int Additionner(int a, int b)
    {
        return a + b;
    }
}
```

À L'ÉCRAN

```
Le résultat de l'addition est : 15
Et 2 + 3 = 5
```

`Additionner` prend deux entiers en entrée (`int a` et `int b`) et **retourne** leur somme, un `int`. Dans `Main`, l'appel `Additionner(5, 10)` est remplacé par la valeur retournée, 15, qu'on range dans `resultat`. On peut aussi utiliser l'appel directement dans un calcul ou un `WriteLine`.

DÉFINITION

return

Le mot-clé qui **renvoie** une valeur au programme qui a appelé la fonction. Il **met fin** à l'exécution de la fonction : ce qui le suit n'est jamais exécuté.

À RETENIR

Les règles de return

- La valeur retournée a le **type annoncé** devant le nom de la fonction.
- Dans ce cours, une fonction a **un seul** `return`, placé **tout à la fin** de ses instructions.
- Une fonction `void` ne retourne rien : pas de `return` avec une valeur.

	Fonction qui retourne	Fonction void
Type de retour	`int`, `double`, `string`, `bool`…	`void`
Fin de la fonction	`return valeur;`	rien à retourner
Appel	dans une affectation ou un calcul : `int c = Carre(4);`	seul sur sa ligne : `AfficherMenu();`

PARTIE 4

Le passage par copie

Quand on appelle une fonction, chaque paramètre est initialisé par une **copie** de la valeur transmise. La fonction travaille sur sa copie ; la variable d'origine **ne change pas**.

c# Ce programme affiche… 13,85 !

```
1  using System;
2
3  class Program
4  {
5      static void Main()
6      {
7          double noteBrute = 13.85;
8          Arrondir(noteBrute);
9          Console.WriteLine("Valeur : " + noteBrute);
10     }
11
12     static void Arrondir(double x)
13     {
14         x = x + 0.5;
15         x = (int)x;
16     }
17 }
```

À L'ÉCRAN

```
Valeur : 13,85
```

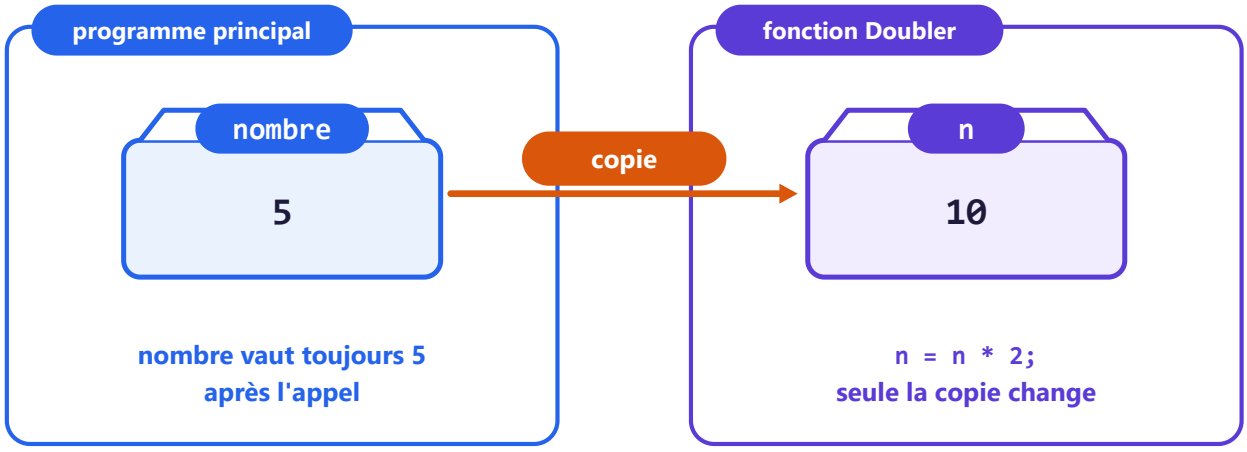


Figure 7.2 — Le paramètre reçoit une copie : modifier la copie ne touche pas l'original.

Ligne 8 : ce n'est pas la variable `noteBrute` qui entre dans la fonction, mais **sa valeur**, 13.85, copiée dans le paramètre `x`. La fonction modifie `x`… puis se termine, et `x` disparaît. La ligne 9 affiche donc toujours la note de départ.

POUR ALLER PLUS LOIN

Le C# affiche les nombres à virgule selon la langue de l'ordinateur : sur un PC réglé en français, 13.85 s'affiche « 13,85 ».

La solution : retourner le résultat

C#

```
using System;
class Program
{
    static void Main()
    {
        double noteBrute = 13.85;
        // on récupère la valeur retournée en l'affectant à noteArrondie
        int noteArrondie = Arrondir(noteBrute);
        Console.WriteLine("Valeur : " + noteArrondie);
    }
    static int Arrondir(double x)
    {
        int arrondi;
        x = x + 0.5;
        arrondi = (int)x;
        // on retourne la valeur arrondie
        return arrondi;
    }
}
```

À L'ÉCRAN

```
Valeur : 14
```

À RETENIR

Pour qu'un résultat sorte d'une fonction

- La fonction le **retourne** avec `return`.
- Le programme principal le **récupère** : `int noteArrondie = Arrondir(noteBrute);`.

PARTIE 5

Appeler les fonctions toutes faites

Appeler une fonction déjà définie est simple : son nom, suivi de parenthèses. Si elle attend des paramètres, on les donne entre les parenthèses, séparés par des virgules.

Appel	Ce qu'il fait	Résultat
`Console.WriteLine("Salut")`	affiche un texte et passe à la ligne	rien (`void`)
`Console.ReadLine()`	lit une ligne tapée au clavier	un `string`
`int.Parse("42")`	convertit un texte en entier	`42`
`Math.Max(7, 12)`	le plus grand de deux nombres	`12`
`Math.Abs(-5)`	la valeur absolue (sans le signe)	`5`
`Math.Sqrt(25)`	la racine carrée	`5`

C#

```
using System;
class Program
{
    static void Main()
    {
        int a = 7, b = 12;
        Console.WriteLine("Le plus grand : " + Math.Max(a, b));
        Console.WriteLine("Écart : " + Math.Abs(a - b));
    }
}
```

À L'ÉCRAN

```
Le plus grand : 12
Écart : 5
```

À RETENIR

Découper un programme

- Une fonction = une tâche, avec un nom qui la décrit : `CalculerMoyenne`, `AfficherMenu`, `EstPair`.
- `Main` organise le travail : il appelle les fonctions dans le bon ordre.

BILAN DU CHAPITRE 7

L'essentiel à retenir

1. Une **fonction** regroupe des instructions sous un nom ; on l'écrit une fois et on l'**appelle** autant de fois qu'on veut.
2. `Console.WriteLine()` et `Console.ReadLine()` sont des fonctions toutes faites ; `Main` est la fonction principale, le **point d'entrée**.
3. Une fonction a un **type de retour**, un **nom**, des **paramètres** (facultatifs) et un **corps** entre accolades.
4. En C#, chaque fonction du programme commence par `static`.
5. `return` renvoie la valeur et **termine** la fonction ; une fonction `void` ne retourne rien.
6. Les paramètres sont passés **par copie** : modifier un paramètre ne change pas la variable d'origine.
7. Pour récupérer un résultat : `int noteArrondie = Arrondir(noteBrute);`.

Le lexique

Fonction
Instructions regroupées sous un nom, qui réalisent une tâche précise.

Appel
Utiliser une fonction : son nom et ses valeurs entre parenthèses, `Additionner(5, 10)`.

Paramètre
Valeur reçue par la fonction, déclarée entre parenthèses avec son type.

Type de retour
Le type de la valeur renvoyée par la fonction (`int`, `string`… ou `void`).

return
Renvoie une valeur et termine la fonction.

void
« Vide » : la fonction ne retourne rien.

Passage par copie
Le paramètre reçoit une copie de la valeur : l'original ne change pas.

static
Mot-clé placé devant chaque fonction du programme.

Fonction préexistante
Fonction déjà écrite dans les bibliothèques du C# : `Console.WriteLine`, `Math.Max`…

Je fais le point

Coche une case par ligne : où en es-tu avant l'interro ?

Je suis capable de…	Oui	Presque	Pas encore
Citer les quatre éléments d'une fonction.	☐	☐	☐
Écrire une fonction `static` avec paramètres et `return`.	☐	☐	☐
Appeler une fonction et récupérer son résultat.	☐	☐	☐
Écrire une fonction `void`.	☐	☐	☐
Expliquer le passage par copie sur un exemple.	☐	☐	☐
Utiliser une fonction toute faite comme `Math.Max`.	☐	☐	☐

CHAPITRE 7 · EXERCICES

Teste tes connaissances

avant l'interro

COMMENT T'Y PRENDRE

1. **Relis la synthèse** : l'essentiel, le lexique et « Je fais le point ».
2. **Fais les exercices sans le cours**, comme à l'interro.
3. **Corrige-toi** avec la correction vue en classe ou sur le site.
4. **Reviens sur tes erreurs** : relis la partie du cours concernée.

LES 5 EXERCICES

1. **Le vocabulaire des fonctions**
2. **Lire un en-tête**
3. **Le bon mot**
4. **Passage par copie**
5. **Mes premières fonctions**

Exercice 1 Le vocabulaire des fonctions

▸ **Coche la bonne réponse.**

1. Quelle est la fonction lancée en premier dans un programme C# ?

☐ Start() ☐ Main() ☐ Program()

2. Que signifie `void` devant le nom d'une fonction ?

☐ elle retourne un entier ☐ elle ne retourne rien ☐ elle n'a pas de paramètre

3. Que se passe-t-il après un `return` ?

☐ la fonction continue ☐ la fonction se termine ☐ le programme s'arrête

4. Quel mot-clé met-on devant chaque fonction du programme, pour l'instant ?

☐ public ☐ static ☐ class

Exercice 2 Lire un en-tête

▸ **Pour chaque en-tête, complète le tableau.**

En-tête	Retourne	Nom	Paramètres
`static int Aire(int l, int L)`			
`static bool EstPair(int n)`			
`static void AfficherMenu()`			
`static double Moyenne(double a, double b)`			

Exercice 3 Le bon mot

▸ **Relie chaque morceau de code à ce qu'il est (écris la lettre).**

1	`n` dans `static int Carre(int n)` →	**A**	un appel
2	`Carre(7)` →	**B**	renvoie le résultat
3	`return n * n;` →	**C**	le type de retour
4	`int` devant `Carre` →	**D**	un paramètre

Exercice 4 Passage par copie

▸ **Lis le programme, puis réponds.**

```c#
using System;
class Program
{
    static void Main()
    {
        int vies = 3;
        PerdreVie(vies);
        Console.WriteLine("Vies : " + vies);
    }
    static void PerdreVie(int v)
    {
        v = v - 1;
    }
}
```

1. Qu'affiche ce programme ?

2. Pourquoi ?

3. Comment corriger pour qu'il affiche 2 ?

Exercice 5 Mes premières fonctions

▸ **Écris le programme C# complet : using, class, Main, la fonction, accolades et points-virgules.**

1. Écris une fonction `Carre` qui reçoit un entier et retourne son carré. Dans `Main`, demande un nombre, appelle `Carre` et affiche le résultat.

EXEMPLE D'ÉCRAN

```
Un nombre ?
7
Son carré vaut 49
```

2. Écris une fonction `EstPair` qui reçoit un entier et retourne `true` s'il est pair, `false` sinon (un seul `return`). Dans `Main`, teste-la avec le nombre 10.

EXEMPLE D'ÉCRAN

```
10 est pair : True
```